035, 036

13.Most dominant colors in an image using KMeans clustering

#Question[Roll Nos. 035,036] Step 1 – Importing required libraries. Step 2 – Read and describe the data Step 3 – Define the no. of clusters for the KMeans algorithm. Step 4 – Read the input image and print its shape Step 5 – Resize and flatten images to get results fast and print them Step 6 – Make a KMeans Clustering object with n_clusters set to 5 and extract cluster centers Step 7 – We have to calculate the dominance of each dominant color. Step 8 – We have to zip percentages and colors together in tuples. Step 9 – Sort this zip object in descending order Step 10 - Plotting out the results of most dominant colors Step 11- We need to create and save the final output image.

Overview and description:

Importing packages required to find most dominant colors in an image.

Defining the no. of clusters for the KMeans algorithm.

Reading our input image.

Keeping a copy of it for future use.

Printing its shape.

Resizing our image to get results fast.

Printing resized image shape.

Flattening the image. In this step, we are just keeping all the columns of the image after each other to make just one column out of it.

After this step, we will be left with just 1 column and rows equal to the no. of pixels in the image.

Flattened shape becomes 200*293 = 58600 Making a KMeans Clustering object with n_clusters set to 5 as declared.

Fit our image in Kmeans Clustering Algorithm. In this step, the flattened image is working as an array containing all the pixel colors of the image.

These pixel colors will now be clustered into 5 groups.

These groups will have some centroids which we can think of as the major color of the cluster

We are extracting these cluster centers. Now we know that these 5 colors are the dominant colors of the image but still, we don't know the extent of each color's dominance.

We are calculating the dominance of each dominant color. np.unique(kmeans.labels_,return_counts=True), this statement will return an array with 2 parts, first part will be the predictions like [2,1,0,1,4,3,2,3,4…], means to which cluster that pixel belongs and the second part will contain the counts like [100,110,310,80,400] where 100 depicts the no. of pixels belonging to class 0 or cluster 0(our indexing starts from 0), and so on, and then we are simply dividing that array by the total no. of pixels, 1000 in the above case, so the percentage array becomes [0.1,0.11,0.31,0.08,0.4]

We are zipping percentages and colors together like, [(0.1,(120,0,150)), (0.11,(230,225,34)), …]. It will consist of 5 tuples. First tuple is (0.1, (120,0,150)) where first part of the tuple (0.1) is the percentage and (120,0,150) is the color.

Sort this zip object in descending order. Now the first element in this sorted object will be the percentage of the most dominant colors in the image and the color itself. We are plotting blocks of dominant colors.

We are plotting the following bar.

We are creating the final result.

We are saving the output image.

Step 1 – Importing required libraries.

```
1
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from sklearn.cluster import KMeans
6 import imutils
```

Step 2 – Read and describe the data

Step 3 – Define the no. of clusters for the KMeans algorithm.

Step 4 – Read the input image and print its shape

Step 5 – Resize and flatten images to get results fast and print them

```
clusters = 5 # try changing it

img = cv2.imread('/content/sample_data/img2.jpg')
org_img = img.copy()
print('Org image shape --> ',img.shape)

img = imutils.resize(img,height=200)
print('After resizing shape --> ',img.shape)

flat_img = np.reshape(img,(-1,3))
print('After Flattening shape --> ',flat_img.shape)
```

```
Org image shape -->  (484, 800, 3)
After resizing shape -->  (200, 330, 3)
After Flattening shape -->  (66000, 3)
```

Step 6 – Make a KMeans Clustering object with n_clusters set to 5 and extract cluster centers

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

```
kmeans = KMeans(n_clusters=clusters,random_state=0)
kmeans.fit(flat_img)

```

```
KMeans(n_clusters=5, random_state=0)
```

Step 7 – We have to calculate the dominance of each dominant color.

```
dominant_colors = np.array(kmeans.cluster_centers_,dtype='uint')

