INF250: Mandatory Exercise01

- Author: Mahrin Tasfe
- Email: mahrin.tasfe@nmbu.no

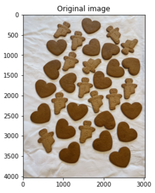
Solution starts here

In [19]:
```
# importing all the necessary modules
import numpy as np
from skimage import io
import matplotlib.pyplot as plt
```

In [20]:
```
def get_image(name):
    """ Using the skimage.io import the image with given file name.
    This function returns the color image as numpy array """

    filename = '../Images/'+name
    image = io.imread(filename)

    return image
```

In [21]:
```
def show_image(image, label ="Image:"):
    """This function shows/plots the given image."""
    plt.subplots(1, 1, figsize = (7,5))
    plt.title(label = label, fontsize = 12)
    plt.imshow(image, cmap='gray')
    plt.show()
```

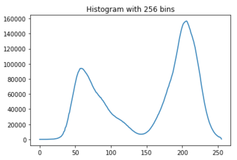
In [22]:
```
#importing the original image and plotting it
image = get_image("gingerbreads.jpg")
show_image(image, "Original image")
```



In [23]:
```
def histogram(image):
    """Returns the histogram with 256 bins."""
    # Setup
    shape = np.shape(image)
    hist = np.zeros(256)

    if len(shape) == 3:#making sure it is a color/RGB image
        image = image.mean(axis=2)
    elif len(shape) > 3:
        raise ValueError('Must be at 2D image')

    # Start to make the histogram
    ## WRITE YOUR CODE HERE

    for i in range(shape[0]):
        for j in range(shape[1]):
            pixval = int(image[i,j])
            hist[pixval] += 1

    return hist
```

In [24]:
```
def show_histogram(histogram, label = "Histogram with 256 bins"):
    """This function plots the given histogram (in the numpy array format)"""
    plt.figure()
    plt.title(label = label, fontsize=12)
    plt.plot(histogram)
    plt.show()
```

In [25]:
```
#Generating histogram for the original image & plotting it
histogram_ans = histogram(image)
show_histogram(histogram_ans)
```



In [26]:
```
def otsu(image):
    """Finds the optimal threshold value of given image using Otsu's method."""

    hist = histogram(image) #Obtaining the histogram of the image
    th = 0

    ## WRITE YOUR CODE HERE
    max_t = 0
    weight_denominator = sum(hist)
    max_between_class_variance = 0
    for t in range(1, 256):# skipping the 1st iteration t = 0,
        #because it will generate a division by zero error while calculating the means u_b & u_f

        w_b_top = sum(hist[0:t+1])
        w_b = w_b_top/weight_denominator

        u_b_top = 0
        u_b_bottom = 0
        for i in range(0, t+1):
            u_b_top += (i*hist[i])
            u_b_bottom+= (hist[i])
        u_b = u_b_top/u_b_bottom


        w_f_top = sum(hist[t:256])
        w_f = w_f_top/weight_denominator


        u_f_top = 0
        u_f_bottom = 0
        for i in range(t, 256):
            u_f_top += (i*hist[i])
            u_f_bottom+= hist[i]
        u_f = u_f_top/u_f_bottom


        current_between_class_variance = w_b*w_f*((u_b - u_f)**2)


        if t == 1:
            # If it is the 1st iteration, putting the 1st value as max
            if not np.isnan(current_between_class_variance):
                max_between_class_variance = current_between_class_variance
                max_t = t
        else:
            #except for the 1st iteration, comparing the current_between_variance with the maximamum.
            #and updating the max_between_variance
            if not np.isnan(current_between_class_variance):
                if(max_between_class_variance < current_between_class_variance):
                    max_between_class_variance = current_between_class_variance
                    max_t = t

    return max_t
```

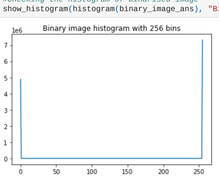
In [27]:
```
#Finding and printing the optinal value from Otsu algoritm
print("The optimal Otsu threshold value:", otsu(get_image("gingerbreads.jpg")))
```

```
The optimal Otsu threshold value: 138
```

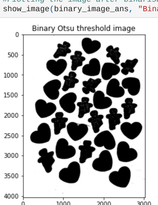
In [28]:
```
def threshold(image, th = None):
    """Returns a binarised version of given image, thresholded at given value.

    Binarises the image using a global threshold `th`. Uses Otsu's method
    to find optimal thrshold value if the threshold variable is None. The
    returned image will be in the form of an 8-bit unsigned integer array
    with 255 as white and 0 as black.

    Parameters:
    -----------
    image : np.ndarray
        Image to binarise. If this image is a colour image then the last
        dimension will be the colour value (as RGB values).
    th : numeric
        Threshold value. Uses Otsu's method if this variable is None.

    Returns:
    --------
    binarised : np.ndarray(dtype=np.uint8)
        Image where all pixel values are either 0 or 255.
    """
    # Setup
    shape = np.shape(image)
    binarised = np.zeros([shape[0], shape[1]], dtype=np.uint8)

    if len(shape) == 3:# checking for 3 channels or RGB/color image
        image = image.mean(axis=2)

    elif len(shape) > 3:
        raise ValueError('Must be at 2D image')

    if th is None:
        th = otsu(image) #Obtaining the optimal threshold value

    #binarising the image
    for i in range(shape[0]):
        for j in range(shape[1]):
            if int(image[i,j] >= th):
                binarised[i, j] = 255

    return binarised
```

In [29]:
```
#Binarising the image with otsu threshold value
binary_image_ans = threshold(get_image("gingerbreads.jpg"))
```

In [30]:
```
#Checking the histogram of binarised image
show_histogram(histogram(binary_image_ans), "Binary image histogram with 256 bins")
```



In [31]:
```
#Plotting the image after binarisation
show_image(binary_image_ans, "Binary Otsu threshold image")
```



In [ ]:

In [ ]:

In [ ]: