```
#
#
#

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import numpy.linalg as LA
import argparse
import  scipy.misc   as misc
from sklearn import decomposition
from sklearn.cluster import KMeans
from skimage import io
import cv2
import time
from sklearn import cluster, datasets, mixture



def reconstr_matrix(U,D,V,k):
    rec_mat=np.dot(U[:,:k],np.dot(D[:k,:k],V[:k,:]))
    return rec_mat


def ParseArguments():
    parser = argparse.ArgumentParser(description="Project ")
    parser.add_argument('--image', default="", required=True, help='image  (default: %(default)s)')
    parser.add_argument('--alg', default="kmeans", required=False, help='which alg  (default: %(default)s)')
    parser.add_argument('--k', default="2", required=False, help='Number of clusters (default: %(default)s)' )
    args = parser.parse_args()
    return args.image,args.alg, args.k


image_file, alg, nr_of_clusters  =  ParseArguments()

nr_of_clusters=int(nr_of_clusters)
image = io.imread(image_file)




fig_orig = plt.figure(1)
ax_orig = fig_orig.add_subplot(111)
ax_orig.imshow(image)
ax_orig.set_title("Original image")

imageYCC=cv2.cvtColor(image,cv2.COLOR_BGR2YCR_CB)

fig_YCC = plt.figure(2)
ax_YCC = fig_YCC.add_subplot(111)
ax_YCC.imshow(imageYCC[:,:,2])
ax_YCC.set_title("Image converted to: YCrCb ")


image_height=image.shape[0]
image_width=image.shape[1]

image_points=image.reshape(-1,3)

print("Calculating kmeans (RGB)...", end="", flush=True)
start_time = time.time()
kmeansRGB = KMeans(n_clusters=nr_of_clusters)
kmeansRGB.fit(image_points)
clusters_kmeansRGB = kmeansRGB.predict(image_points)
print("\t\t took %s seconds " % round((time.time() - start_time),5))

image_clusteredRGB = clusters_kmeansRGB.reshape(image_height,image_width)


fig_kmeansRGB = plt.figure(3)
ax_kmeansRGB = fig_kmeansRGB.add_subplot(111)
ax_kmeansRGB.imshow(image_clusteredRGB,cmap='gray')
ax_kmeansRGB.set_title("Kmeans rgb: k =" +str( nr_of_clusters))
```

```
imageYCC_points=imageYCC.reshape(-1,3)

imageYCC_points=imageYCC_points[:,2]
imageYCC_points=imageYCC_points.reshape(-1,1)


print("Calculating kmeans (YCrCb )...", end="", flush=True)
start_time = time.time()
kmeansYCC = KMeans(n_clusters=nr_of_clusters)
kmeansYCC.fit(imageYCC_points)
clusters_kmeansYCC = kmeansYCC.predict(imageYCC_points)
print("\t\t took %s seconds " % round((time.time() - start_time),5))

image_clusteredYCC = clusters_kmeansYCC.reshape(image_height,image_width)



fig_kmeansYCC = plt.figure(4)
ax_kmeansYCC = fig_kmeansYCC.add_subplot(111)
ax_kmeansYCC.imshow(image_clusteredYCC,cmap='gray')
ax_kmeansYCC.set_title("Kmeans YCrCb (only on Cr channel): k =" +str(nr_of_clusters))



#GMM
#GMM
print("Calculating GMM (RGB)...", end="", flush=True)
gmmRGB =  mixture.GaussianMixture(n_components=nr_of_clusters)
gmmRGB.fit(image_points)
clusters_gmmRGB = gmmRGB.predict(image_points)
print("\t\t took %s seconds " % round((time.time() - start_time),5))

image_clustered_gmmRGB = clusters_gmmRGB.reshape(image_height,image_width)

fig_gmmRGB = plt.figure(5)
ax_gmmRGB = fig_gmmRGB.add_subplot(111)
ax_gmmRGB.imshow(image_clustered_gmmRGB)#,cmap='gray')
ax_gmmRGB.set_title("GMM rgb: k =" +str( nr_of_clusters))




#GMM
print("Calculating GMM (YCrCb)...", end="", flush=True)
gmmYCC =  mixture.GaussianMixture(n_components=nr_of_clusters)
gmmYCC.fit(imageYCC_points)
clusters_gmmYCC = gmmYCC.predict(imageYCC_points)
print("\t\t took %s seconds " % round((time.time() - start_time),5))

image_clustered_gmmYCC  = clusters_gmmYCC .reshape(image_height,image_width)

fig_gmmYCC = plt.figure(6)
ax_gmmYCC = fig_gmmYCC.add_subplot(111)
ax_gmmYCC.imshow(image_clustered_gmmYCC)#,cmap='gray')
