MODULE 10 – Thresholding, Simple Segmentation

A. PURPOSE

1. Students are able to understand the Thresholding concept
2. Students understand some Thresholding techniques
3. Students can develop several Thresholding techniques using Python on Google Colab

B. TOOLS AND MATERIAL

1. PC/LAPTOP
2. Github
3. *Google Colaborator*

C. THEORY REVIEW

C.1 Definition of Thresholding Operations

Thresholding is the simplest form of image segmentation method. Usually used in grayscale / color images and the basic idea is how to separate the foreground object from the background object.

The following image shows the original image before and after thresholding.

Figure 1. Left image before threshold, right image after threshold (binary image)

Some of the methods of Threshold that will be discussed in this module include:

a. Global Threshold
b. Adaptive Threshold
c. Otsu's Threshold
d. Introduction to Image Segmentation using K-Means

C.2 Global Threshold

Examine the histogram of an image shown in the following image:

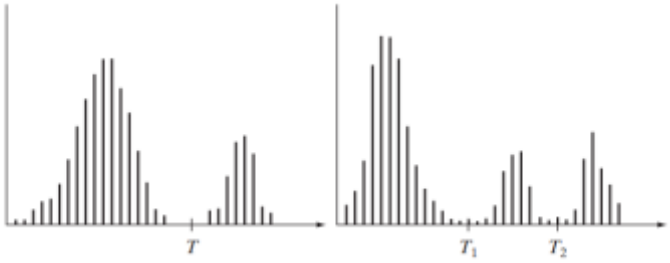


Figure 2.The intensity of the histogram can be separated by 1 Threshold (a) or multiple Threshold (b)

Assume that the image above is a histogram of the f (x, y) image, which consists of bright objects on a dark background. Assuming that the object and background have a color intensity value that can be separated into 2 dominant groups. The easiest and most obvious way is to select a specific value (Threshold, T) that will separate the two groups. Each point (x, y) in the image where f (x, y)> T can be termed an object point , while the other group is called a background point . In other words, the segmented image g (x, y) is written as follows:

$$g(x,y) = \begin{cases} 1 & jika\ f(x,y) > T \\ 0 & jika\ f(x,y) \leq T \end{cases}$$

When T is determined as a constant in the entire image, this process is known as *global thresholding*. When the T value changes in the process of one image, this is called *variable / adaptive thresholding*.

Figure 2 (a) above shows a thresholding problem involving 3 dominant groups. It is assumed that there are 2 light groups and 1 dark group on the histogram. The multiple tholding process will group the points (x, y) as a background if *f(x, y)* <= T_1 , to object group one if T_1 < *f(x, y)* <= T_2, and object group if *f(x, y)* > T_2. Or it can be written as follows:

$$g(x,y) = \begin{cases} a & jika\ f(x,y) > T_2 \\ b & jika\ T_1 < f(x,y) \leq T_2 \\ c & jika\ f(x,y) \leq T_1 \end{cases}$$

OpenCV has provided a library for Thresholding, such are Global, Adaptive, or Otsu Threshold. The following are some of the Global Thresholding provided:

a. Binary Threshold
b. Binary-Inverted Threshold
c. Truncate Threshold
d. Threshold To Zero
e. Threshold To Zero-Inverted

To illustrate how each threshold works, consider the histogram of the original image src (x, y) in Figure 3 below. The blue horizontal line shows the threshold value (fixed).

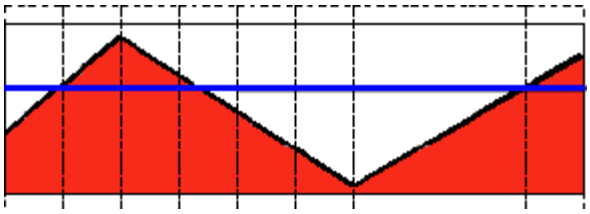


Figure 3. Histogram of image *src(x,y)*

a. Binary Threshold

Binary Threshold, as previously explained, is a threshold that will separate the two color groups. For this threshold, each color group will be changed to a dark (black) value for objects that are considered background, and changed to a light (white) value for objects that are considered foreground. Here is the equation:

$$dst(x,y) = \begin{cases} maxVal & jika\ src(x,y) > thresh \\ 0 & jika\ lainnya \end{cases}$$

So if the color pixel intensity value at src (x, y) is higher than the threshold , the value will be changed to maxVal . Otherwise it will be changed to 0.

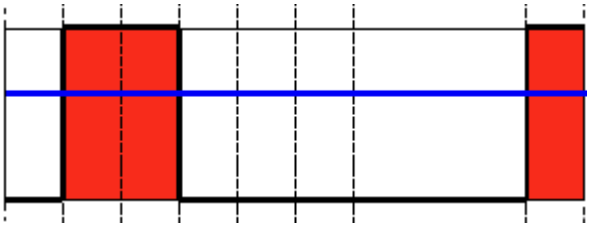


Figure 4. The histogram of the Binary Threshold result

b. Binary-Inverted Threshold

This threshold is the opposite of the Binary Threshold. If the color intensity is above the Threshold, the value will be changed to 0 and vice versa. Here is the equation:

$$dst(x,y) = \begin{cases} 0 & jika\ src(x,y) > thresh \\ maxVal & jika\ lainnya \end{cases}$$

Figure 5 below is a histogram of the Binary-Inverted Threshold results.

