


浙江大学爱丁堡大学联合学院

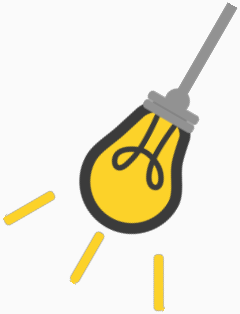
ZJU-UoE Institute

Lecture 07 - Segmentation

Nicola Romanò - nicola.romano@ed.ac.uk

Learning objectives

- Describe the problem of segmentation
- Define and give examples of semantic and instance segmentation.
- Describe and use methods for semantic segmentation (thresholding, clustering, etc.)
- Describe and use methods for instance segmentation (watershed)



Introduction

Segmentation - definition

- Segmentation is a long-studied (and complex!) problem in computer vision.
- Process of dividing an image into sets of pixels called *segments* or *objects*
- Each pixel gets a label identifying which object it belongs to.
- The different segments can then be analysed independently.

Application of segmentation

In biomedical imaging

- Segmentation of cells to measure their properties
- Analysis of X-ray images to identify pathologies
- Aid in surgery planning

Application of segmentation

In biomedical imaging

- Segmentation of cells to measure their properties
- Analysis of X-ray images to identify pathologies
- Aid in surgery planning

In computer vision in general

- Car, pedestrian, break lights detection (e.g. for autonomous car navigation)
- Face detection (e.g. for facial recognition, emotion analysis etc.)
- Fingerprint recognition
- …

Methods for segmentation

Segmentation is a difficult problem to solve, with many important practical applications, so it has been widely studied.

Methods for segmentation

Segmentation is a difficult problem to solve, with many important practical applications, so it has been widely studied.

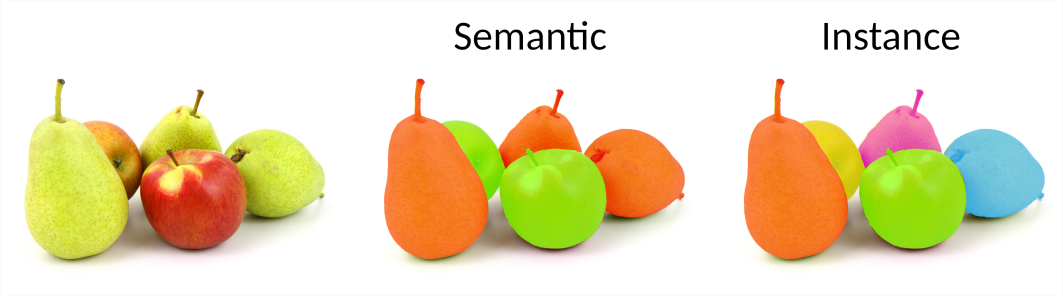
This lecture will cover some of the **traditional methods** for image segmentation. More recent **machine learning-based methods** will be covered later in the course.

Semantic segmentation vs instance segmentation

There are two main types of segmentation:

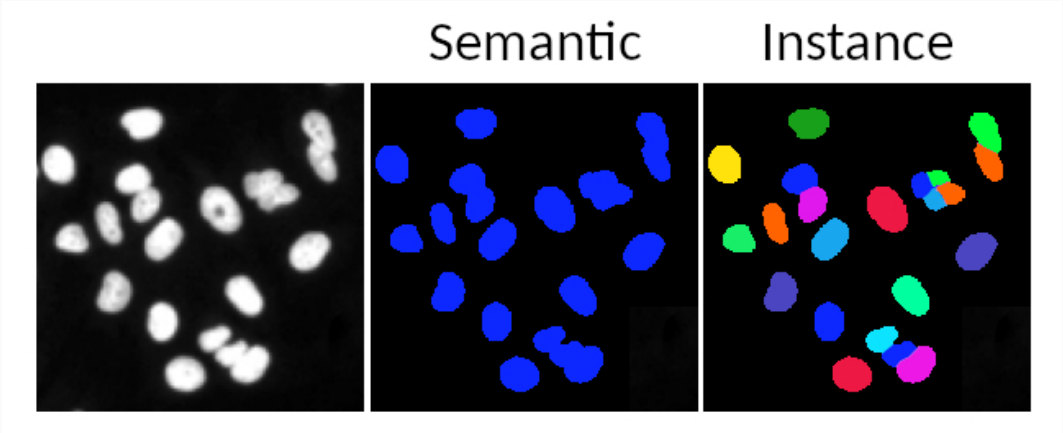
- **Semantic segmentation**: divides the image in regions, each of which belongs to a specific class. Multiple objects of the same class will be detected in the same region.
- **Instance segmentation**: divides the image in regions, each of which is an instance of a class. Multiple objects of the same class will be detected as separate regions.

