DEPARTMENT OF PHYSICS
UNIVERSITY OF COLOMBO

PH 3034- Digital Image Processing – I
2021

Image scaling and Otsu method

Name: Umeshika Dissanayaka

Index Number: s14320

Date: 18/10/2021

TABLE OF CONTENTS

1 INTRODUCTION 1
1.1 Objectives 2
2 METHODOLOGY 2
2.1 Exercise 1: Nearest-neighbor interpolation 2
2.2 Exercise 2: Bilinear interpolation 4
2.3 Exercise 3: Bicubic interpolation 7
2.4 Exercise 4: Otsu's method for Image Thresholding 10
3 RESULTS AND ANALYSIS 13
3.1 Exercise 1: Nearest-neighbor interpolation 13
3.2 Exercise 2: Bilinear interpolation 14
3.3 Exercise 3: Bicubic interpolation 15
3.4 Exercise 4: Otsu's method for Image Thresholding 16
4 REFERENCES 17

TABLE OF FIGURES

Figure 2-1 The graphical representation of Nearest Neighbor Interpolation for enlarge image(reference: https://www.imageeprocessing.com/2017/11/nearest-neighbor-interpolation.html) 2
Figure 2-2 The code for the nearest neighbour method to get zoom image 4
Figure 2-3 The (x,y) coordinate system for implementing the Bilinear interpolation (reference:https://www.programmersought.com/article/57674202296/) 4
Figure 2-4 The code for bilinear interpolation method to get zoom image 6
Figure 2-5 The graphical representation of Bicubic interpolation implementation image (reference: https://www.programmersought.com/article/77525868180/) 7
Figure 2-6 The code for the bicubic interpolation method to get zoom image 10
Figure 2-7 The code of the OTSU method to get threshold image 12
Figure 3-1 The original image 13
Figure 3-2 The Zoom image using the nearest neighbour method 13
Figure 3-3 The original image 14
Figure 3-4 The Zoom image using Bilinear Interpolation method 14
Figure 3-5 The original image 15
Figure 3-6 The Zoom image using the Bicubic Interpolation method 15
Figure 3-7 The original flower image 16
Figure 3-8 threshold flower image that was taken by using OTSU method 16

1 INTRODUCTION

In the modern world, technology is revolute and replaced by another technology generation by generation. Consequently, computer graphics and digital imaging are also a revolution day by days such as medical imaging and graphic and video applications in electronic devices. Hence the digital image processing and computer graphics are needed to do the image scaling that means resizing the image with greater than or less than the original size of the image. At that time image scaling techniques were created by using computer numerical analysis. Therefore, one of the main methods is used for image scaling is an interpolation that can be categorised as linear and nonlinear interpolation methods. The nearest neighbour interpolation is a linear interpolation method which is a low-intricacy calculation, yet it is given about enlarge image with impeding and associating antiquities. The best image scaling method is the bilinear interpolation method it is implemented by considering the objective pixel that can be acquired by utilizing the linear interpolation in both horizontal and vertical ways. A further well-known polynomial-based technique is the bicubic interpolation method which utilizes visualization of the cubic model in the 2D ordinary lattice. There are several examples of nonlinear interpolation methods such as bilateral filter, curvature interpolation, and weighted median interpolation. Therefore using these numerical methods and the pixels in an image, the image can be resized within scaling factor. This report first part consists of the study of image scaling using nearest neighbour interpolation, bilinear interpolation and bicubic interpolation in the first three exercises.

Secondly, this report consists of the study of image thresholding using the Otsu method. The OTSU method Algorithm is invented by Nobuyuki Otsu which is used for image binarization of black and white of two pieces of highly contrasting. Separating the image pixel intensity of two levels which are background and foreground. Using the greater the variance, the lower the correlation, the largest variance of grey value pixel intensity is founded that is the binarized separator limit which is called the variance maximum threshold value in the image. Then using that binarized separator limit, the binary image can be obtained which is introduced as image thresholding using OTSU method Algorithm.