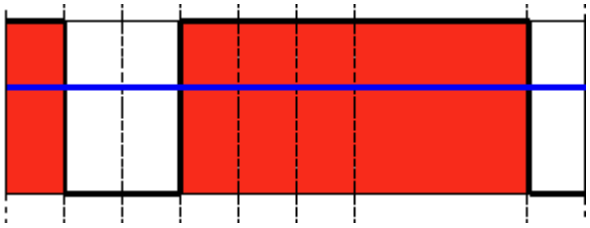


Figure 5. The histogram of the Binary-Inverted Threshold result

c. Truncate Threshold

If the *src (x, y)* color intensity value is greater than the *threshold* , the value will be truncated. Here are the equations and the resulting histogram.

$$dst(x,y) = \begin{cases} thresh & jika\ src(x,y) > thresh \\ src(x,y) & jika\ lainnya \end{cases}$$

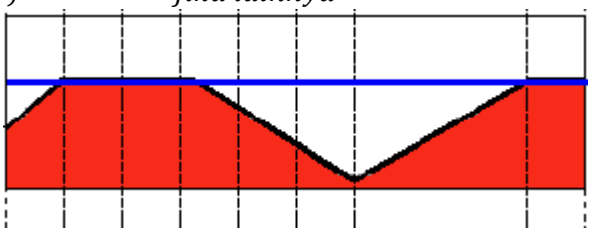


Figure 6. The histogram resulting from the Truncate Threshold

d. Threshold To Zero

If the value of *src (x, y)* is lower than the *threshold* value, the new pixel value will be changed to 0. Below are the equation and histogram image of the threshold result.

$$dst(x,y) = \begin{cases} src(x,y) & jika\ src(x,y) > thresh \\ 0 & jika\ lainnya \end{cases}$$

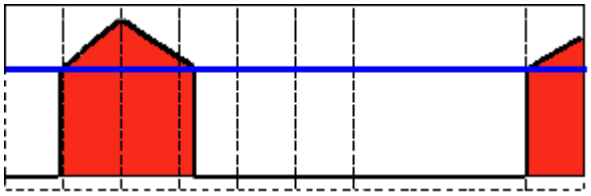


Figure 7 . Histogram result from Threshold To Zero

e. Threshold To Zero – Inverted

If the value of src (x, y) is greater than the threshold value , the new pixel value will be changed to 0. Below are the equation and the histogram image of the threshold result.

$$dst(x,y) = \begin{cases} 0 & jika\ src(x,y) > thresh \\ src(x,y) & jika\ lainnya \end{cases}$$

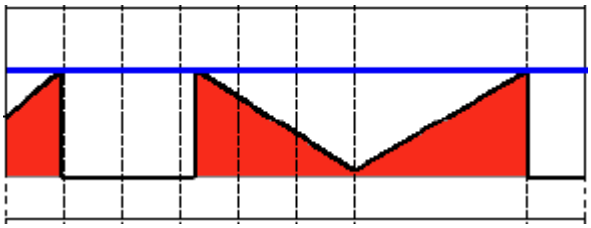


Figure 8. The histogram of the result from Threshold To Zero

The global threshold mentioned above can be used directly using the OpenCV cv.Threshold library. here is the documentation link:

https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

The following code shows the use of the OpenCV cv.Threshold library

```
import cv2 as cv
from google.colab.patches import cv2_imshow
import numpy as np
import matplotlib.pyplot as plt

filename = ('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/gradient
.jpg')
img = cv.imread(filename)
thresh = 127      #nilai Threshold yang ditentukan

#1. thresh1 jika pixel di img>127, maka thresh1 bernilai 1(putih) selain
 itu bernilai 0(hitam)
ret,thresh1 = cv.threshold(img,thresh,255,cv.THRESH_BINARY)
#2. thresh2 adalah binary threshold inverse
ret,thresh2 = cv.threshold(img,thresh,255,cv.THRESH_BINARY_INV)
#3. Threshold Truncate
ret,thresh3 = cv.threshold(img,thresh,255,cv.THRESH_TRUNC)
#4. Threshold Tozero
ret,thresh4 = cv.threshold(img,thresh,255,cv.THRESH_TOZERO)
#5. Threshold Tozero Inverse
ret,thresh5 = cv.threshold(img,thresh,255,cv.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC', 'TOZERO', 'TOZ
ERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

plt.figure(figsize = (15,5))
for i in range(len(images)):
  plt.subplot(2,3,i+1),plt.imshow(images[i],'gray', interpolation='neare
st')
  plt.title(titles[i])
  plt.xticks([]),plt.yticks([])
plt.show()
```

The following figure shows the difference between each global threshold which is the output of the code above . As seen in the code, the global threshold value is 127 and this value applies to the entire image.

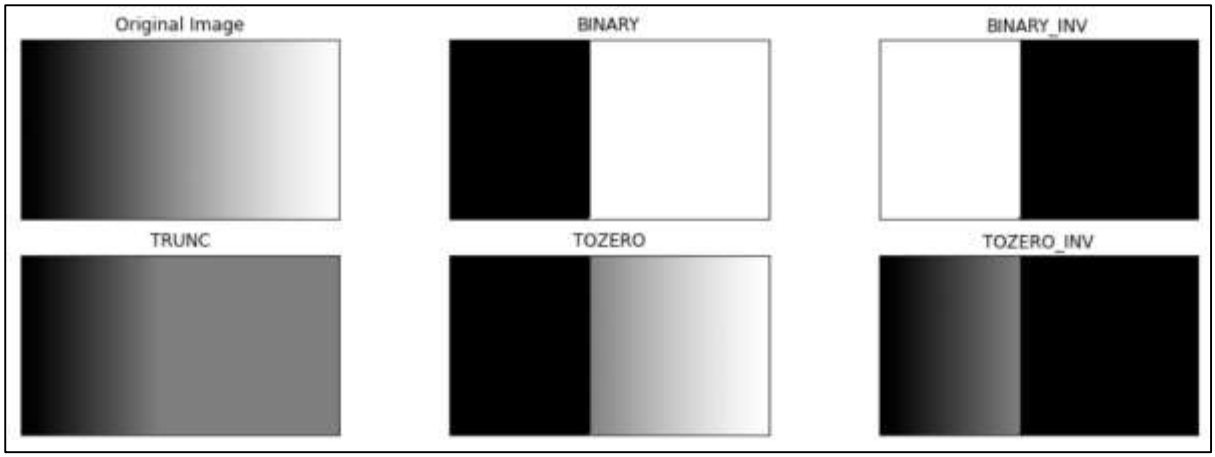


Figure 9. Global Threshold from the OpenCV Library

C.3 Adaptive Threshold

In the previous section, we used global values as threshold values . This is sometimes not good enough in all conditions where the image has different lighting conditions. In this case, adaptive thresholding can be used . An algorithm for calculating the threshold value imposed for a certain area of the entire image . So that different threshold values will be obtained for different areas of the same image . It is expected to give better results for images with different lighting.

There are 2 libraries provided, namely: cv.ADAPTIVE_THRESH_MEAN_C (the threhold value is the average of the defined neighbor area) and cv.ADAPTIVE_THRESH_GAUSSIAN_C (the threshold value is the number of weights of the neighbor value where the weight is the gaussian window). The neighboring area is defined by Block Size, while C is a given constant which will be subtracted from the average value or the number of weights. The following is a sample code for Adaptive Threshold.

```
filename = ('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/sudoku-original.jpg')
citra = cv.medianBlur(cv.imread(filename),5)
gray = cv.cvtColor(citra, cv.COLOR_BGR2GRAY)
#gray = cv.medianBlur(gray,5)

thresh = 127

ret,th1 = cv.threshold(gray,thresh,255,cv.THRESH_BINARY)
th2 = cv.adaptiveThreshold(gray,255,cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY,11,2)
th3 = cv.adaptiveThreshold(gray,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY,11,2)

titles = ['Citra Asli', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
citra2 = [gray, th1, th2, th3]

plt.figure(figsize = (10,10))
for i in range(len(citra2)):
    plt.subplot(2,2,i+1),plt.imshow(citra2[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

The following is an image of the result of the code above.

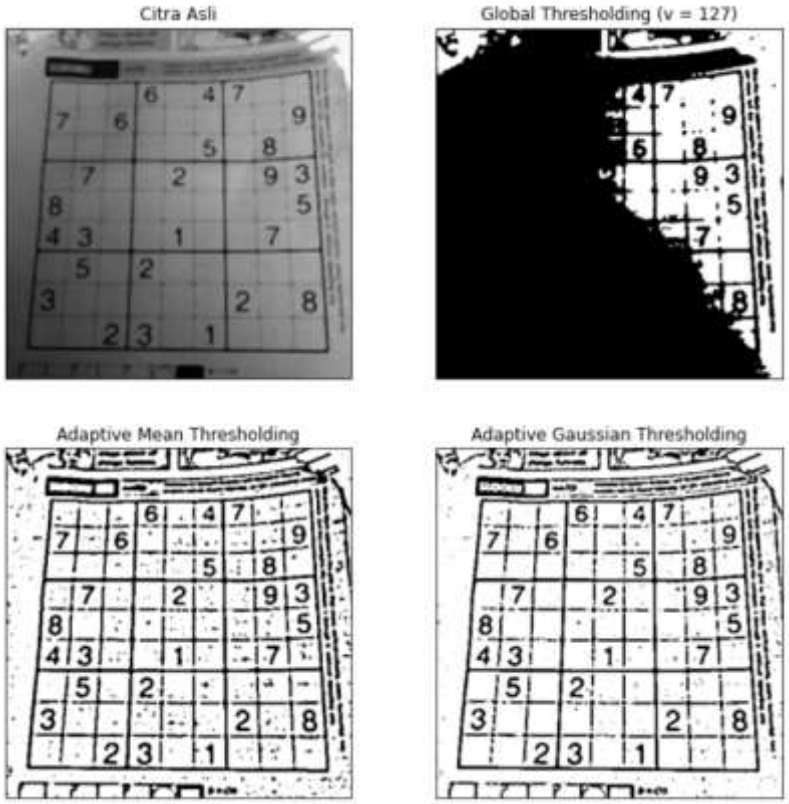


Figure 10. Image resulting from Adaptive Threshold Mean and Gaussian Window

C.4 Otsu's Threshold

Otsu's threshold is a very popular automatic segmentation method, even today it is still often used as the standard reliable binary threshold method. This threshold is named according to the name of its founder, Nobuyuki Otsu. In its simplest form, this algorithm will segment pixels into two classes, namely foreground and background. This threshold is determined by minimizing the intensity of intra-class variance, or by maximizing inter-class variance.

Otsu's algorithm iteratively looks for the smallest threshold of intra-class variance, which is defined as the sum of weights of the variance of two classes:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

The weights ω_0 and ω_1 are the probabilities of the 2 classes separated by the threshold t. σ_0^2 and $\sigma_1^2$is the variance of each class.

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

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

Finding the minimum value of the equivalent intra-class variance by finding the maximum value of the inter-class variance.

$$\sigma_B^2(t) = \sigma^2 - \sigma_W^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$
$$= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

Wherein the mean value of the class $\mu_0(t)$, $\mu_1(t)$, and μ_T are:

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

Consider the following examples of calculations for Otsu's algorithm.

The following image shows a 6x6 resolution image consisting of 6 colors, along with the histogram.

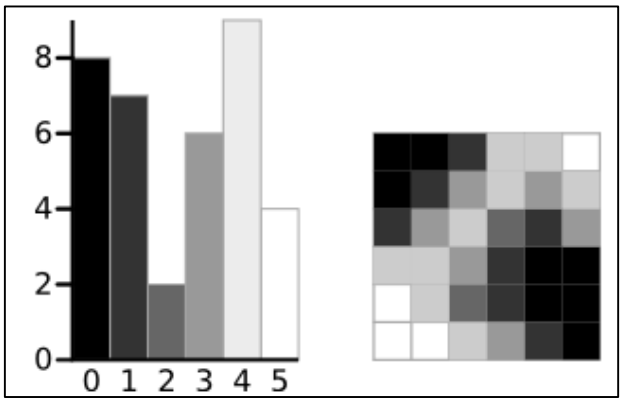


Figure 1 1 . Example of a 6x6 resolution image and its histogram

From the picture above, do the calculation of Weight, Mean, Variance for each background and foreground for each threshold value. The following example is a calculation with a threshold value = 3.

Threshold = 3

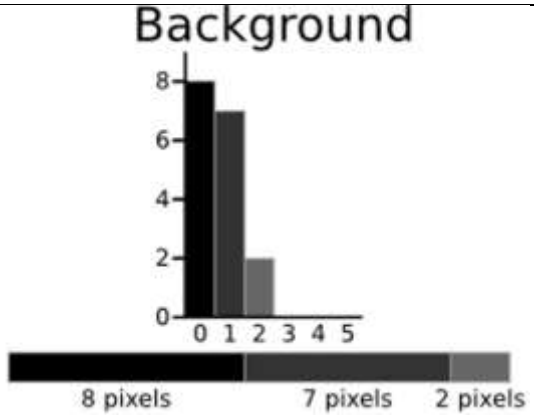