Semantic segmentation vs instance segmentation - an example



Source: Apples and pears - Petr Kratochvil - CC0

Semantic segmentation vs instance segmentation - an example



Source: Nicola Romanò

Semantic segmentation - thresholding methods

Thresholding

Thresholding is the simplest way of performing semantic segmentation, when there is a clear distinction between the object(s) of interest and the background.

We define a threshold t and create a mask M such that:

$$M(x, y) = \begin{cases} 0 & \text{for } I(x, y) < t \\ 1 & \text{for } I(x, y) \geq t \end{cases}$$

Thresholding

Thresholding is the simplest way of performing semantic segmentation, when there is a clear distinction between the object(s) of interest and the background.

We define a threshold t and create a mask M such that:

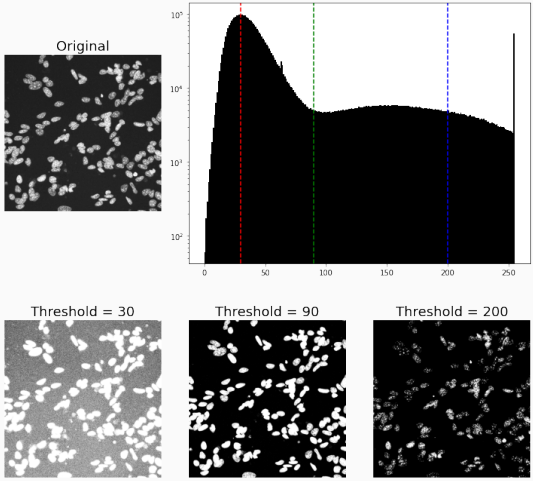
$$M(x,y) = \begin{cases} 0 & \text{for } I(x,y) < t \\ 1 & \text{for } I(x,y) \geq t \end{cases}$$

This can be extended to multi-class segmentation by chosing $t_1, t_2, \ldots, t_n$ and generating a mask

$$M(x,y) = \begin{cases} 0 & \text{for } I(x,y) < t_1 \\ 1 & \text{for } t_1 \leq I(x,y) < t_2 \\ \ldots & \\ n & \text{for } I(x,y) > t_n \end{cases}$$

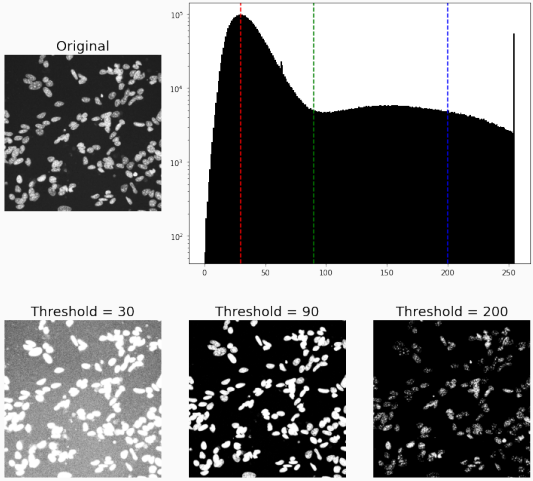
Choosing a threshold

We can manually choose a threshold by inspecting the image histogram



Choosing a threshold

We can manually choose a threshold by inspecting the image histogram



Is there a better way?