Therefore, the main purpose of this practical was to Implement and visualize the image resizing and Otsu method image thresholding by using the open software of the high-level programming language of Python 03. Python 03 is an object-oriented scripting language, and it has been easy to hence the excellent advantage of learning python is that has permitted debugging of snippets of code and interactive testing. Currently, python 03 is utilized in plenty of fields of digital image processing the fact that everybody can download openly and various and different principal standard image processing libraries and packages are accessible in python which can import and get the yield easily such as Scikit-image, Opencv, NumPy and matplotlib. Not just that it is used English punctuations which are straightforward and learn.

1.1 Objectives

- The main objective of this practical was to gain brief knowledge about the mechanism of resizing images in the computers using interpolation methods such as the nearest neighbour, bilinear, bicubic interpolation methods.
- The image scaling algorithms were implemented by using the nearest neighbour, bilinear, bicubic interpolation methods and high-level programming language of Python 03.
- Secondly this practical was gain brief knowledge about the mechanism of Image Thresholding Technique in the computers using Otsu's Thresholding Technique
- Furthermore, it was more familiarized with how to used Opencv libraries and python coding.

2 METHODOLOGY

2.1 Exercise 1: Nearest-neighbor interpolation

Firstly, the numerical background of the Nearest-neighbor interpolation was identified for image scaling and implemented the mathematical functions and steps were before creating the python code. Therefore, an image is a two-dimensional array with a pixel that is used for zooming to get the new scaled image. The new scale image could be enlarged using the scaling ratio.

$$scaling\ ratio\ for\ height = \frac{original\ image\ height}{Zoomed\ image\ height}$$

$$scaling\ ratio\ for\ width = \frac{original\ image\ width}{Zoomed\ image\ width}$$

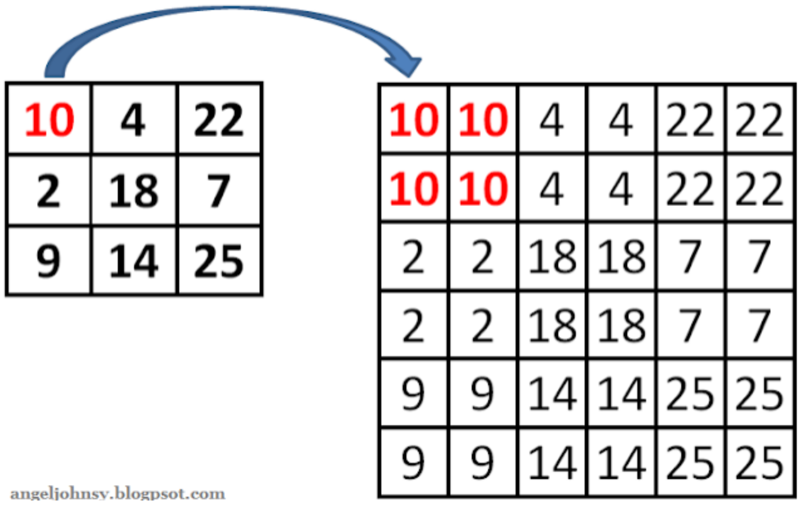

Figure 2-1 The graphical representation of Nearest Neighbor Interpolation for enlarge image(reference: https://www.imageeprocessing.com/2017/11/nearest-neighbor-interpolation.html)

In this figure has 6/3=2 the scaling factor because the original image has (3x3) pixels and enlarge image has (6X6) pixels and each pixel are multiplied with the scaling factor in Nearest Neighbor. Each value inside the pixels represents the value of colour intensity. Furthermore, the zoomed image has coordinated of $(x \times \text{scale\_x}, y \times \text{scale\_y})$ of integer values used to implement the result of nearest neighbour interpolation that pixelates every pixel with the nearest pixel in the zoom image.