Figure 1 2 . Histogram for background object with T = 3

weight: $\omega_b = \frac{8+7+2}{36} = 0.4722$

mean $\mu_b = \frac{(0\ x\ 8)+(1\ x\ 7)+(2\ x\ 2)}{17} = 0.6471$

variance

$$\sigma_b^2 = \frac{((0-0.6471)^2 x8) + ((1-0.6471)^2 x7) + ((2-0.6471)^2 x2)}{17}$$

$$= \frac{(0.4187x8) + (0.1246x7) + (1.8304x2)}{17} = \mathbf{0.4637}$$

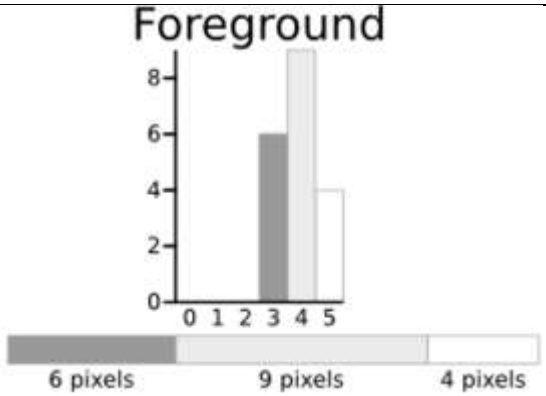


Figure 1 3 . Histogram for Foreground object with T = 3

weight: $\omega_f = \frac{6+9+4}{36} = 0.5278$

mean $\mu_f = \frac{(3\ x\ 6)+(4\ x\ 9)+(5\ x\ 4)}{19} = 3.8947$

variance

$$\sigma_f^2 = \frac{((3-3.8947)^2 x6) + ((4-3.8947)^2 x9) + ((5-3.8947)^2 x4)}{19}$$

$$= \frac{(4.8033x6) + (0.0997x9) + (4.8864x4)}{19} = \mathbf{0.5152}$$

Furthermore, the within class variance of 2 variances (background and foreground) can be calculated by:

Within class variance:

$\sigma_W^2 = \omega_b \sigma_b^2 + \omega_f \sigma_f^2 = 0.4722\ x\ 0.4637 + 0.5278\ x\ 0.5152 = \mathbf{0.4909}$

This calculation is performed for every possible threshold value (0 < = T < = 5). The **smallest** within class variance value will be the selected threshold.

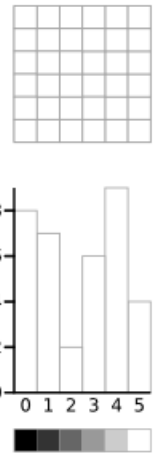
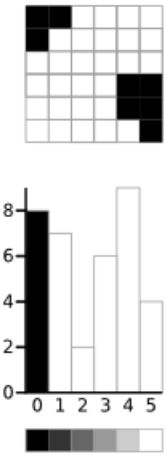
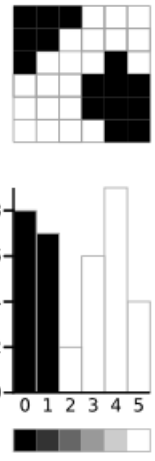
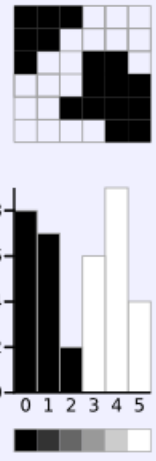
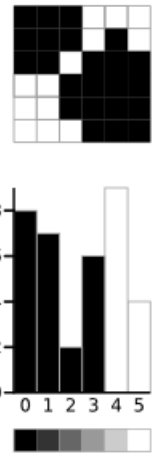
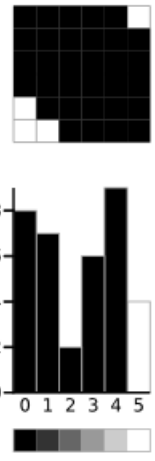
Threshold	T=0	T=1	T=2	T=3	T=4	T=5
						
Weight, Background	W_b = 0	W_b = 0.222	W_b = 0.4167	W_b = 0.4722	W_b = 0.6389	W_b = 0.8889
Mean, Background	M_b = 0	M_b = 0	M_b = 0.4667	M_b = 0.6471	M_b = 1.2609	M_b = 2.0313
Variance, Background	σ^2_b = 0	σ^2_b = 0	σ^2_b = 0.2489	σ^2_b = 0.4637	σ^2_b = 1.4102	σ^2_b = 2.5303
Weight, Foreground	W_f = 1	W_f = 0.7778	W_f = 0.5833	W_f = 0.5278	W_f = 0.3611	W_f = 0.1111
Mean, Foreground	M_f = 2.3611	M_f = 3.0357	M_f = 3.7143	M_f = 3.8947	M_f = 4.3077	M_f = 5.000
Variance, Foreground	σ^2_f = 3.1196	σ^2_f = 1.9639	σ^2_f = 0.7755	σ^2_f = 0.5152	σ^2_f = 0.2130	σ^2_f = 0
Within Class Variance	σ^2_W = 3.1196	σ^2_W = 1.5268	σ^2_W = 0.5561	σ^2_W = **0.4909**	σ^2_W = 0.9779	σ^2_W = 2.2491

Figure 1 4 . Within Class Variance calculation for all possible T values

The following image shows the results of the image that is threshold with Otsu's (T = 3)

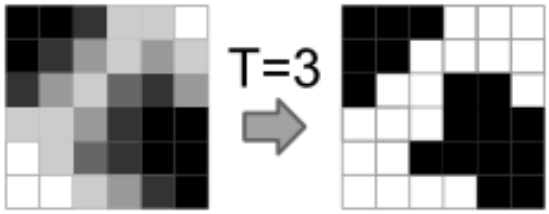


Figure 15. Otsu's Threshold image with a value of T = 3

Determination of the selected threshold value can also be done using the Between class variance calculation, where the **largest** Between class variance value is the selected threshold.

Between class variance:

$$\sigma_B^2 = \sigma^2 - \sigma_W^2$$
$$= \omega_b(\mu_b - \mu)^2 + \omega_f(\mu_f - \mu)^2 \qquad (dimana\ \mu = \omega_b\mu_b + \omega_f\mu_f)$$
$$= \omega_b\omega_f(\mu_b - \mu_f)^2$$

Threshold	T=0	T=1	T=2	T=3	T=4	T=5
Within Class Variance	σ^2_W = 3.1196	σ^2_W = 1.5268	σ^2_W = 0.5561	σ^2_W = 0.4909	σ^2_W = 0.9779	σ^2_W = 2.2491
Between Class Variance	σ^2_B = 0	σ^2_B = 1.5928	σ^2_B = 2.5635	σ^2_B = 2.6287	σ^2_B = 2.1417	σ^2_B = 0.8705

Otsu's is provided in the OpenCV library (cv.THRESH_OTSU) which is usually combined with a binary threshold (cv.THRESH_BINARY). This library is called with the cv.threshold object. The following is an example code of using Otsu's with the OpenCV library.