Otsu's method

Otsu's method is a simple way to automatically choose a threshold.

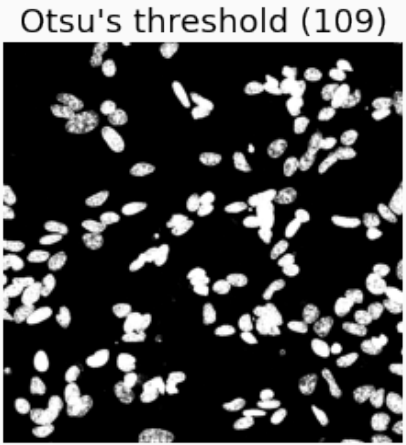
The algorithm makes an exhaustive search for the optimal threshold that maximizes the between-class variance (equivalent to minimizing the intra-class variance).

$$\sigma_\omega^2 = \omega_0 \sigma_0^2 + \omega_1 \sigma_1^2$$

$$\omega_0 = \sum_{i=0}^{t-1} p(i) \qquad \omega_1 = \sum_{i=t}^{L-1} p(i)$$

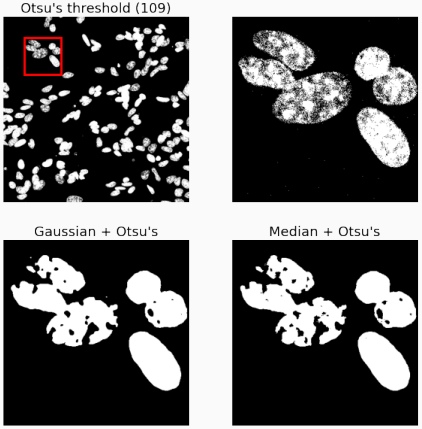
Otsu's method - an example

```
from skimage.filters import threshold_otsu

t = threshold_otsu(img)

plt.imshow(img > t, cmap="gray")
plt.axis("off")
plt.title(f"Otsu's threshold ({t})",
fontsize=18)
plt.show()
```



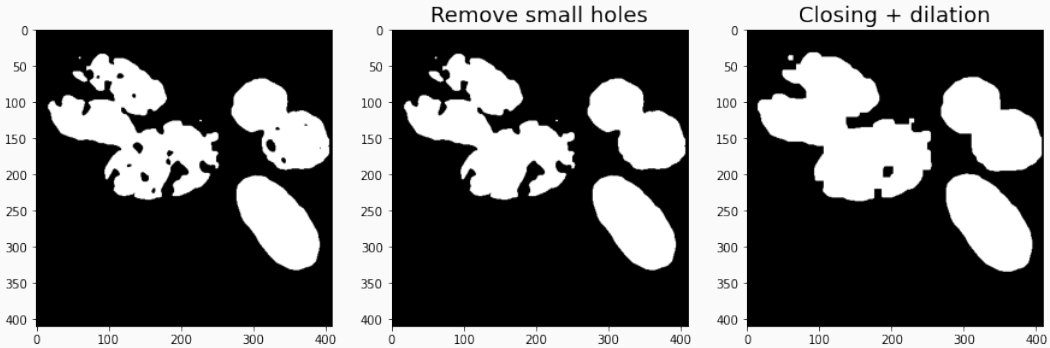
Issues with thresholding

- Noise is a problem -> Gaussian (or median) filter helps
- Need to remove holes afterwards



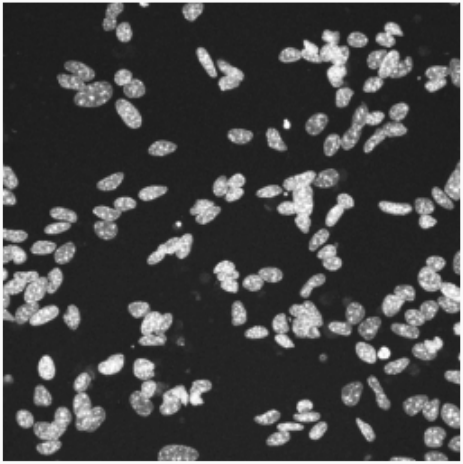
Filling holes

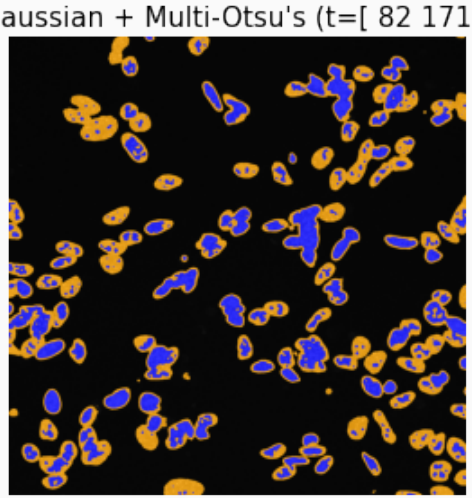
Morphological operations allow to fill small holes. The ‘skimage.morphology.remove_small_holes‘ function is an example of such an operation.



Multi-Otsu segmentation

The Otsu's method can be extended to multi-class segmentation.





Multi-Otsu segmentation - code

```
from skimage.filters import threshold_multiotsu, gaussian
from skimage import img_as_ubyte
import numpy as np
from skimage.color import label2rgb

t = threshold_multiotsu(img, classes=3)
```

Multi-Otsu segmentation - code

```
from skimage.filters import threshold_multiotsu, gaussian
from skimage import img_as_ubyte
import numpy as np
from skimage.color import label2rgb

t = threshold_multiotsu(img, classes=3)

# Remember to go back to unsigned 8-bit integers from float!
img_gaus = img_as_ubyte(gaussian(img, 5))
# Convert to mask with 3 levels
img_thr_gaus = np.digitize(img_gaus, t)
```

Multi-Otsu segmentation - code

```
from skimage.filters import threshold_multiotsu, gaussian
from skimage import img_as_ubyte
import numpy as np
from skimage.color import label2rgb

t = threshold_multiotsu(img, classes=3)

# Remember to go back to unsigned 8-bit integers from float!
img_gaus = img_as_ubyte(gaussian(img, 5))
# Convert to mask with 3 levels
img_thr_gaus = np.digitize(img_gaus, t)

# Handy function to map the labels to colors
img_with_overlay = label2rgb(img_thr_gaus, image = img, bg_label=0, colors =
["orange", "blue"], alpha = .8)

# Then visualise using Matplotlib!
```

Clustering methods

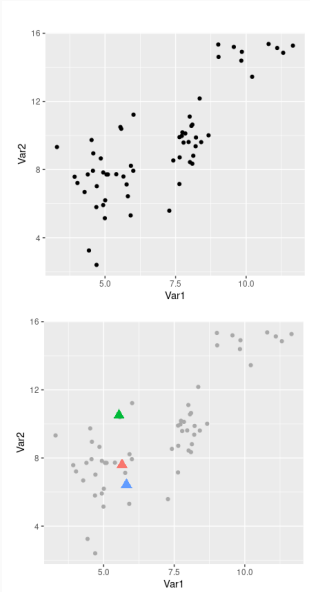
k-means segmentation

k-means clustering is a popular method for segmentation.

One of the simplest approaches to clustering k-means iteratively divides the dataset in k clusters

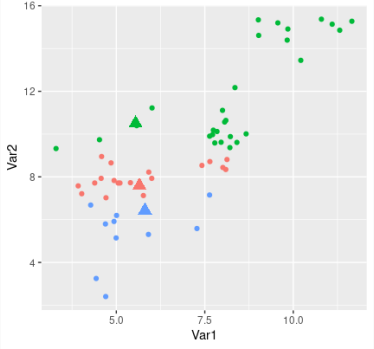
How k-means works

1. We select k random points as the starting centers of the clusters (**centroids**)



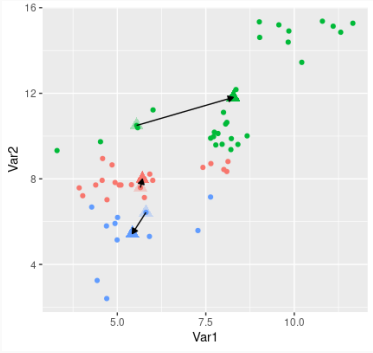
How k-means works

1. We select k random points as the starting centers of the clusters (**centroids**)
2. We assign each point in the dataset to the closest centroid, according to a distance metric of our choice



