TP_Unsupervised_2_DermoSegmentation

November 25, 2020

1 Skin lesion segmentation

In this section, you will use the algorithm of k-means to segment skin lesion images. You will use two images from the ISIC dataset (www.isic-archive.com), one nevus and one melanoma with their respetive manual segmentation.

1.1 Goal

The goal of this section is to delineate the contours (i.e. segment) of the skin lesions using k-means.

Please complete the code where requested and answer all the questions

Let's first load the needed libraries

[1]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
plt.close('all')
import optuna
from skimage.io import imread
from skimage.transform import resize
from skimage.transform import rescale
from skimage import img_as_bool
from mpl_toolkits.axes_grid1 import AxesGrid
from sklearn.cluster import KMeans
from scipy.spatial.distance import dice
from skimage.measure import find_contours
```

Let's load the data

[2]:
```
# if 'google.colab' in str(get_ipython()):
#   from google_drive_downloader import GoogleDriveDownloader as gdd
#   gdd.
 ↪download_file_from_google_drive(file_id='1_TeYzLLDoKbPX4xXAOAM_mQiT2nLHgvp',
#   dest_path='./data/nevus.jpg')
#   gdd.
 ↪download_file_from_google_drive(file_id='1iQZdUiuK_FwZ7mik7LB3eN_H_IUc5l7b',
#   dest_path='./data/nevus-seg.jpg')
#   gdd.
 ↪download_file_from_google_drive(file_id='1yZ46UzGhwO7g5T8397JpewBl6UqgRo5J',
```

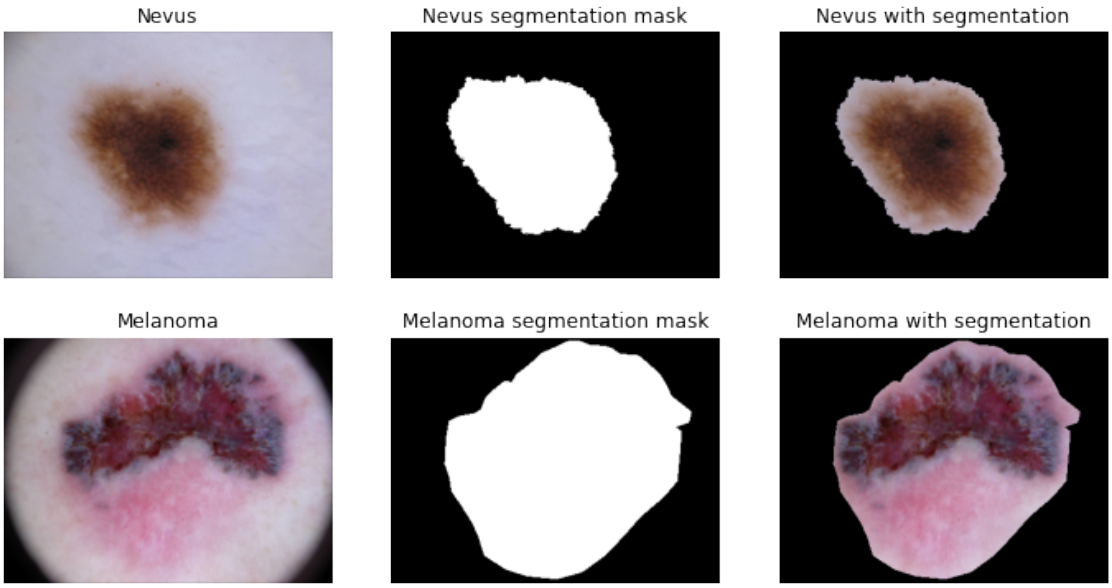
```
#   dest_path='./data/melanoma.jpg')
#   gdd.
 ↪download_file_from_google_drive(file_id='1B2Ol92mBcHN6ah3bpoucBbBbHkPMGC8D',
#   dest_path='./data/melanoma-seg.jpg')
# else:
#   print('You are not using Colab. Please define working_dir with the absolute␣
 ↪path to the folder where you downloaded the data')

# Please modify working_dir only if you are using your Anaconda (and not Google␣
 ↪Colab)
Working_directory="./"
```

Let's load the images, rescale them so that the computations are faster and plot them

[3]:
```
# Nevus
nevus = imread(Working_directory + 'nevus.jpg')
nevus=nevus[2:-2,2:-2,:] # remove border (it contains artifacts)
nevusMask = imread(Working_directory + 'nevus-seg.jpg')
nevusMask=nevusMask[2:-2,2:-2] # remove border (it contains artifacts)
# We rescale to speed up computations
nevus = rescale(nevus, 0.25,multichannel=True, anti_aliasing=True)
# We need all these options to preserve the binary values
nevusMask = rescale(nevusMask, 0.25, anti_aliasing=False, order=0, ␣
 ↪preserve_range=True)
nevusMask_boolean = (nevusMask/255).astype(np.uint8) # To get uint8
nevusMask_expand = np.expand_dims(nevusMask_boolean, axis=2) # To have a 3␣
 ↪channels boolean mask