```

Step 8 – We have to zip percentages and colors together in tuples.

```

percentages = (np.unique(kmeans.labels_,return_counts=True)[1])/flat_img.shape[0]
p_and_c = zip(percentages,dominant_colors)
p_and_c = sorted(p_and_c,reverse=True)
```

Step 9 – Sort this zip object in descending order

```
block = np.ones((50,50,3),dtype='uint')
plt.figure(figsize=(12,8))
for i in range(clusters):
    plt.subplot(1,clusters,i+1)
    block[:] = p_and_c[i][1][::-1] # we have done this to convert bgr(opencv) to rgb(matplotlib)
    plt.imshow(block)
    plt.xticks([])
    plt.yticks([])
    plt.xlabel(str(round(p_and_c[i][0]*100,2))+'%')
```

```
bar = np.ones((50,500,3),dtype='uint')
plt.figure(figsize=(12,8))
plt.title('Proportions of colors in the image')
start = 0
i = 1
for p,c in p_and_c:
    end = start+int(p*bar.shape[1])
    if i==clusters:
        bar[:,start:] = c[::-1]
    else:
        bar[:,start:end] = c[::-1]
    start = end
    i+=1


plt.imshow(bar)
plt.xticks([])
plt.yticks([])
```

```
([], <a list of 0 Text major ticklabel objects>)
```

Step 10 - Plotting out the results of most dominant colors

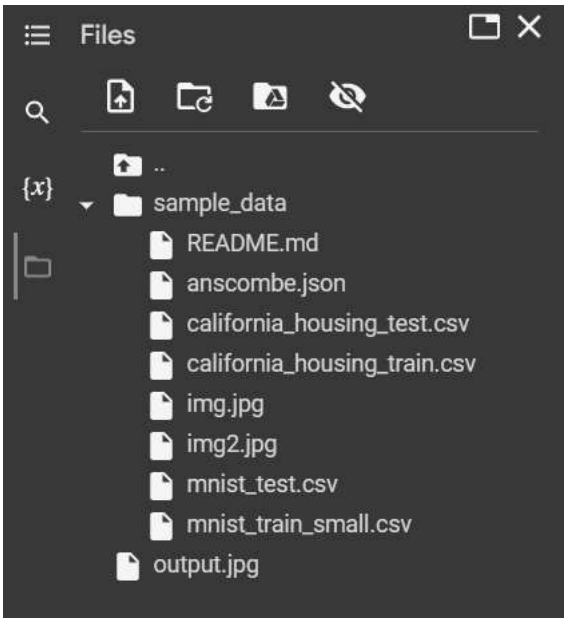
```

rows = 1000
cols = int((org_img.shape[0]/org_img.shape[1])*rows)
img = cv2.resize(org_img,dsize=(rows,cols),interpolation=cv2.INTER_LINEAR)

copy = img.copy()
cv2.rectangle(copy,(rows//2-250,cols//2-90),(rows//2+250,cols//2+110),(255,255,255),-1)

final = cv2.addWeighted(img,0.1,copy,0.9,0)
cv2.putText(final,'Most Dominant Colors in the Image',(rows//2-230,cols//2-40),cv2.FONT_HERSHEY_DUPLEX,0.8,(0,0,0),1,cv2.
```

Step 11- We need to create and save the final output image.

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. cv2.imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix, im_show is used to display the image. imwrite method is used to write an image into a particular file.

```

start = rows//2-220
for i in range(5):
    end = start+70
    final[cols//2:cols//2+70,start:end] = p_and_c[i][1]
    cv2.putText(final,str(i+1),(start+25,cols//2+45),cv2.FONT_HERSHEY_DUPLEX,1,(255,255,255),1,cv2.LINE_AA)
    start = end+20

from google.colab.patches import cv2_imshow

cv2_imshow(img)

cv2.imwrite('output.jpg',final)
```

```
True
```




```
1 print('Showing the dominating colors :')
2 cv2_imshow(final)
3
```

```
Showing the dominating colors :
```