```
# -*- coding: utf-8 -*-
"""hw6_sara_ghavampour_9812762781.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1ZuYvNYj7VqAQmEeUWVrUinwhV-Ff9F1g
"""

############## Sara Ghavampour 9812762781 ########

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import cv2
from math import *
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
# %matplotlib inline

def show_img(*args, figsize=10, is_gray=True, title=None, fontsize=12):
    if isinstance(figsize, int):
        figsize = (figsize, figsize)
    images = args[0] if type(args[0]) is list else list(args)
    cmap=None
    if not is_gray:
        images = list(map(lambda x: cv2.cvtColor(x, cv2.COLOR_BGR2RGB), images))
    else:
        cmap = 'gray'
    plt.figure(figsize=figsize)
    for i in range(1, len(images)+1):
        plt.subplot(1, len(images), i)
        if title is not None:
            plt.title(title[i-1], fontsize=fontsize)

        plt.imshow(images[i-1], cmap=cmap)
        plt.axis('off')

lena_img=cv2.imread('Lena.bmp')
lena_img = cv2.cvtColor(lena_img,cv2.COLOR_BGR2RGB)
plt.imshow(lena_img)
plt.axis('off')
plt.show()
#(512, 512) uint8

baboon_img=cv2.imread('Baboon.bmp')
baboon_img = cv2.cvtColor(baboon_img,cv2.COLOR_BGR2RGB)
plt.imshow(baboon_img,cmap='gray')
plt.axis('off')
plt.show()
#(512, 512) uint8

def normalize(img):
  min = np.min(img)
  max = np.max(img)
  return ((img-min)/(max-min)*255).astype('uint8')

"""
6.1.1"""

############## 6.1.1 ########################

def HSI_converter(img):
  red,green,blue = img[:,:,0],img[:,:,1],img[:,:,2]
  # scale rgb between 0-1
  red =  red / 255.0
  green =green / 255.0
  blue =   blue /255.0

  H = np.zeros((img.shape[0],img.shape[1]))
  S = np.zeros((img.shape[0],img.shape[1]))

# theta formula in page 62 of slide 7
  hue_numerator = 1/2 * ((red-green)+(red-blue))
  hue_denumerator = np.sqrt((red-green)**2 + ((red-blue)*(green - blue)))
  hue = np.arccos(hue_numerator / hue_denumerator)

  # 2 conditions for  H
  for i in range (img.shape[0]):
    for j in range(img.shape[1]):
      if hue_denumerator[i,j]  != 0 :
        if blue[i,j]<= green[i,j] : H[i,j]=hue[i,j]
        if blue[i,j]> green[i,j] : H[i,j]=np.pi- hue[i,j]
      ###################################
      ## compute s
      mean = min(min(red[i,j],green[i,j]),blue[i,j])
      S[i,j]= 1-(3/(red[i,j]+green[i,j]+blue[i,j]))*mean

  # compute I
  I = (red + green + blue) / 3

  H = normalize(H)
  return H,S,I

H_lena,S_lena ,I_lena = HSI_converter(lena_img)

show_img(H_lena)

show_img(S_lena)

show_img(I_lena)

"""6.2.1"""

#################### 6.2.1 ##########

def mse(img1,img2):
  pic_1=img1.copy().ravel().astype('uint8')
  pic_2=img2.copy().ravel().astype('uint8')
  sqr_diff=(pic_1 - pic_2)**2
  size=pic_1.shape[0]
  err = np.sum(sqr_diff) / size
  return err

def Quantize_channel(img,n_final__levels):
```

```
  output= np.floor(img / (256/n_final__levels))
  output= output * np.floor(255/(n_final__levels -1 ))
  return output

def Quantize_colored(img,r_level,g_level,b_level):
  r_channel , g_channel , b_channel = img[:,:,0] , img[:,:,1],img[:,:,2]
  # quantise each color channel speratly
  r_channel = Quantize_channel(r_channel,r_level)
  g_channel = Quantize_channel(g_channel,g_level)
  b_channel = Quantize_channel(b_channel,b_level)

  quantized_img = np.zeros((img.shape))
  quantized_img[:,:,0] = r_channel
  quantized_img[:,:,1] = g_channel
  quantized_img[:,:,2] = b_channel

  quantized_img=quantized_img.astype('uint8')

  psnr  = cv2.PSNR(img,quantized_img)

  return quantized_img , mse(img.astype('uint8'),quantized_img) , psnr

quantized_lena_8,mse_8,psnr_8 = Quantize_colored(lena_img,8,8,8)
quantized_lena_16,mse_16,psnr_16 = Quantize_colored(lena_img,16,16,16)
quantized_lena_32,mse_32,psnr_32 = Quantize_colored(lena_img,32,32,32)
quantized_lena_64,mse_64,psnr_64 = Quantize_colored(lena_img,64,64,64)

print('mse_8: ',mse_8,' psnr_8: ',psnr_8)
show_img(quantized_lena_8)

print('mse_16: ',mse_16,' psnr_16: ',psnr_16)
show_img(quantized_lena_16)

print('mse_32: ',mse_32,' psnr_32: ',psnr_32)
show_img(quantized_lena_32)

print('mse_64: ',mse_64,' psnr_64: ',psnr_64)
show_img(quantized_lena_64)

"""6.2.2"""

################## 6.2.2 ######################

# 3,3,2 bits for r,g,b --> 8,8,4 levels for r,g,b
q_622_img,q_622_mse,q_622_psnr = Quantize_colored(lena_img,8,8,4)
print('q_622_mse: ',q_622_mse,' q_622_psnr: ',q_622_psnr)
show_img(q_622_img)

"""6.2.3"""

############# 6.2.3 ###################

def safe_rgb(img,colors_count):
  kmeans = KMeans(colors_count,random_state=0)
  clustered_img = np.reshape(img, (img.shape[0] * img.shape[1], img.shape[2]))
  kmeans.fit_predict(clustered_img)

  cluster_centroids = kmeans.cluster_centers_ .astype('uint8')
  labels = kmeans.labels_.reshape(img.shape[0],img.shape[1])

  segmented_pic = np.zeros((img.shape))

  for i in range(labels.shape[0]):
    for j in range(labels.shape[1]):
      segmented_pic[i,j,:]=cluster_centroids[labels[i,j]].astype('uint8')


  return segmented_pic.astype('uint8') , mse(img,segmented_pic.astype('uint8'))

segmented_baboon_8,mse_baboon_8 = safe_rgb(baboon_img,8)
segmented_baboon_16,mse_baboon_16 = safe_rgb(baboon_img,16)
segmented_baboon_32,mse_baboon_32 = safe_rgb(baboon_img,32)

print('mse_baboon_8 : ', mse_baboon_8)
show_img(segmented_baboon_8)

print('mse_baboon_16 : ', mse_baboon_16)
show_img(segmented_baboon_16)

print('mse_baboon_32 : ', mse_baboon_32)
show_img(segmented_baboon_32)

################ The End of hw6 ########
```