ARTIFICIAL INTELLIGENCE (UCS411)

PROJECT

SUBGROUP: 2CO7

GROUP NUMBER: 12

GROUP MEMBERS: RAVINSHU KUSHWAHA (102003156)

SANYAM SOOD (102003163)

TOPIC: COLOUR DETECTION IN IMAGES

CODE:

In [26]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import imutils

clusters = 6
img = cv2.imread('./yellow.jpg')     #Chose a random image
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)
kmeans = KMeans(n_clusters=clusters,random_state=0)
kmeans.fit(flat_img)
dominant_colors = np.array(kmeans.cluster_centers_,dtype='uint')
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)
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([])
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//2-350,cols//2-90),(rows//2+350,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.LINE_AA
start = rows//2-305
for i in range(clusters):
    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
plt.show()
cv2.imshow('img',final)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite('E:/4Th SEM/AI/PROJECT/1.jpg',final)
```

IDEA: We have made use of K-Means algorithm to detect colours in an image

K-Means Algorithm: K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabelled 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. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training. It is a centroid-based algorithm, where each cluster is associated with a centroid.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K centre points or centroids by an iterative process.
- Assigns each data point to its closest k-centre. Those data points which are near to the particular k-centre, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

CODE DESCRIPTION

Libraries used:

- OpenCV – Used for image/video processing and computer vision tasks. (cv2 is the module import name)
- Numpy – Used to perform wide varieties of mathematical operations on arrays.
- Matplotlib.pyplot – Pyplot is a module in matplotlib. It is a collection of functions like creating a plotting area, plotting lines, creating a figure, adding plot labels etc.
- Scikit-learn(sklearn) – Used to import Kmeans clustering tool. Kmeans algorithm is an iterative algorithm that tries to partition the dataset into *K* pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group**.**

- Imutils: Imutils is a package based on OpenCV, which can call OpenCV interface more simply. It can easily realize a series of operations such as image translation, rotation, scaling, resizing etc.

Code explanation:

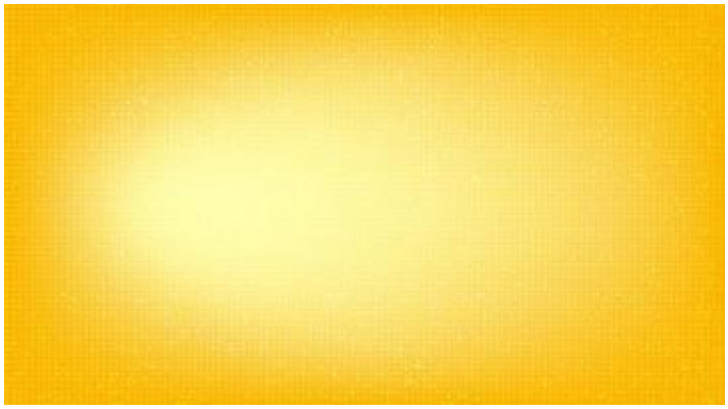
- Imported above stated libraries.
- Then we defined a value for clusters which can be changed as per choice. It is basically number of groups/clusters of pixels we want or simply the number of colours we want the program to detect.
- We then resized the image to generate faster results. It will be helpful if the image is large. Also 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.
- Making a **KMeans Clustering** object with n_clusters initially set to 7 as declared in the starting. We fit our image in Kmeans Clustering Algorithm. In this step, the flattened image is working as an array containing all the pixel colours of the image. These pixel colours will now be clustered into 7 groups. These groups will have some centroids which we can think of as the major colour of the cluster (In Layman's terms we can think of it as the boss of the cluster).
- We are extracting these cluster centres. Now we know that these 7 colours are the dominant colours of the image but still, we don't know the extent of each colour's dominance. So we are calculating the dominance of each dominant colour.
  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 colours together like, **[(0.1, (120,0,150)), (0.11, (230,225,34)), …]**. It will consist of 7 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 colour**.

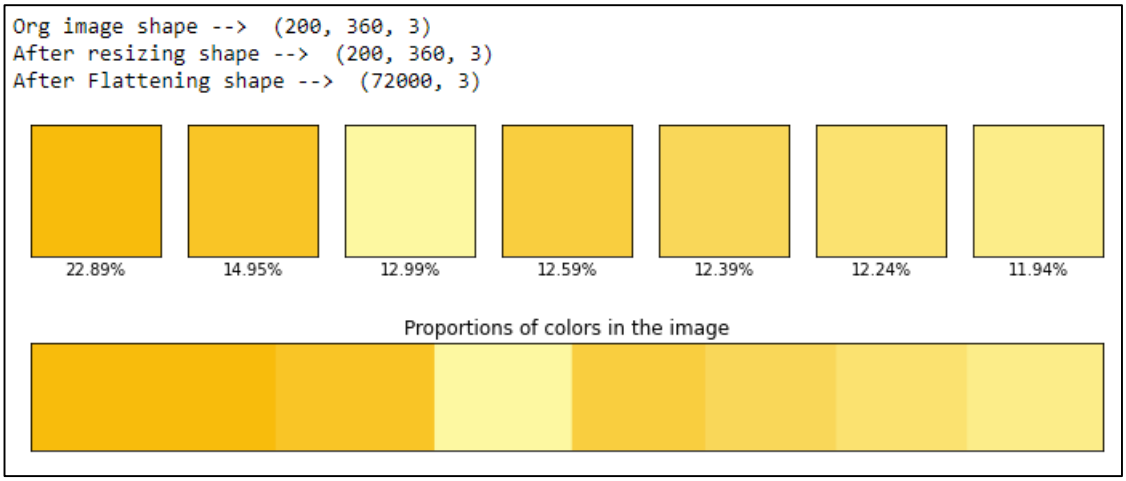
- Sorted this zip object in descending order. Now the first element in this sorted object will be the percentage of the most dominant colours in the image and the colour itself.
- We then plot blocks of dominant colours and the following bar.
- In the end we save the result in the desired folder.

OUTPUT

1) Example 1



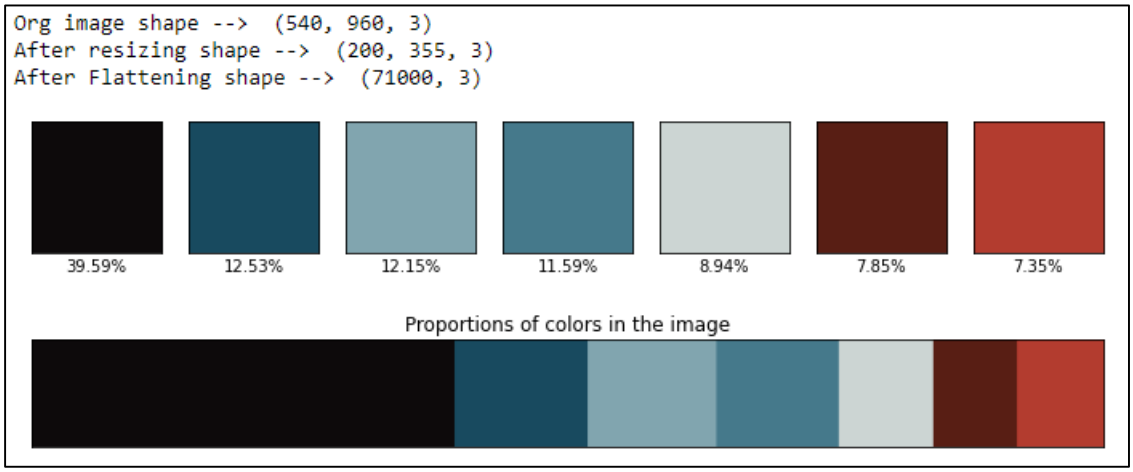
(Original Image)



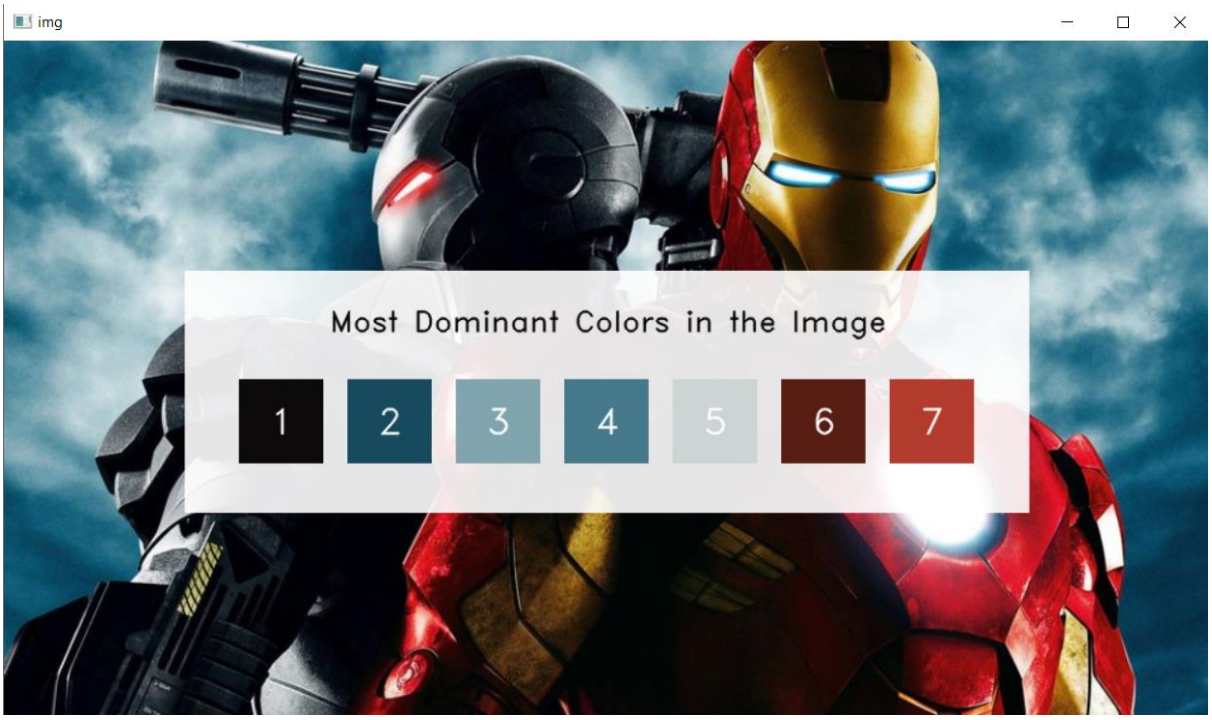
(Colour proportions)

2) Example 2

(Original Image)



(Colour proportions)


img
Most Dominant Colors in the Image
1
2
3
4
5
6
7