**Color Quantization using K-Means**

The goal is to partition n data points into k clusters. Each of the n data points will be assigned to a cluster with the nearest mean. The mean of each cluster is called its “centroid” or “center”

Overall, applying k-means yields k separate clusters of the original n data points. Data points inside a particular cluster are considered to be “more similar” to each other than data points that belong to other clusters.

In our case, we will be clustering the pixel intensities of a RGB image. Given a MxN size image, we thus have MxN pixels, each consisting of three components: Red, Green, and Blue respectively.

We will treat these MxN pixels as our data points and cluster them using k-means.

Pixels that belong to a given cluster will be more similar in color than pixels belonging to a separate cluster.

One caveat of k-means is that we need to specify the number of clusters we want to generate **ahead of time**.

I used Spyder for this project.

# OpenCV and Python K-Means Color Clustering:

cluster pixel intensities using OpenCV, Python, and k-means:

# importing the packages

import numpy as np

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import argparse

import cv2

# creating centroid and clustering the colors

def centroid\_histogram(clt):

numLabels = np.arange(0, len(np.unique(clt.labels\_)) + 1)

(hist, \_) = np.histogram(clt.labels\_, bins = numLabels)

hist = hist.astype("float")

hist /= hist.sum()

return hist

def plot\_colors(hist, centroids):

bar = np.zeros((50, 300, 3), dtype = "uint8")

startX = 0

for (percent, color) in zip(hist, centroids):

endX = startX + (percent \* 300)

cv2.rectangle(bar, (int(startX), 0), (int(endX), 50),

color.astype("uint8").tolist(), -1)

startX = endX

return bar

# construct the argument parser and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("image", help = "Path to the image")

ap.add\_argument("clusters", type = int,help = "# of clusters")

args = ap.parse\_args()

# load the image and convert it from BGR to RGB so that

# we can dispaly it with matplotlib

image = cv2.imread(args.image)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# show our image

plt.figure()

plt.axis("off")

plt.imshow(image)

# reshape the image to be a list of pixel

image = image.reshape((image.shape[0] \* image.shape[1], 3))

# cluster the pixel intensities

clt = KMeans(n\_clusters = args["clusters"])

clt.fit(image)

hist = centroid\_histogram(clt)

bar = plot\_colors(hist, clt.cluster\_centers\_)

plt.figure()

plt.axis("off")

plt.imshow(bar)

plt.show()

To execute our script, issue the following command:

F:\ML work\Color Clustering>python color\_Cluster.py image.png 3

Here is the actual image:-



Result Image:



F:\ML work\Assignment\Color Clustering>python color\_Cluster.py pixel.jpeg 4

Here is the actual image:-



Result Image:

