ICT for Health
Laboratory # 3
Segmentation of moles

Monica Visintin

Politecnico di Torino

2024/25

Table of Contents

Description [1]

- Download file `images.zip` from the DropBox folder for laboratory 3
- Unzip the file, you'll get several jpeg images of moles, the name of the files are
  1. `low_risk_n.jpg` (where `n` is an integer) for moles that have a low probability of being melanoma (i.e. tumors)
  2. `medium_risk_n.jpg` (where `n` is an integer) for moles that have a low probability of being melanoma
  3. `melanoma_n.jpg` (where `n` is an integer) for moles that have a high probability of being melanoma
- View all the pictures, so that you have an idea of what you have to work with.
- The goal is to help dermatologists in the classification of moles.

Table of Contents

Goal of the lab

- We want to segment the moles, i.e. to find the region of the picture in which the moles are present.
- Having segmented the moles, it is then possible to extract the features (i.e. the numbers) that are relevant for the diagnosis of melanoma.
- Dermatologists use 5 features (ABCDE):
  - **A** asymmetry
  - **B** border
  - **C** color
  - **D** diameter
  - **E** evolution
- For example a feature that can be extracted is the ratio between the length of the border of the mole and the circumference of a perfect circle with the same area: the higher this ratio, the higher the indentation of the mole border, the higher the risk of the mole to be a cancer.

Table of Contents

Main idea

- We use K-means in scikit-learn to find three color clusters, i.e. we quantize of the image so that it has only three colours. It is possible to use more than 3 colors, but 3 is fine.
- We find the contour of the darkest cluster, corresponding to the mole.
- Unfortunately this method sometimes finds non just the mole (in the center of the picture, typically), but other regions where for example you simply have shadows in the picture.
- Then we work on the coordinates of the pixels in the the darkest cluster, and we use DBSCAN on these coordinates, in order to group together pixels that are spatially closer. DBSCAN will generate some clusters (with labels from 0 to $N-1$) and will separate isolated points (outliers, with label -1). The number of found clusters depend on the chosen hyper-parameters for DBSCAN.
- The mole will probably be in the cluster with a relatively large number of points (how many? you must decide) and with a low moment of inertia.
- Then we filter the cluster that only includes the mole to have a smoother contour and we apply another filter to get the mole border.

Table of Contents

The jpeg image [1]

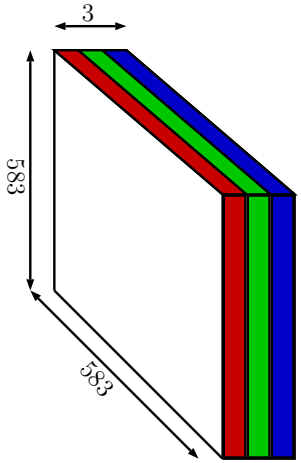
- The jpeg image is an image that has been compressed according to the jpeg standard
- To read the image in Python:

```
1 import matplotlib.image as mpimg
2 filein=...
3 im_or = mpimg.imread(filein)
```

- `im_or` is an Ndarray with shape **N1 x N2 x 3** and elements of type unint8 (unsigned integer with 8 bits); the original image is made of N1 x N2 pixels (for example 583x583)
- `im_or[:,:,0]` stores the amount of **red** color, from 0 to 255
- `im_or[:,:,1]` stores the amount of **green** color, from 0 to 255
- `im_or[:,:,2]` stores the amount of **blue** color, from 0 to 255
- Value `[0,0,0]` corresponds to black, value `[255,255,255]` corresponds to white

The jpeg image [2]

The image dimension is 3



The jpeg image [3]

- To show the image in Python:

```
plt.figure()
plt.imshow(im,interpolation=None)
plt.title('original image')
plt.show()
```

K-means [1]

- Import K-means by writing:

```
1 from sklearn.cluster import KMeans
```

- To instantiate the K-means object write:

```
1 kmeans = KMeans(n_clusters=3, random_state=0)
```

- To find the clusters you should write:

```
1 kmeans.fit(im)
```

but Python gives you an error, because it requires a 2D Ndarray, not a 3D Ndarray.

K-means [2]

- What can we do? the K-means algorithm does not take into consideration the order of the data, therefore we can reshape the 3D Ndarray into a 2D Ndarray:

```
[N1,N2,N3]=im.shape
im_2D=im_or.reshape((N1*N2,N3)) # N1*N2 rows and N3 columns
```

- Then, use k-means as:

```
kmeans.fit(im_2D)
```

- Note that class KMeans takes time to find the clusters: it actually tests some hundreds of different initial vectors and gives you the best clustering that minimizes the moment of inertia.
- The attributes of class `KMeans` are:

K-means [3]

```
kmeans.cluster_centers_ # the centroids of the clusters
kmeans.labels_ # the N1*N2 classes/clusters each pixel belongs to
Ncluster = len(kmeans.cluster_centers_)
```

Note that the centroids are float numbers, but we need `uint8` numbers to show the image; therefore the centroids become:

```
centroids=kmeans.cluster_centers_.astype('uint8')
```

These three centroids represent the three colors that K-means found as representatives of all the image colors (the original image has potentially $2^{24} \simeq 16 \times 10^6$ different colors, but we want only 3 different colors, quantized)

- To see the image with quantized colors, you can use the following lines:

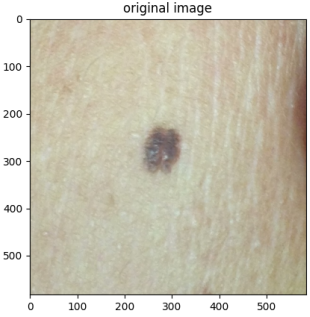
K-means [4]

```
1 im_2D_quant=im_2D.copy()
2 for kc in range(Ncluster):
3     im_2D_quant[(kmeans.labels_==kc),:]=centroids[kc,:]
4 im_quant=im_2D_quant.reshape((N1,N2,N3))
```

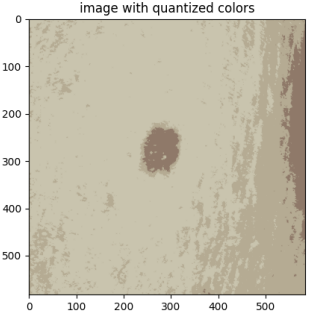
K-means [5]

`low_risk_4.jpg`

original colors



quantized colors



The contour [1]

- The classical algorithm to find the contour is the "**snake**" algorithm (or active contour), available both in Matlab and Python. **You are NOT allowed to use the snake algorithm in this lab** (otherwise you have nothing to do....)
- Among the centroids, find the darkest color, it is the colour of the mole (the other two colours are for the skin and shadows, typically). It might be convenient to convert RGB colors into gray colors:

```
centroids=kmeans_centroids
conv_to_gray=np.array([0.2125,0.7154,0.0721])
centroids_gray = centroids@conv_to_gray
i_col=centroids_gray.argmin()# index of the cluster that corresponds
# to the darkest color
```

The contour [2]

- Generate the image with the 3 indexes and find the positions (i, k) of the pixels belonging to cluster `i_col` (position of the mole):

```
1 im_clust=kmeans.labels_.reshape(N1,N2)
2 mole_pos=np.argwhere(im_clust==i_col)
```

Array `mole_pos` is a 2D array with as many rows as the number of pixels belonging to cluster `i_col` and 2 columns (row and column of each pixel).

- Use DBSCAN with appropriate parameters so that you can identify the mole alone:

```
1 from sklearn.cluster import DBSCAN
2 clusters = DBSCAN(eps=?,min_samples=?).fit(mole_pos)
```

Remember that you will probably get a cluster with label -1, which corresponds to isolated pixels. Of course that cluster will not contain the mole.