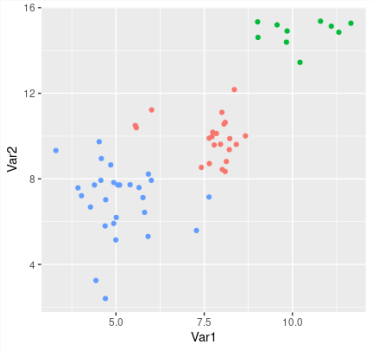
How k-means works

1. We select k random points as the starting centers of the clusters (**centroids**)
2. We assign each point in the dataset to the closest centroid, according to a distance metric of our choice
3. We move the centroids to the center of each cluster



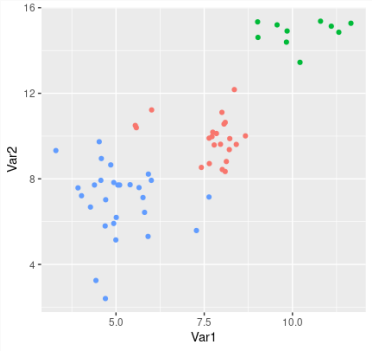
How k-means works

1. We select k random points as the starting centers of the clusters (**centroids**)
2. We assign each point in the dataset to the closest centroid, according to a distance metric of our choice
3. We move the centroids to the center of each cluster
4. We reassign clusters and continue repeating until clusters don't change anymore or until we reach a certain number of iterations

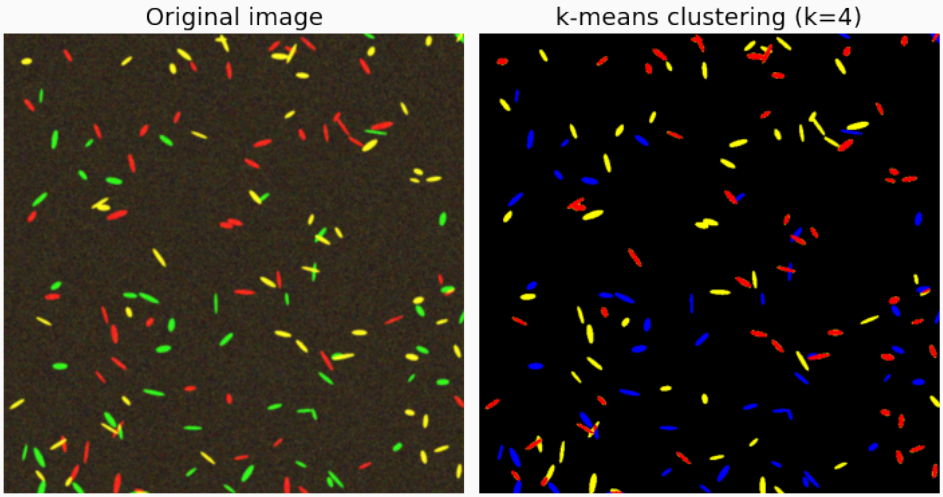


How k-means works

1. We select k random points as the starting centers of the clusters (**centroids**)
2. We assign each point in the dataset to the closest centroid, according to a distance metric of our choice
3. We move the centroids to the center of each cluster
4. We reassign clusters and continue repeating until clusters don't change anymore or until we reach a certain number of iterations
5. Most often k-means converges after 10-20 iterations