The python library was imported and loaded the original image using OpenCV packages. After that special function was defined which was named “nearestneighbor” and using that function the input image was enlarged with double size. Finally, the code execution time was calculated using “datetime library”

```python
"""
Created on Thu Oct 14 18:53:02 2021
@author: Umeshika
"""
#import the libraries
import cv2
import numpy as np
from datetime import datetime
#The variable is create to get the starting time when the code is run
starttime=datetime.now()
#load the image
img = cv2.imread("a.jpg")
#The function is defined to zoom
def nearestneighbor(img,new_height,new_width):
    h,w,_=img.shape#height and width original image
    #RGB image zero matrix is taken
    rgbimg=np.zeros((new_height,new_width,3),dtype=np.uint8)
    for i in range(new_height-1):
        for j in range(new_height-1):
            # scale factor
            sh = (h * 1.0) / new_height
            sw = (w * 1.0) / new_width
            # insert pixel to the new img
            '''the zoom image coordinates(i, j) and pixelate it with
            the coordinate of original image'''
            zoom_x=round(i*sh)
            zoom_y=round(j*sw)
            #rgb channel image is taken
            rgbimg[i,j]=img[zoom_x,zoom_y]
    return rgbimg

#image double size of height and width
h2 = img.shape[0]*2
w2 = img.shape[1]*2
#the zoomed image is taken
zoom = nearestneighbor(img,h2,w2)
cv2.imshow("The Zoom image using nearest neighbor method", zoom)
cv2.imshow("Original image", img)
```

```
39 cv2.imwrite('Original_image.png', img)
40 cv2.imwrite('Zoomedimage.png', zoom)
41 cv2.waitKey(0)
42
43 """The code executing time is calculated using
44  difference of the start time and end time"""
45 print (datetime.now()-starttime)
```

Figure 2-2 The code for the nearest neighbour method to get zoom image

In this code line, 14 is shown a defined function of “nearestneighbor” which has three input elements of an input image, height value for zoom image and width for zoom image. Inside the function height and width values of the original image were taken using “.shape” command. Then two dimensional zero matrices were taken using “np.zeors” command. The zoom image coordinates (i, j) were pixelated using double for-loop within range to new height value and width value therefore, inside the loop each pixel value was taken the rounded multiple of scale factor and index i or j. The scale factors were taken using above mentioned theoretical way. And the last part of this code shows the output of the zoom image which was given the double size of the original image.

The code execution time= 03.384608 s

2.2 Exercise 2: Bilinear interpolation

Firstly, the numerical background of the Bilinear interpolation was identified for image scaling and implemented the mathematical functions and steps were before creating the python code.

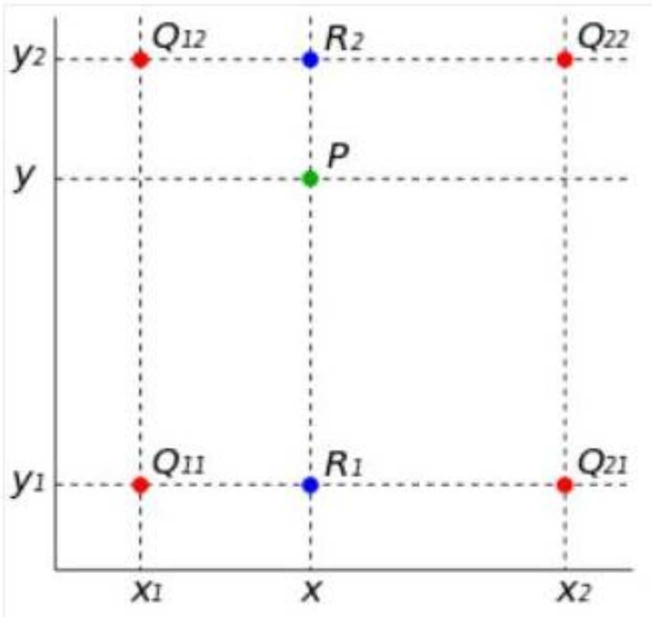

Figure 2-3 The (x,y) coordinate system for implementing the Bilinear interpolation (reference:https://www.programmersought.com/article/57674202296/)

This figure shows that bilinear interpolation operations in the cartesian plane. Let's consider known examples with the points of Q11, Q12, Q21, and Q22 point to implement for known coordinates of (0, 0), (0, 1), (1, 0) and (1, 1). Then the reduced bilinear interpolation formula for point P(x,y) was given as,

$$f(x,y) \approx f(0,0)(1-x)(1-y) + f(1,0)x(1-y) + f(0,1)(1-x)y + f(1,1)xy$$

The above formula can be written as matrix form,

$$f(x,y) \approx [(1-x) \;\; x] \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

The result of the above formula can be taken as $b_1 + b_2x + b_3y + b_4xy$ and each coefficient are represented the data point of considering the function of f(x,y).

$$b_1 = f(0,0)$$

$$b_2 = f(1,0) - f(0,0)$$

$$b_3 = f(0,1) - f(0,0)$$

$$b_4 = f(1,1) - f(1,0) - f(0,1) + f(0,0)$$

Therefore, function f(x,y) is an image of a two-dimensional array with a pixel that is used for zooming to get the new scaled image. The new scale image could be enlarged using the scaling ratio.

$$scaling\ ratio\ for\ height = \frac{Zoomed\ image\ height}{original\ image\ height}$$

$$scaling\ ratio\ for\ width = \frac{Zoomed\ image\ width}{original\ image\ width}$$

The python library was imported and loaded the original image using OpenCV packages. After that special function was defined which was named "bilinear_interpolation" and using that function the input image was enlarged with double size. Finally, the code execution time was calculated using "datetime library"

```python
"""
Created on Sat Oct 16 22:22:17 2021
@author: umeshika
"""
#import the libraries
import cv2
import numpy as np
from datetime import datetime
#The variable is create to get the starting time when the code is run
starttime=datetime.now()
#load the image
img=cv2.imread("a.jpg")
```

```
13
14 #The bilinear interpolation function is defined to zoom
15 def bilinear_interpolation(img,new_height,new_width):
16     #height and width original image
17     height,width,channels =img.shape
18     #RGB image zero matrix is taken
19     zoom_img=np.zeros((new_height,new_width,channels),np.uint8)
20     value=[0,0,0]
21     # scale factor
22     sh=new_height/height
23     sw=new_width/width
24     for i in range(new_height):
25         for j in range(new_width):
26             #x,y coordinates with float values
27             x = i/sh
28             y = j/sw
29             p=(i+0.0)/sh-x
30             q=(j+0.0)/sw-y
31             #x,y coordinates with integer values
32             x=int(x)-1
33             y=int(y)-1
34             '''the zoom image coordinates(i, j) and pixelate it with
35             the coordinate of original image'''
36             for k in range(3):
37                 if x+1<new_height and y+1<new_width:
38                    value[k]=  int(img[x,y][k]*(1-p)*(1-q)\
39                                 +img[x,y+1][k]*q*(1-p)\
40                                 +img[x+1,y][k]*(1-q)*p\
41                                 +img[x+1,y+1][k]*p*q)
42             #rgb channel image is taken
43             zoom_img[i, j] = (value[0], value[1], value[2])
44     return zoom_img
45 #image double size of height and width
46 h2 = img.shape[0]*2
47 w2 = img.shape[1]*2
48 #the zoomed image is taken
49 zoom=bilinear_interpolation(img,h2,w2)
50 cv2.imshow("Bilinear Interpolation",zoom)
51 cv2.imshow("image",img)
52 cv2.waitKey(0)
53 """The code executing time is calculated using
54  difference of the start time and end time"""
55 print (datetime.now()-starttime)
```

Figure 2-4 The code for bilinear interpolation method to get zoom image

In this code line, 15 is shown a defined function of "bilinear_interpolation" which has three input elements of the input image, height value for zoom image and width for zoom image. Inside the function height and width values of the original image were taken using ".shape" command. Then two dimensional zero matrices were taken using "np.zeors" command. The zoom image coordinates (i, j) were pixelated using double for-loop within range to new height value and width value after that, inside the loop another for loop was included to take the four-point coordinates of the image function using the above-mensioned interpolation formula. The scale factors were taken using above mentioned theoretical way. And the last part of this code shows the output of the zoom image which was given the double size of the original image.