```
# Dengan Library
filename = ('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/lena_gs_
lc2.jpg')
img = cv.imread(filename,0)
blur = cv.GaussianBlur(img,(5,5),0)
thresh = 127

ret,th1 = cv.threshold(blur,thresh,255,cv.THRESH_BINARY)
ret2,th2 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

x = ("Otsu's Thresholding dgn library (v = ")+str(ret2)+")"
titles = ['Citra Asli', 'Global Thresholding (v = 127)', x]
citra3 = [blur, th1, th2]

plt.figure(figsize = (10,10))
plt.subplot(2,2,1),plt.hist(blur.ravel(),256,[50,200])
plt.vlines(ret,0,40000,colors='red')      #garis vertikal merah menunjuka
n threshold global 127
plt.vlines(ret2,0,40000,colors='black')   #garis vertikal hitam menunjukk
an threshold 92 hasil otsu's
plt.title('Histogram Citra Asli')
for i in range(len(citra3)):
    plt.subplot(2,2,i+2),plt.imshow(citra3[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

The following image shows the results of the code above. Gari s red vertical histogram showing the threshold value 127. From the figure it is seen that the color of the left side of the red line will be changed all the colors to black, so that the threshold will be black all. In contrast to the results of otsu's, it can still separate the two classes for background and foreground. The black vertical line shows the threshold value generated by otsu's. It can be seen that the line can still divide the histogram into two parts, namely background and foreground.

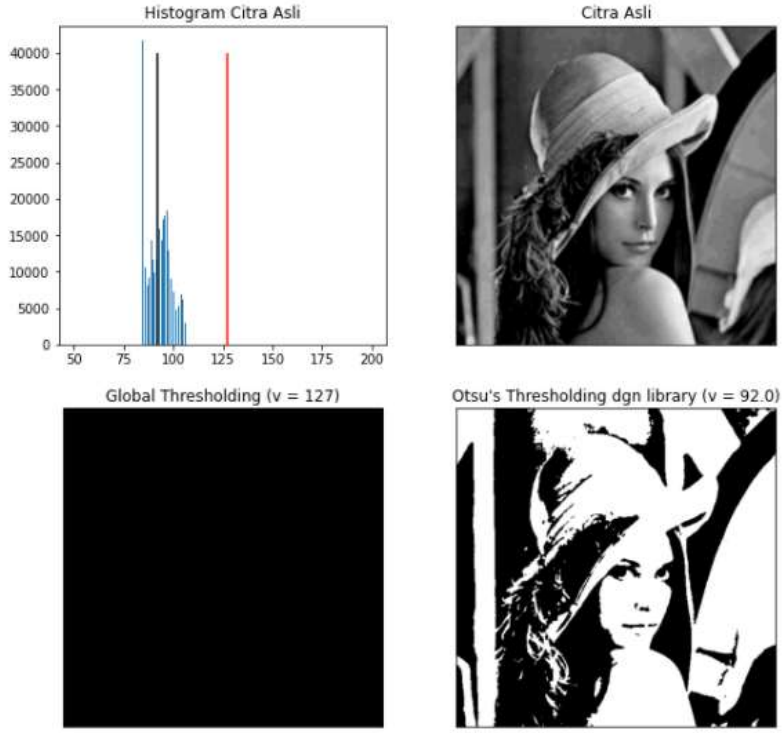


Figure 16. The image of the threshold results using the binary threshold and Otsu's