ax_gmmYCC.set_title("GMM YCrCb (channel Cr): k =" +str( nr_of_clusters))




plt.show()

quit()
print("Calculating SVD ...", end="", flush=True)
start_time = time.time()
U,d,V=LA.svd(M)
print("\t\t took %s seconds " % round((time.time() - start_time),5))



h=M.shape[0]
w=M.shape[1]

#d jest wektorem, zamieniamy w macierz diagonalna
D=np.diag(d)

```

```
ile=4;
nr=1;

how_much_rec=1;

rr=reconstr_matrix(U,D,V,how_much_rec)

error_x = []
error_y = []
eigen_sum = []




for i in range(0,ile):
    for j in range(0, ile):
        fig1=plt.figure(1);
        plt.subplot(ile,ile,nr);

        #zrekonstruuj macierz uzywajac how_much_rec wektorow wlasnych
        rr=reconstr_matrix(U,D,V,how_much_rec)
        #liczymy bledy
        error_x.append(how_much_rec)
        err = ((M-rr)**2).sum();
        error_y.append(err)
        eigen_sum.append((d[how_much_rec:len(d)]**2).sum())
        #rysuj rr
        plt.imshow(rr, cmap=plt.get_cmap('gray'))
        plt.title("ile = " + str(how_much_rec))
        plt.axis('off')
        how_much_rec+=2;

        nr=nr+1
fig1.suptitle("PCA reconstructed")


nr=1;
ile=4
how_much_rec=0
for i in range(0,ile):
    for j in range(0, ile):
        how_much_rec+=1;
        fig2=plt.figure(2);
        plt.subplot(ile,ile,nr);
        k=how_much_rec
        rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
        plt.imshow(rr_eigenimage, cmap=plt.get_cmap('gray'))
        plt.title("i = " + str(how_much_rec))
        plt.axis('off')
        nr=nr+1

fig2.suptitle("PCA eigenimages")




#osobny rys. z bledami
plt.figure(3)
plt.plot(error_x,np.zeros(len(error_x)), color='red');
plt.plot(error_x,error_y, color='blue')

plt.plot(error_x,np.zeros(len(error_x)), color='red');
plt.plot(error_x,eigen_sum, color='green')


plt.figure(4)
plt.imshow(M, cmap=plt.get_cmap('gray'))
plt.title("Original (grayscale)")




#NMF:

# ~ plt.figure(5)
# ~ plt.title("NMF")
# ~ M_recons=np.dot(W,H)
# ~ plt.imshow(M_recons, cmap=plt.get_cmap('gray'))
```

```
239
240
241 if(alg=="nmf" or alg=="NMF"):
242
243     #p_red=min(int(h/2),int(w/2))
244     p_red=min(int(h),int(w))
245
246     print("Calculating NMF...", end="", flush=True)
247     start_time = time.time()
248     model = decomposition.NMF(n_components=p_red, init='random', random_state=0)
249     W=model.fit_transform(M)
250     H=model.components_
251     print("\t\t took %s seconds " % round((time.time() - start_time),5))
252
253
254     nr=1;
255     ile=4
256     how_much_rec=0
257     for i in range(0,ile):
258         for j in range(0, ile):
259             how_much_rec+=8;
260             fig5=plt.figure(5);
261             plt.subplot(ile,ile,nr);
262             k=how_much_rec
263             rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
264             M_recons=np.dot(W[:,0:k],H[0:k,:])
265             plt.imshow(M_recons, cmap=plt.get_cmap('gray'))
266             plt.title("i = " + str(how_much_rec))
267             plt.axis('off')
268             nr=nr+1
269
270     fig5.suptitle("NMF reconstructed")
271
272
273     nr=1;
274     ile=4
275     how_much_rec=0
276     for i in range(0,ile):
277         for j in range(0, ile):
278             how_much_rec+=1;
279             fig6=plt.figure(6);
280             plt.subplot(ile,ile,nr);
281             k=how_much_rec
282             rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
283             M_recons=np.dot(W[:,(k-1):k],H[(k-1):k,:])
284             plt.imshow(M_recons, cmap=plt.get_cmap('gray'))
285             plt.title("i = " + str(how_much_rec))
286             plt.axis('off')
287             nr=nr+1
288
289     fig6.suptitle("NMF eigenimages")
290
291
292
293
294
295
296 plt.show()
297
```