The code execution time= 11.279917 s

2.3 Exercise 3: Bicubic interpolation

Firstly, the numerical background of the Bicubic interpolation was identified for image scaling and implemented the mathematical functions and steps were before creating the python code. Let's consider 3rd-degree polynomials two Bicubic basis functions for 0=<x<1 and 1<=x<2 otherwise 0.

$$W(x) = \begin{cases} x^3 - 2x^2 + 1 \, for \; 0 =< \mathrm{x} < 1 \\ -x^3 + 5x^2 - 8x + 4 \, for \; 1 =< \mathrm{x} < 2 \end{cases}$$

Otherwise 0

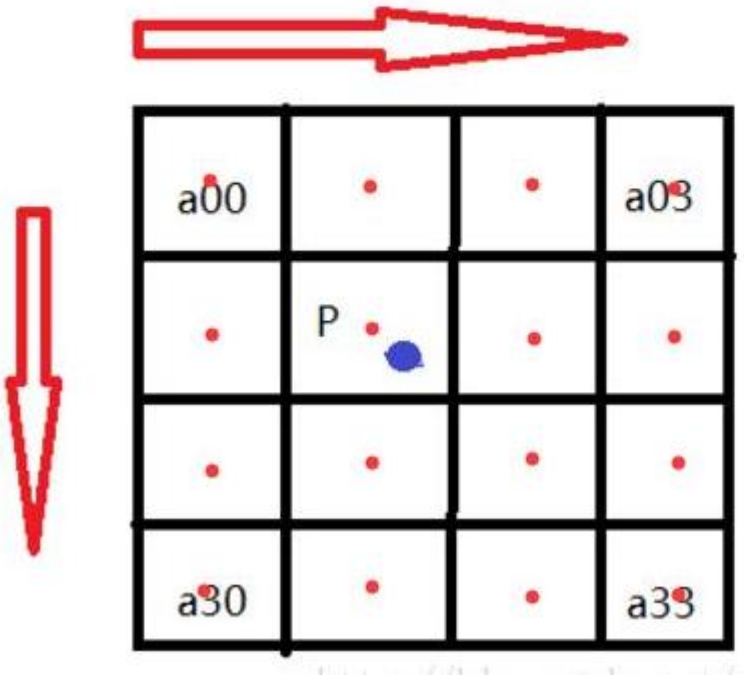

Figure 2-5 The graphical representation of Bicubic interpolation implementation image (reference: https://www.programmersought.com/article/77525868180/)

This figure shows that 16-pixel images with corner points are represented coefficient values and the enlarged image can be obtained using the weight of 16-pixel points. Therefore, BiCubic basis function w is used to take the weight of 16-pixel points. Let's consider a 2-dimensional image of bicubic interpolation which have B(X, Y) pixel values ,

$$B(X,Y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} \times w(i) \times W(j)$$

Therefore, using the above-mentioned function is rewritten in a matrix form, then a two-dimensional array with a pixel that is called an image can be taken. In addition, the new scale image could be enlarged using the scaling ratio and the above-mentioned method.

$$scaling\ ratio\ for\ height = \frac{original\ image\ height}{Zoomed\ image\ height}$$

$$scaling\ ratio\ for\ width = \frac{original\ image\ width}{Zoomed\ image\ width}$$