The following code will display the results of Otsu's threshold without a Gaussian Filter and using a Gaussian Filter.

```
filename = ('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/noisy2.png')
img = cv.imread(filename,0)

#Global Thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# Otsu's thresholding setelah dilakukan Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
#plotting semua image
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Image Noisy Asli','Histogram','Global Thresholding (v=127)',
          'Image Noisy Asli','Histogram',"Otsu's Thresholding",
          'Image dgn Gaussian Filter','Histogram',"Otsu's Thresholding"]
plt.figure(figsize = (15,10))
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

The following image shows the results of the code above. It can be concluded that by first filtering the input image, Otsu's threshold results can be better.

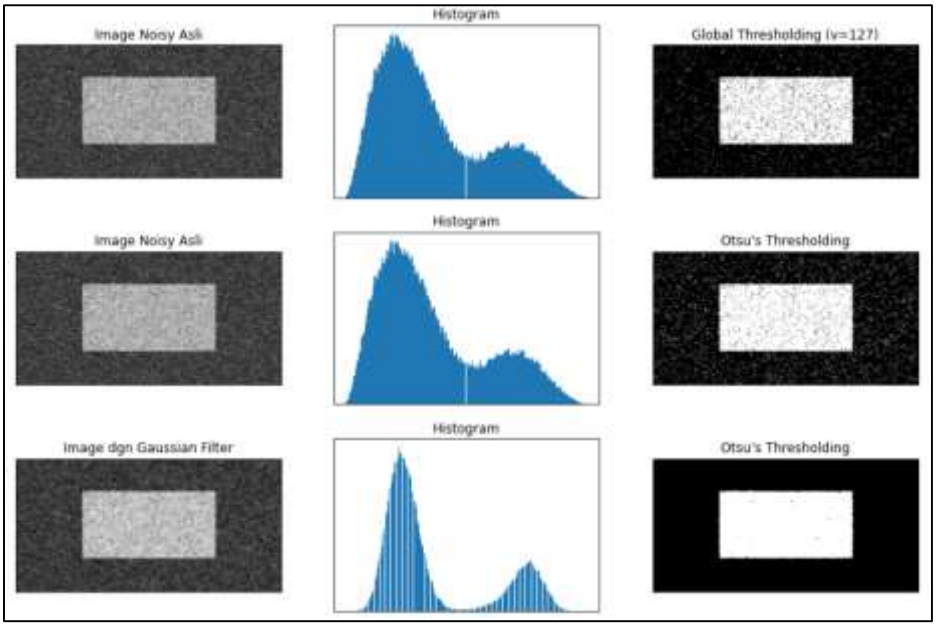


Figure 17. Otsu's image without preprocess and preprocess Gaussian Filter

C.5 Sementasi Citra menggunakan K-Means

K-Means is a clustering method commonly used to segment images by determining the number of clusters we expect. The following is a general explanation link for K-Means:

- https://youtu.be/_aWzGGNrcic
- https://stanford.edu/class/engr108/visualizations/kmeans/kmeans.html

The following is the code for using K-Means in image segmentation.

```
#KMeans Image Segmentation
filename = ('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/jungle.png'
)
img = cv.imread(filename)
img = cv.cvtColor(img,cv.COLOR_BGR2RGB)
'''
we will use the function cv.kmeans () that asks array 2D as input, while
the image of the original is an array of 3D
Next we need to flatten the input image array
'''
#reshape array to 2D form
pixel_values = img.reshape((-1, 3))
# convert to float
pixel_values = np.float32(pixel_values)



'''
```

```
The condition for stopping Dr. KMeans' iteration is if the centroid has not
shiftedtoo much position between current interaction with the previous iter
ation (convergent). Because the amount of data that is great, we will stop
iterating when the number of iterations = 100 or epsilon (the difference
between the position of the centroid skrg with the position of the centroid
in the iteration before) < 0.2
'''
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100, 0.2)
'''
If you look at the original image , there are 3 main colors (green, blue,
and white / orange). for this experiment we will use 3 clusters for this
image
'''
k = 3
_, labels, (centers) = cv.kmeans(pixel_values, k, None, criteria, 10, cv.KM
EANS_RANDOM_CENTERS)
#Convert centroid point to an integer
centers = np.uint8(centers)
#flattening label array
labels = labels.flatten()
#konversi warna pixel asli kewarna dari tiap centroidnya
segmented_image = centers[labels.flatten()]
# Reshape to the original image
segmented_image = segmented_image.reshape(img.shape)
plt.figure(figsize = (20,20))
plt.subplot(1,2,1),plt.imshow(img)
plt.subplot(1,2,2),plt.imshow(segmented_image)
```

The following is the result of the code above

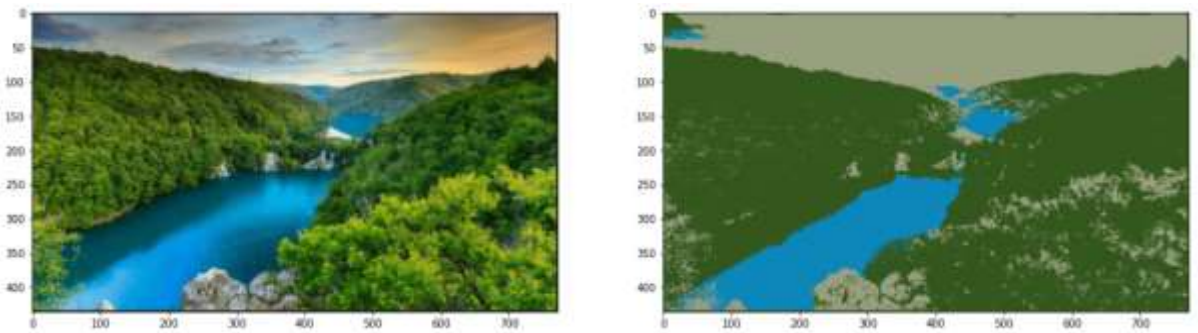


Figure 18. The results of image segmentation using K-Means Clustering

The following code is a continuation of the code above which is used to change the color for a particular cluster.