The contour [3]

- To select the correct cluster among those found by DBSCAN, consider that:
  1. In any case the mole has a compact shape, even if indented. Can moment of inertia be a reasonable parameter to check?
  2. The mole cannot have less than 1000 pixels (approximately 30 pixels by 30 pixels). This allows you to exclude very small moles that sometimes appear together with a larger mole in the same picture. Of course we are interested in the larger mole.

The contour [4]

- Once you find the correct index (let us say that the index is `i_mole`), you can see the mole isolated from the background using the following lines

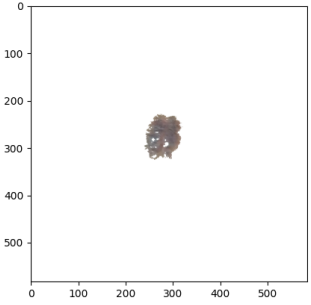
```
true_mole_pos = mole_pos[clusters.labels_==i_mole]
im_only_mole = 0*im_or-1# white image
x=true_mole_pos[:,0]
y=true_mole_pos[:,1]
im_only_mole[x,y,...]=im_or[x,y,...]
plt.figure()
plt.imshow(im_only_mole,interpolation=None)
im_mole_pos = np.zeros((N1,N2))
im_mole_pos[x,y]=1
plt.figure()
plt.imshow(im_mole_pos,interpolation=None)
```

Note that `im_only_mole` has the original colors, not the quantized ones, whereas `im_mole_pos` shows the positions of the mole pixels (white and black; `im_mole_pos` has only 2 dimensions, like a matrix). In the next slide a smaller image was obtained using only the area where the mole was found (`im_mole_pos_red`).

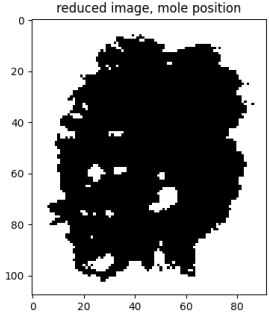
The contour [5]

`low_risk_4.jpg`

Mole



Smaller image, position of the mole

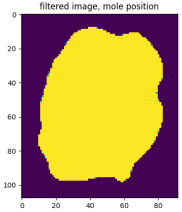


The contour [6]

- The borders of the found area are not clear (too indented). It is better to smooth these borders, using a filter. Read the file with the prerequisites on filters in DropBox.
- Generate the new image `im_mole_pos_red_filt` by setting the value of the pixel in position $[i, k]$ with the **median of the values** in the area $[i - \Delta : i + \Delta, k - \Delta : k + \Delta]$ in the image `im_mole_pos_red`. Choose a correct value for Δ.

```
1 im_mole_pos_red_filt = 0*im_mole_pos_red
2 for kr in range(delta,N1-delta):
3     for kc in range(delta,N2-delta):
4         sub = im_mole_pos_red[kr-delta:kr+delta,kc-delta:kc+delta]
5         im_mole_pos_red_filt[kr,kc] = np.median(sub)
```

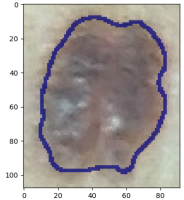
The contour [7]



The contour [8]

- Once you get a clean filtered image, you can use Sobel filters (both of them) to find the border of the mole. In practice each 3×3 pixel original sub-image with center pixel in position $[k_r, k_c]$ is multiplied elementwise by the 3×3 matrix $\mathbf{H}_x$ (or $\mathbf{H}_y$), and the sum of all the products is the value in position $[k_r, k_c]$ of the filtered image. The code is similar to that in slide 22.

$$\mathbf{H}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad \mathbf{H}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \mathbf{H}_x^T$$



The contour [9]

- Note that any measurement instrument has a precision and an accuracy. Perfect instruments do not exist, you must always find a compromise. In the picture of the previous slide you might start discussing whether the mole pixels are all included in the border or not, but the discussion might go on forever, for each pixel...

From segmentation to features

What could you do to extract relevant features from the isolated mole, according to the ABCDE rule? You are not asked to implement the code to extract the features, but you should think about it, so that during the exam you can answer the questions.