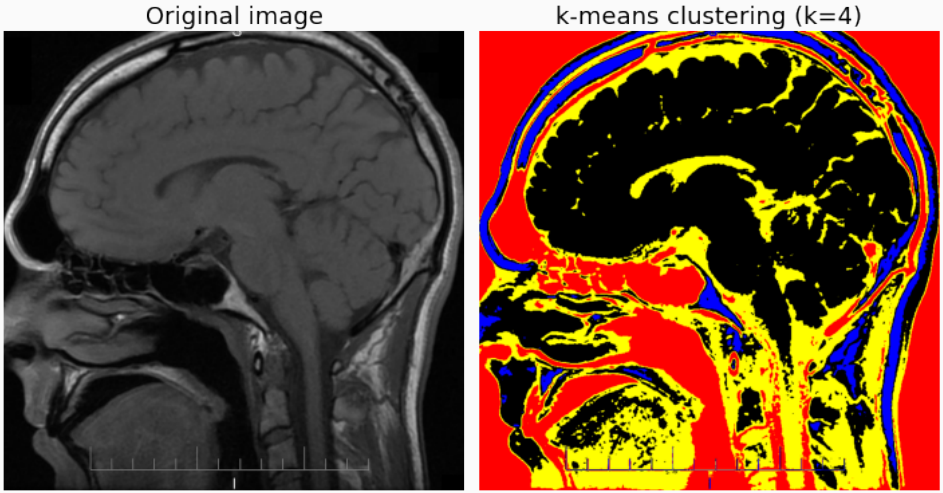


k-means segmentation - an example



Synthetic data simulating cells expressing either or both of a red and green fluorescent protein.

k-means segmentation - an example



MRI: Bryan Kiechle, CC-BY-NC-2.0

k-means segmentation - code

```
from sklearn.cluster import KMeans
from skimage.color import label2rgb

blobs = imread("blobs_green_red.tif")
```

k-means segmentation - code

```
from sklearn.cluster import KMeans
from skimage.color import label2rgb

blobs = imread("blobs_green_red.tif")

# Create k-means model with 4 clusters - set random state for reproducibility
kmeans = KMeans(n_clusters=4, random_state=42)
# Fit the model with our data.  We reshape it in 3 columns (R;G;B)
# For a grayscale image we would reshape to (-1, 1)
kmeans.fit(blobs.reshape(-1, 3))
# Predict the cluster for each pixel and reshape back to original size.
clusters = kmeans.predict(blobs.reshape(-1, 3))
clusters = clusters.reshape(blobs.shape[0], blobs.shape[1])
```

k-means segmentation - code

```
from sklearn.cluster import KMeans
from skimage.color import label2rgb

blobs = imread("blobs_green_red.tif")

# Create k-means model with 4 clusters - set random state for reproducibility
kmeans = KMeans(n_clusters=4, random_state=42)
# Fit the model with our data.  We reshape it in 3 columns (R;G;B)
# For a grayscale image we would reshape to (-1, 1)
kmeans.fit(blobs.reshape(-1, 3))
# Predict the cluster for each pixel and reshape back to original size.
clusters = kmeans.predict(blobs.reshape(-1, 3))
clusters = clusters.reshape(blobs.shape[0], blobs.shape[1])

plt.imshow(label2rgb(clusters, bg_label=0))
plt.plot()
```

Fuzzy-C-Means segmentation

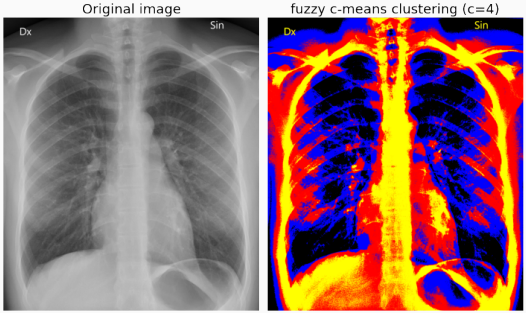
Fuzzy clustering methods differs from hard clustering (such as k-means) in that data points can belong to multiple clusters, with different probabilities.

Fuzzy C-Means (FCM) is the most commonly used fuzzy method. It is similar to k-means, but more robust to noise, inconsistent illuminations etc.

FCM is implemented in the `FCM` function of the `fuzzy-c-means` package (imported as `fcmeans`).

FCM example

```
fcm = FCM(n_clusters=n_clusters,
max_iter=100)
fcm.fit(img.reshape(-1, 1))
clusters = fcm.predict(img.reshape(-1,
1))
clusters = clusters.reshape(img.shape)
```



Instance segmentation

Motivation

Often we want to measure properties of individual objects in an image.

In order to do that we need to segment individual instances of the objects.

We can then create masks for each single object and use it to extract its features (e.g. size, shape, color, intensity etc.).

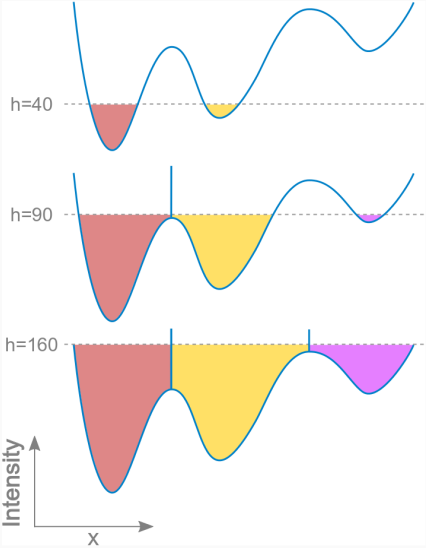
Instance segmentation

Watershed

Watershed

The watershed algorithm was developed in 1979 by Beucher and Lantuéjoul.

If you consider the image as a surface, the watershed algorithm finds the local minima and starts "flooding the image with water" from these points, until the whole image is covered. It then places "barriers" when water from two minima touches each other.



Source: ImageJ.net

Watershed - an example

We are going to segment cell nuclei in an image using the watershed algorithm.

We will perform the following steps:

1. Load the image.

Watershed - an example

We are going to segment cell nuclei in an image using the watershed algorithm.

We will perform the following steps:

1. Load the image.
2. Apply a median filter to remove noise.

Watershed - an example

We are going to segment cell nuclei in an image using the watershed algorithm.

We will perform the following steps:

1. Load the image.
2. Apply a median filter to remove noise.
3. Find Otsu's threshold and perform a binary thresholding to isolate nuclei from background.

Watershed - an example

We are going to segment cell nuclei in an image using the watershed algorithm.

We will perform the following steps:

1. Load the image.
2. Apply a median filter to remove noise.
3. Find Otsu's threshold and perform a binary thresholding to isolate nuclei from background.
4. Apply the Euclidean distance transform to find the pixel-wise distance to the nearest background pixel.

Watershed - an example

We are going to segment cell nuclei in an image using the watershed algorithm.

We will perform the following steps:

1. Load the image.
2. Apply a median filter to remove noise.
3. Find Otsu's threshold and perform a binary thresholding to isolate nuclei from background.
4. Apply the Euclidean distance transform to find the pixel-wise distance to the nearest background pixel.
5. Find local maxima in the distance transform image and mark them as separate regions.

Watershed - an example

We are going to segment cell nuclei in an image using the watershed algorithm.

We will perform the following steps:

1. Load the image.
2. Apply a median filter to remove noise.
3. Find Otsu's threshold and perform a binary thresholding to isolate nuclei from background.
4. Apply the Euclidean distance transform to find the pixel-wise distance to the nearest background pixel.
5. Find local maxima in the distance transform image and mark them as separate regions.
6. Use the local maxima as starting point for the watershed algorithm.

Watershed - an example

We are going to segment cell nuclei in an image using the watershed algorithm.