The python library was imported and loaded the original image using OpenCV packages. After that special function was defined which was named “bicubic_interpolation” and using that function the input image was enlarge with double size. Finally, the code execution time was calculated using “datetime library”

```python
"""
Created on Thu Oct 14 21:55:30 2021
@author: Umeshika
"""
#import the libraries
import cv2
import numpy as np
from datetime import datetime
#The variable is create to get the starting time when the code is run
starttime=datetime.now()
#load the image
img=cv2.imread("a.jpg")
#define function for polynomial technique with two different formula
def Bicubic(x):
    x = np.abs(x)
    #consider the third oder degree polynoial function for 0=<x<1 range
    if 0 <= x < 1:
        return 1 - 2 * x * x + x * x * x
    '''consider the  another third oder degree
        polynoial function for 0=<x<1 range'''
    if 1 <= x < 2:
        return 4 - 8 * x + 5 * x * x - x * x * x
    else:
        return 0
#The bicubic interpolation function is defined to zoom
def bicubic_interpolation(img,new_height,new_width):
    #height and width original image
    height,width,channels =img.shape
    #RGB image zero matrix is taken
    zoom_img=np.zeros((new_height,new_width,channels),np.uint8)
    # scale factor
    sh=new_height/height
    sw=new_width/width
    for i in range(new_height):
        for j in range(new_width):
            #x,y coordinates with float values
            x = i/sh
            y = j/sw
            #x,y coordinates with integer values
            p=(i+0.0)/sh-x
            q=(j+0.0)/sw-y
            x=int(x)-2
            y=int(y)-2
            '''the zoom image coordinates(i, j) and pixelate it with
            the coordinate of original image'''
            #first element
            A = np.array([
                [Bicubic(1 + p), Bicubic(p), Bicubic(1 - p), Bicubic(2 - p)]])

            if x>=new_height-3:
                new_height-1
            if y>=new_width-3:
                new_width-1
            if x>=1 and x<=(new_height-3) and y>=1 and y<=(new_width-3):

            #second element
                B = np.array([
                    [img[x-1, y-1], img[x-1, y],
                     img[x-1, y+1],
                     img[x-1, y+1]],
                    [img[x, y-1], img[x, y],
                     img[x, y+1], img[x, y+2]],
                    [img[x+1, y-1], img[x+1, y],
                     img[x+1, y+1], img[x+1, y+2]],
                    [img[x+2, y-1], img[x+2, y],
                     img[x+2, y+1], img[x+2, y+1]],])

                #third element
                C = np.array([
                    [Bicubic(1 + q)],
                    [Bicubic(q)],
                    [Bicubic(1 - q)],
                    [Bicubic(2 - q)]])
                #rgb layers are taken
                B_n = np.dot(np.dot(A, B[:, :, 0]), C)[0, 0]
                G_n = np.dot(np.dot(A, B[:, :, 1]), C)[0, 0]
                R_n = np.dot(np.dot(A, B[:, :, 2]), C)[0, 0]

                #the values ajusts to be in [0,255]
                def adj(bin_size):
                    if bin_size > 255:
                        bin_size = 255
                    elif bin_size < 0:
```

```
84              bin_size = 0
85          return bin_size
86        #rgb channel image is taken
87        B = adj(B_n)
88        G = adj(G_n)
89        R = adj(R_n)
90        zoom_img[i, j] = np.array([B, G, R], dtype=np.uint8)
91
92
93  return zoom_img
94 #image double size of height and width
95 h2 = img.shape[0]*2
96 w2 = img.shape[1]*2
97 #the zoomed image is taken
98 zoom=bicubic_interpolation(img,h2,w2)
99 cv2.imshow("The Zoom image using bicubic interpolation method",zoom)
100 cv2.imshow("Original image",img)
101 cv2.waitKey(0)
102 """The code executing time is calculated using
103  difference of the start time and end time"""
104 print (datetime.now()-starttime)
```

Figure 2-6 The code for the bicubic interpolation method to get zoom image

In these code lines, 14 and 26 are shown the defined function of "bicubic" and "bicubic_interpolation". The Bicubic function has two different third-order degree polynomial functions for 0=<x<1 range and 1=<x<2 range. "bicubic_interpolation" has three input elements of the input image, height value for zoom image and width for zoom image. Inside the function height and width values of the original image were taken using ".shape" command. Then two dimensional zero matrices were taken using "np.zeors" command. The zoom image coordinates (i, j) were pixelated using double for-loop within range to new height value and width value after that, inside the loop using the bicubic function, the Weighting algorithm could be taken in the four-point coordinates of the image function using the above-mensioned interpolation formula. The scale factors were taken using above mentioned theoretical way. Then the bin values adjusted for 0 and 255. And the last part of this code shows the output of the zoom image which was given the double size of the original image.

The code execution time= 12.897502s

2.4 Exercise 4: Otsu's method for Image Thresholding

Firstly, the numerical background of Otsu's method for Image Thresholding was identified and implemented the mathematical functions and steps were before creating the python code. The OTSU algorithm is mainly based on the same two classes of weighted variance for the Thresholding.