# Melanoma
melanoma = imread(Working_directory + 'melanoma.jpeg')
melanoma=melanoma[2:-2,2:-2,:] # remove border (it contains artifacts)
melanomaMask = imread(Working_directory + 'melanoma-seg.png')
melanomaMask=melanomaMask[2:-2,2:-2] # remove border (it contains artifacts)
melanoma = rescale(melanoma, 0.25,multichannel=True, anti_aliasing=True)
melanomaMask = rescale(melanomaMask, 0.25, anti_aliasing=False, order=0, ␣
 ↪preserve_range=True)
melanomaMask_boolean = (melanomaMask/255).astype(np.uint8)
melanomaMask_expand = np.expand_dims(melanomaMask_boolean, axis=2)

fig = plt.figure(figsize=(12, 12))
grid = AxesGrid(fig, 111,
                nrows_ncols = (2, 3),
                axes_pad = 0.5)
grid[0].imshow(nevus)
grid[0].set_title('Nevus')
grid[0].axis('off')
grid[1].imshow(nevusMask_boolean,cmap='gray')
```

```
grid[1].set_title('Nevus segmentation mask')
grid[1].axis('off')
grid[2].imshow(nevusMask_expand*nevus)
grid[2].set_title('Nevus with segmentation')
grid[2].axis('off')
grid[3].imshow(melanoma)
grid[3].set_title('Melanoma')
grid[3].axis('off')
grid[4].imshow(melanomaMask_boolean,cmap='gray')
grid[4].set_title('Melanoma segmentation mask')
grid[4].axis('off')
grid[5].imshow(melanomaMask_expand*melanoma)
grid[5].set_title('Melanoma with segmentation')
grid[5].axis('off')
```

[3]: (-0.5, 374.5, 280.5, -0.5)



Questions

1. (IMP+IMH) Before running K-means, please answer this question. How many classes K should you look for ? Would you use the same K for both images ? Why ?
2. (IMP+IMH) Run the following code for both images. Try to choose different channels (among the channels Red, Green and Blue) and different number of clusters K. Which is the best choice in terms of channel and number of classes ? You can use the Dice score to quantitatively compare your mask and the manual segmentation. Comment the results with respect to the previous answer.
3. (IMP+IMH) Propose a way to automatically select the class (or unions of classes) representing

the skin lesion and not the skin. (You don't need to implement it, just explain how you would do it).

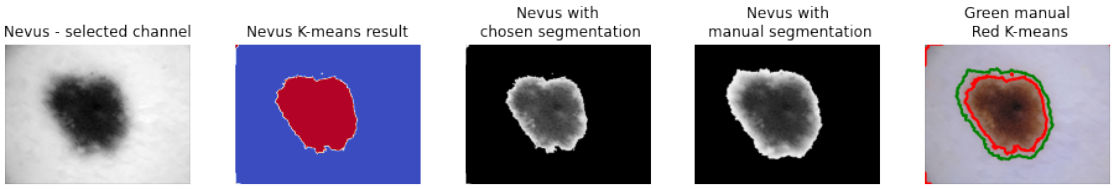
4. (IMP) In K-means we recompute the average at each iteration. The average is not constrained to be one of the original observations. It is usually an *interpolation* of the original observations. How would you change the Lloyd's algorithm to constrain the average to always be one of the original observations ?
5. (IMP - Optional) Code the Lloyd's algorithm for K-Means and compare it with the version of scikit-learn. **Answers:**
6. I will use 2 for nevus but maybe more for melanoma since we want to discarnate more than two classes since we have two different colors that need to be assuming for the same class.
7. for nevus channel 2 and 2 classes we get 83% dice score which is the best I got. for melanoma the best we got was by using channel 0 and 4 classes we get 54% dice score.
8. I would take the RGB of each pixel and compute the sum and give that to the kmean using 3 classes one class (the biggest one most probably) will be the healthy tissue
9. If we like for the average to be one of the original observations after every iteration Ill take the average of each class $(K_1, K_2, ..., K_i)$ and will do the following process $K_{1new}, K_{2new}, ..., K_{inew} = X_{dataset}[np.argmin(np.abs(X_{dataset} - K_1, K_2, ..., K_i))]$ now each class assigned to the nearest observation of the k classes

The dice score (or dice similarity) between two binary masks is defined as $\frac{2TP}{2TP+FP+FN}$ and it ranges between 0 (completely different) and 1 (perfectly equal).

[4]:
```
## NEVUS
# Select a channel (0 for Red, 1 for Green and 2 for Blue)
channel=2
##
nevusB = nevus[:,:,channel]