We will perform the following steps:

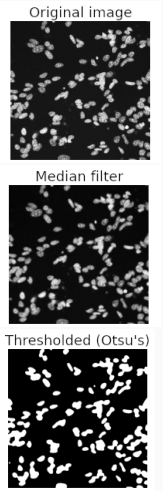
1. Load the image.
2. Apply a median filter to remove noise.
3. Find Otsu's threshold and perform a binary thresholding to isolate nuclei from background.
4. Apply the Euclidean distance transform to find the pixel-wise distance to the nearest background pixel.
5. Find local maxima in the distance transform image and mark them as separate regions.
6. Use the local maxima as starting point for the watershed algorithm.
7. Perform the watershed algorithm on the distance.

Watershed - code

```
# Import the necessary packages
from skimage.segmentation import watershed
from skimage.measure import label
from skimage.filters import median, threshold_otsu
from skimage.morphology import disk
from skimage.transform import rescale
from skimage.feature import peak_local_max
from skimage.color import label2rgb
from scipy.ndimage import distance_transform_edt
```

Watershed - code

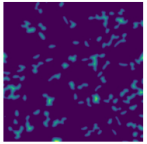
```
# Import the necessary packages
from skimage.segmentation import watershed
from skimage.measure import label
from skimage.filters import median, threshold_otsu
from skimage.morphology import disk
from skimage.transform import rescale
from skimage.feature import peak_local_max
from skimage.color import label2rgb
from scipy.ndimage import distance_transform_edt

# Load the image, smooth and threshold
img = imread("nuclei_DAPI.tif")
img_smooth = median(img, selem=disk(10))
img_otsu = img_smooth > threshold_otsu(img_smooth)
```



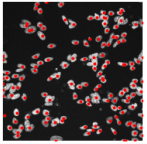
Watershed - code

```
 # Apply the distance transform and find the maxima
img_edt = distance_transform_edt(img_otsu)
peak_idx = peak_local_max(img_edt, min_distance=10,
exclude_border=False)
# Peaks are returned as coordinates in the original
image space,
# so we need to transform them back to the image shape
peak_mask = np.zeros_like(img, dtype=bool)
peak_mask[tuple(peak_idx.T)] = True
# Label the peaks and apply the watershed algorithm
markers = label(peak_mask)
img_watershed = watershed(-img_edt, markers=markers,
mask = img_otsu)
```

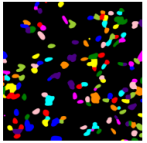
clidian distance transfo



Local maxima



(Masked) watershed



Measuring region properties

Once we have segmented instances, we can measure the properties of each object.

Measuring region properties

Once we have segmented instances, we can measure the properties of each object.

Scikit image provides the `regionprops` function in the `measure` package, that can be used to measure the properties of each region.

Measuring region properties

Once we have segmented instances, we can measure the properties of each object.

Scikit image provides the `regionprops` function in the `measure` package, that can be used to measure the properties of each region.

The `regionprops_table` function returns the properties in a Pandas-compatible table format. (don't know what Pandas is? Check out Workshop 3!)

Measuring properties

```
from skimage.measure import regionprops, regionprops_table
import pandas as pd

props = regionprops(img_watershed, intensity_image=img)

areas = [prop.area for prop in props]
intens = [prop.mean_intensity for prop in props]

plt.scatter(x=areas, y=intens, color="red")
plt.xlabel("Area")
plt.ylabel("Mean intensity")
plt.plot()
```

Measuring properties

```
from skimage.measure import regionprops, regionprops_table
import pandas as pd

props = regionprops(img_watershed, intensity_image=img)

areas = [prop.area for prop in props]
intens = [prop.mean_intensity for prop in props]

plt.scatter(x=areas, y=intens, color="red")
plt.xlabel("Area")
plt.ylabel("Mean intensity")
plt.plot()

# Alternatively, with regionprops_table
tbl = regionprops_table(img_watershed, intensity_image=img,
properties=["area", "mean_intensity"])
tbl = pd.DataFrame(tbl)
tbl.to_csv("cells_area_intensity.csv", index=False)
```

What can I measure?

A full list of properties can be found in the `regionprops` documentation.

These include morphological parameters such as area, perimeter, centroid, bounding box/ellipse, mean intensity and many others!