```
# ubah pixel di cluster 2 menjadi hitam
masked_image = np.copy(img)
# konvert ke bentuk vektor
masked_image = masked_image.reshape((-1, 3))
# cluster yang diubah
cluster = 2
masked_image[labels == cluster] = [0, 0, 0]
# konvert ke bentuk asli
masked_image = masked_image.reshape(img.shape)

plt.figure(figsize = (20,12))
plt.subplot(2,2,1),plt.imshow(img)
plt.subplot(2,2,2),plt.imshow(segmented_image)
plt.subplot(2,2,3),plt.imshow(masked_image)
```

The following is an image of the result of the code above.

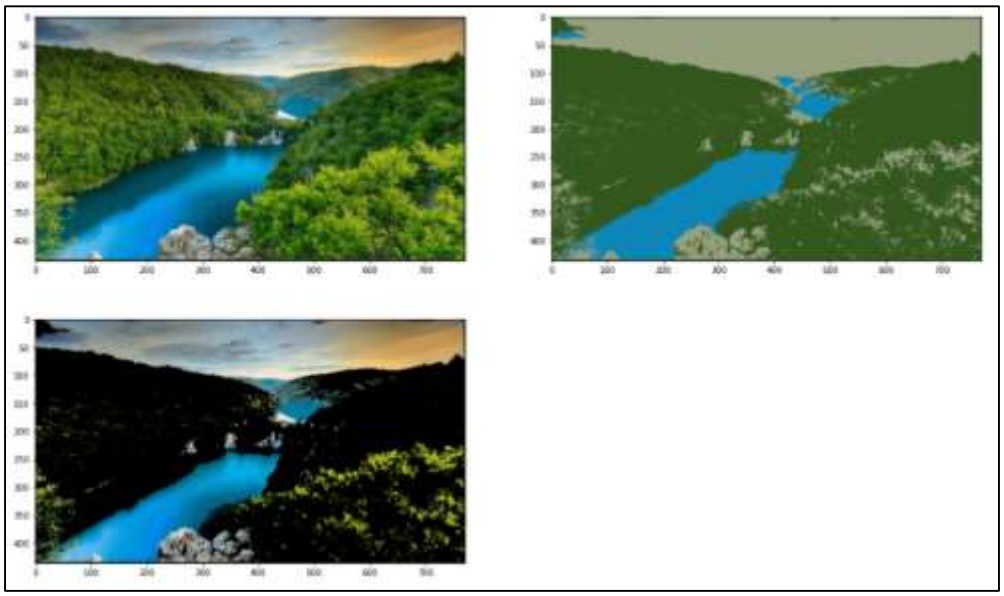


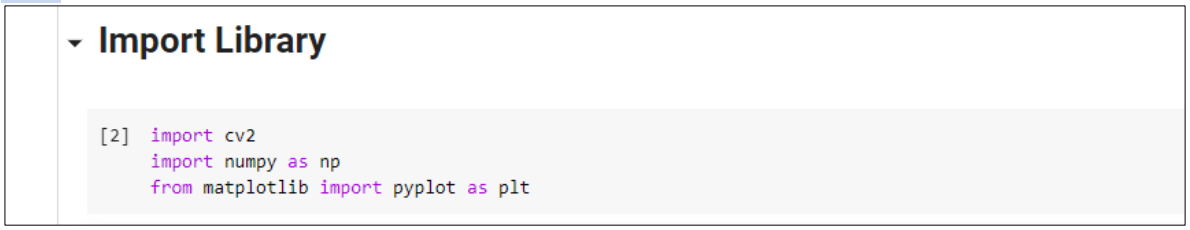
Figure 19. Results of changes in color values in a particular cluster

D. PRACTICUM

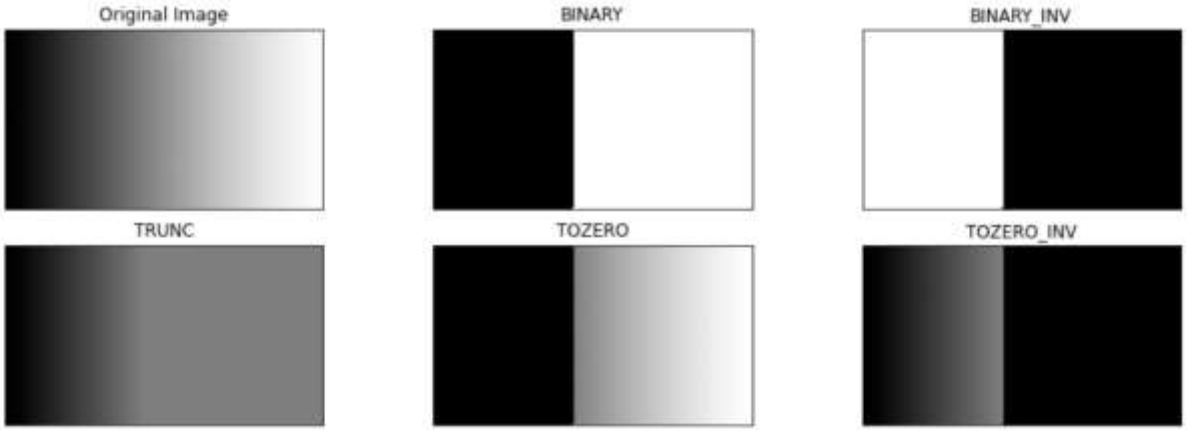
1. Go to https://colab.research.google.com/ . After making sure that Google Colab is connected to your Github , continue by selecting the repository that was used in the lab last week , rename the file to "Week 10 .ipynb".

Then import the existing folder on your Drive as follows.

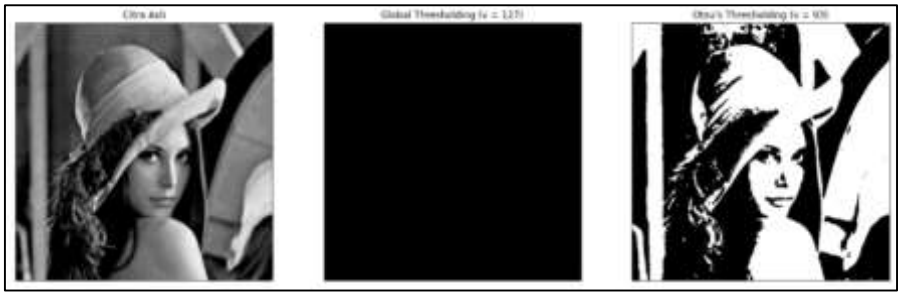
2. Import the following libraries that will be used during the following week 6 practicum trial.



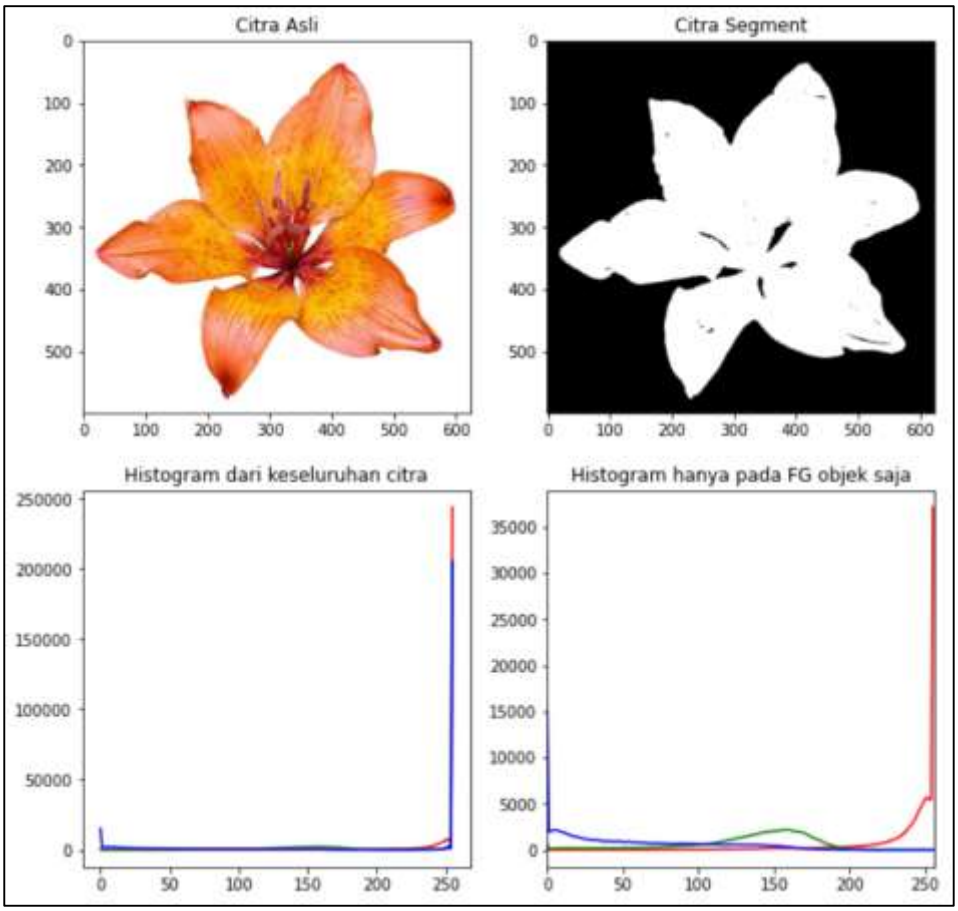
3. Manually create a Global Threshold (BINARY, BINARY_INV, TRUNC, TOZERO, TOZERO_INV) according to the description from the chart shown above.



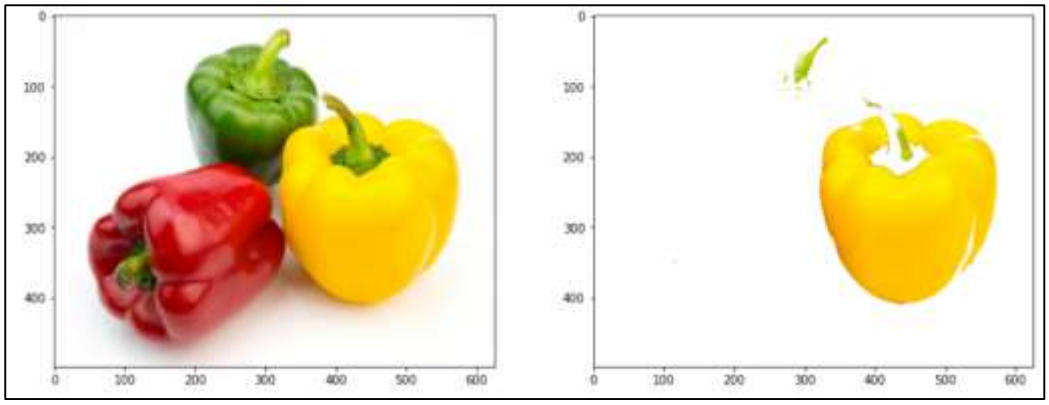
4. Create Otsu Thresholding without using the Library. Also display the threshold value when you use Otsu's, as shown in the following image . (use the image **lena_gs_lc2.jpg** so that it really looks different between the otsu's results and the usual global threshold)



5. Create a histogram from a segmented image, the histogram is only in the foreground image. Use the Lily or Leaves image provided in the images folder .
   Instructions:
   a. You can use cv.calcHist to display a histogram.
   b. Open the folloiwng link https://opencv-tutorial.readthedocs.io/en/latest/histogram/histogram.html
   c. From the link, note that cv.calcHist has one of the parameters, namely mask. If set to **None**, the entire image histogram will be counted. If we specify a mask, only the part of the image masked in **white** will be calculated for the histogram (from the example below it is called **Image Segment**).



6. Perform color segmentation on the image "peppers.jpg", display only yellow colors . (Hint: you can use K-Means to display only certain colors)



When you display a certain color, explain the problem you are facing and why it happened.

……..
……..
……..
……..
……..

7. Open the crossword.jpg file. With the knowledge of thresholding that you have learned. Perform binary thresholding with the best results in your opinion. Copy the code and image results in this module.

--- GOOD LUCK ---