**Credit Hours System**
**HEMN328: Medical Device Standards**

**Cairo University**
**Faculty of Engineering**

# Midterm Project

**Submitted to:** Eng. Peter Farag

| Names | IDs |
|---|---|
| Doaa Sherif | 1170122 |
| Salma Hazem | 1170425 |
| Yara Wael | 1170431 |

# Image Segmentation

## Problem Definition

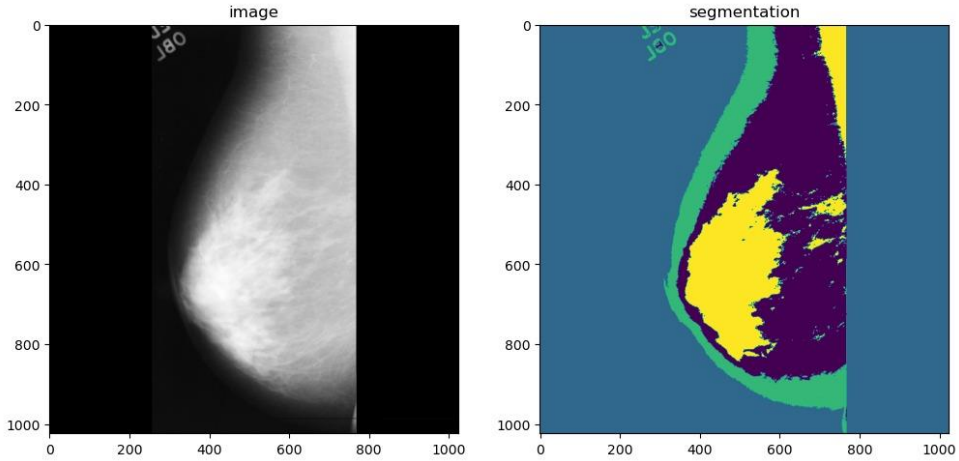
Image Segmentation is the process by which a digital image is partitioned into various subgroups of pixels called Image Objects, which can reduce the complexity of the image, and thus analysing the image becomes simpler. It works by assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

Using these labels, we can specify boundaries, draw lines, and separate the most required objects in an image from the rest of the not-so-important ones.

## Importance in Medical Imaging

Medical image segmentation has an essential role in **computer-aided diagnosis systems** in different applications. It divides an image into areas based on a specified description, such as segmenting body organs/tissues in the medical applications for border detection, tumour detection/segmentation, and mass detection. For example, a tumor, cancer or a block in the blood flow can be easily isolated from its background with the help of image segmentation technique.

It is considered more accurate and time efficient.



# Methods and Algorithms

## 1. C-means Clustering

Clustering is an unsupervised machine learning technique which divides the given data into different clusters based on their distances (similarity) from each other.

- Initialize the Weights (Membership values) with random values
- Calculate the CENTROIDS using the following formula:
  Where U is the membership value, m is the fuzziness parameter (generally taken as 2), and xk is the data.

$$= (\sum_{1}^{n}(\gamma_{ik}^{m} * x_k) / \sum_{1}^{n} \gamma_{ik}^{m}$$

- Then Calculate the distance between the centroids and data
- Calculate the new weights by using this formula

$$\gamma = \sum_{1}^{n} (d_{ki}^2 / d_{kj}^2)^{1/m-1}]^{-1}$$

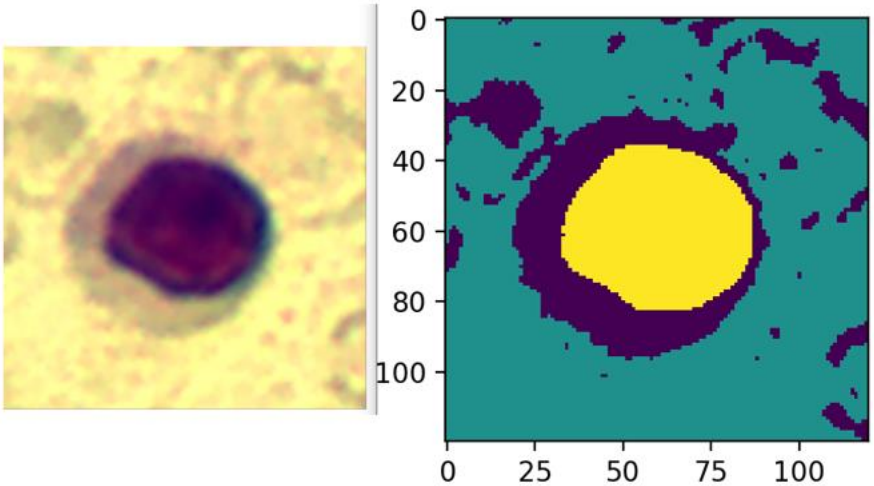
- After that calculate the new Delta
- Check if delta is smaller than the epsilon or the iterator exceeds the maximum iterations BREAK

```
    # Calculating Delta
    delta = math.sqrt(np.sum(pow((W - old_w), 2)))
    # if the iterations exceed the maximum no. of iterations break
    if delta < epsilon or i > max_iterations:
        break
```

- Get the indices of the maximum values

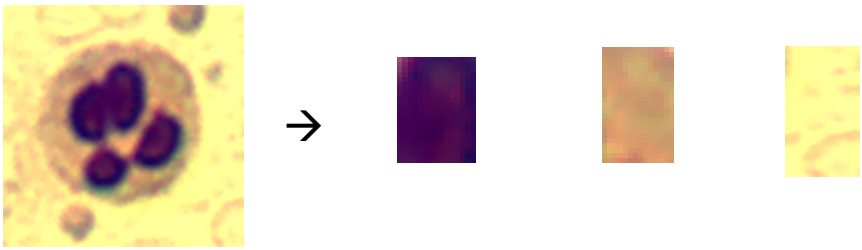
```
result = np.argmax(W, axis = 1)
result = result.reshape(shape).astype('int')
plt.imshow(result)
plt.show()
return result
```

- The Output of the C-means Clustering:



## 2. Support Vector Machine

- Crop each object in image in a separate image to be our training data



- Read each object separately

```
        self.object_1_image = cv2.imread('object_1.bmp')
        self.object_2_image = cv2.imread('object_2.bmp')
        self.object_3_image = cv2.imread('object_3.bmp')
```

- Extract features from each object, which will be mean of all pixels
  First, we iterate over each pixel and the mean in the three channels (RGB), then get the mean of the whole image

```
def get_mean_of_image(self, img):
        one_channel_image = []
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):

                current_pixel_mean = np.mean([img[i][j][2], img[i][j][1], img[i][j][0]])
                one_channel_image.append(current_pixel_mean)

        one_channel_image = np.array(one_channel_image)
        return np.mean(one_channel_image)
```

- Start training process where the feature vector will be the pixels' mean of each object, and labels will be [1, 2, 3], then fit the model using **fit** function in **sklearn.svm** library

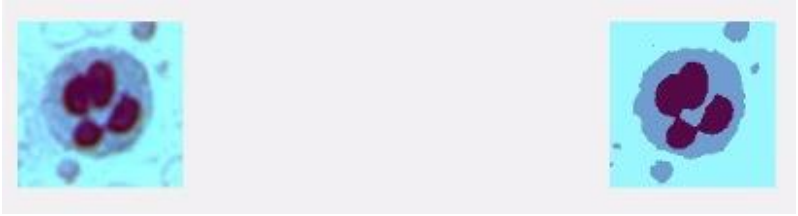
```
X = np.array([[mean_object_1], [mean_object_2], [mean_object_3]])

Y = np.array([1, 2, 3])
clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
clf.fit(X, Y)
```

- Read test image, iterate over each pixel get its mean in the three channels, predict its label using **predict** function in **sklearn.svm** library
- For visualizing the output, we first get the mean of pixels in each channel in the three objects
  To be able to replace each segmented region pixel colour with its mean colour

```
if predicted_label[0] == 1:
                    final_image[i][j][0] = b_channel_mean_1/255
                    final_image[i][j][1] = g_channel_mean_1/255
                    final_image[i][j][2] = r_channel_mean_1/255
                elif predicted_label[0] == 2:
                    final_image[i][j][0] = b_channel_mean_2/255
```

```
                final_image[i][j][1] = g_channel_mean_2/255
                final_image[i][j][2] = r_channel_mean_2/255
            elif predicted_label[0] == 3:
                final_image[i][j][0] = b_channel_mean_3/255
                final_image[i][j][1] = g_channel_mean_3/255
                final_image[i][j][2] = r_channel_mean_3/255
```

## Results



## 3. Shallow Neural Network

Data sets used for Red Blood Cells (RBCs) and White Blood Cells (WBCs)

```
train_WBC = get_array('segmentation_WBC-master/segmentation_WBC-
master/Dataset 1/training/')
train_RBC = get_array('segmentation_WBC-master/RBCs/training/')
```

**To initialize parameters**

```
W1 = np.random.randn(n_h,n_i)*0.01
    b1 = np.zeros((n_h,1))
```

W is the weight matrix for each layer and b is bias

**Sigmoid function**

It is the activation function
Sigmoid takes any real value as input and outputs values in the range -1 to 1, similar to tanh

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

**Forward Propagation**

Returns the activation function, using sigmoid

```
Z1 = np.dot(W1,X)+b1
A1 = sigmoid(Z1)
```

**Compute Loss**

Using gradient descent

```
logprobs = -np.sum(Y*np.log(A2)+(1-Y)*np.log(1-A2))
```

**Update parameters**

new weight = old weight - learning rate * gradient

```
W1 = W1-learning_rate*dW1
```

## Results

Accuracy Train: 93%
Accuracy Test: 93%