# Select the number of cluster K to look for
K= 2 # choose a number of clusters
##
kmeans=KMeans(n_clusters=K, random_state=1)
labels=kmeans.fit_predict(nevusB.reshape(-1,1))
labels=np.reshape(labels,(nevusB.shape[0],nevusB.shape[1]))

# Depending on the number of classes K, K-means returns one integer per pixel
# which indicates the number of the cluster.
# Choose the integer to use as mask between 0 and K-1
index =1
mask=labels==index ## choose which label should be

contourMask = find_contours(mask, 0.5)
contourManual = find_contours(nevusMask_boolean, 0.5)

# plot the results
```

```
fig = plt.figure(figsize=(16, 12))
grid = AxesGrid(fig, 111,
                nrows_ncols = (1, 5),
                axes_pad = 0.5)
grid[0].imshow(nevusB,cmap='gray')
grid[0].set_title('Nevus - selected channel')
grid[0].axis('off')
grid[1].imshow(labels,cmap='coolwarm')
grid[1].set_title('Nevus K-means result')
grid[1].axis('off')
grid[2].imshow(mask*nevusB,cmap='gray')
grid[2].set_title('Nevus with\n chosen segmentation')
grid[2].axis('off')
grid[3].imshow(nevusMask_boolean*nevusB,cmap='gray')
grid[3].set_title('Nevus with\n manual segmentation')
grid[3].axis('off')
grid[4].imshow(nevus)
for contour in contourMask:
  grid[4].plot(contour[:, 1], contour[:, 0], linewidth=2, c='r')
for contour in contourManual:
  grid[4].plot(contour[:, 1], contour[:, 0], linewidth=2, c='g')
grid[4].set_title('Green manual\n Red K-means')
grid[4].axis('off')

# Compute the dice score between your mask and the manual segmentation
print('The dice score is ', 1-dice(nevusMask_boolean.reshape(1,-1), mask.
 ↪reshape(1,-1)))
```

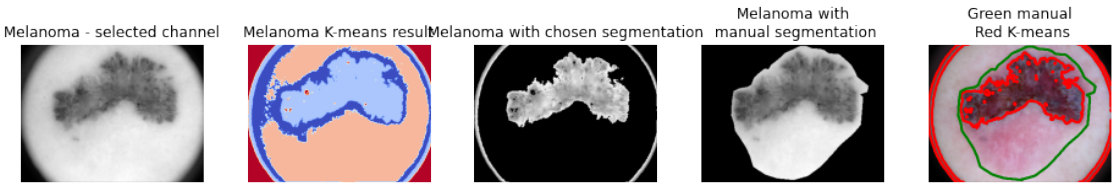
```
The dice score is  0.8296296296296296
```



[5]:
```
## MELANOMA
# Select a channel (0 for Red, 1 for Green and 2 for Blue)
channel=0
##
melanomaB = melanoma[:,:,channel]

# Select the number of cluster K to look for
K= 4# choose a number of clusters
##
```

```
kmeans=KMeans(n_clusters=K, random_state=1)
labels=kmeans.fit_predict(melanomaB.reshape(-1,1))
labels=np.reshape(labels,(melanomaB.shape[0],melanomaB.shape[1]))

# Depending on the number of classes K, K-means returns one integer per pixel
# which indicates the number of the cluster.
# Choose the integer to use as mask between 0 and K-1
index =1
mask=labels==index ## choose which label should be

contourMask = find_contours(mask, 0.5)
contourManual = find_contours(melanomaMask_boolean, 0.5)

fig = plt.figure(figsize=(16, 12))
grid = AxesGrid(fig, 111,
                nrows_ncols = (1, 5),
                axes_pad = 0.5)
grid[0].imshow(melanomaB,cmap='gray')
grid[0].set_title('Melanoma - selected channel')
grid[0].axis('off')
grid[1].imshow(labels,cmap='coolwarm')
grid[1].set_title('Melanoma K-means result')
grid[1].axis('off')
grid[2].imshow(mask*melanomaB,cmap='gray')
grid[2].set_title('Melanoma with chosen segmentation')
grid[2].axis('off')
grid[3].imshow(melanomaMask_boolean*melanomaB,cmap='gray')
grid[3].set_title('Melanoma with\n manual segmentation')
grid[3].axis('off')
grid[4].imshow(melanoma)
for contour in contourMask:
  grid[4].plot(contour[:, 1], contour[:, 0], linewidth=2, c='r')
for contour in contourManual:
  grid[4].plot(contour[:, 1], contour[:, 0], linewidth=2, c='g')
grid[4].set_title('Green manual\n Red K-means')
grid[4].axis('off')

print('The dice score is ', 1-dice(melanomaMask_boolean.reshape(1,-1), mask.
 ↪reshape(1,-1)))
```

```
The dice score is  0.5405599517769427
```



[6]:
```
## (IMP - Optional)
def KMeansLectue(X):
  XXXXXXXX
  return XXXXX
```