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

Where ω_0 *and* ω_1 are the probabilities and t is a threshold value

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \; and \; \omega_1(t) = \sum_{i=0}^{l-1} p(i)$$

The two weighted class means are obtained using the below formula.

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)} \; and \; \mu_1(t) = \frac{\sum_{i=0}^{l-1} ip(i)}{\omega_1(t)}$$

After that two class variances can be expressed using the above means and weighted probabilities,

$$g = \omega_0(t)\omega_1(t)[\mu_0(t)\mu_1(t)]^2$$

The python library was imported and loaded the original image using OpenCV packages. After that, the image was converted to a grey image using image data. Then special function was defined which was named “otsu” and using that function the input image was thresholding with maximum thresholding value. Finally, the code execution time was calculated using “datetime library”

```python
"""
Created on Thu Oct 14 21:55:30 2021
@author: Umeshika
"""
#import the libraries
from datetime import datetime
import cv2
import numpy as np
from skimage import  color
#The variable is create to get the starting time when the code is run
starttime=datetime.now()
#load the image
img = cv2.imread('b.jpg').astype(np.float32)
#convert data to gray color with float32
gray_img = color.rgb2gray(img).astype(np.float32)
#otsu function is defined
def otsu(gray_img):
    #Image width and height
    h = gray_img.shape[0]
    w = gray_img.shape[1]
    threshold_t = 0
    max_g = 0
    #every grayscale layer is pixelated
    for t in range(255):
        #array directly taken using  Numpy
        #define two arrays for two weight of classes
        x = gray_img[np.where(gray_img < t)]
        x1 = gray_img[np.where(gray_img>=t)]
        #ω0 and ω1 are the proportions of foreground and background
        w0 = len(x)/(h*w)
```

```
31         w1 = len(x1)/(h*w)
32         #foreground mean µ0
33         u0 = np.mean(x) if len(x)>0 else 0
34         #background  mean µ1
35         u1 = np.mean(x1) if len(x1)>0 else 0
36         '''the variance g is calculated using foreground µ0
37         and the background µ1'''
38         g = w0*w1*(u0-u1)**2
39         #consider segmentation
40         if g > max_g :
41             max_g = g
42             threshold_t = t
43     print ('The variance maximum threshold:',threshold_t)
44     # take binary image using global thresholding
45     gray_img[gray_img<threshold_t] = 0
46     gray_img[gray_img>threshold_t] = 255
47     return gray_img
48 #plot the thresholdig image using Otsu's Method
49 otsu_img = otsu(gray_img)
50 cv2.imshow('otsu method image',otsu_img)
51 cv2.waitKey(0)# Wait 3000ms = 3s
52 cv2.destroyAllWindows()
53 """The code executing time is calculated using
54  difference of the start time and end time"""
55 print (datetime.now()-starttime)
```

Figure 2-7 The code of the OTSU method to get threshold image

In this code line, 17 is shown a defined function of "otsu" which has only one input element of the grey image. Inside the function height and width values of the original image were taken using ".shape" command. Then the initial threshold value and weighted variance were initialized as 0. The grey image coordinates were pixelated using the for-loop within range to 0 and 255 because grey image every pixel could be taken using that range. Then two grey image matrices were created in two weighted classes. In lines 30 and 31, the probability of foreground and background were calculated using the length of two grey image matrices dividing the total number of pixels. After that means of foreground and background were calculated using the "np. mean" command. In line 38 the maximizing inter-class variance was calculated using the above mentioned theoretical formula. Then considering each segmentation was converted to the maximum threshold value. And the last part of this code shows the output of the threshold image that was taken by using the OTSU method.

The code execution time= 01.332565 s

3 RESULTS AND ANALYSIS

3.1 Exercise 1: Nearest-neighbor interpolation

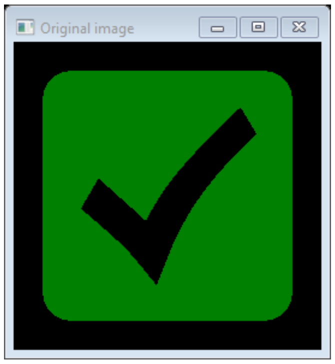

Figure 3-1 The original image

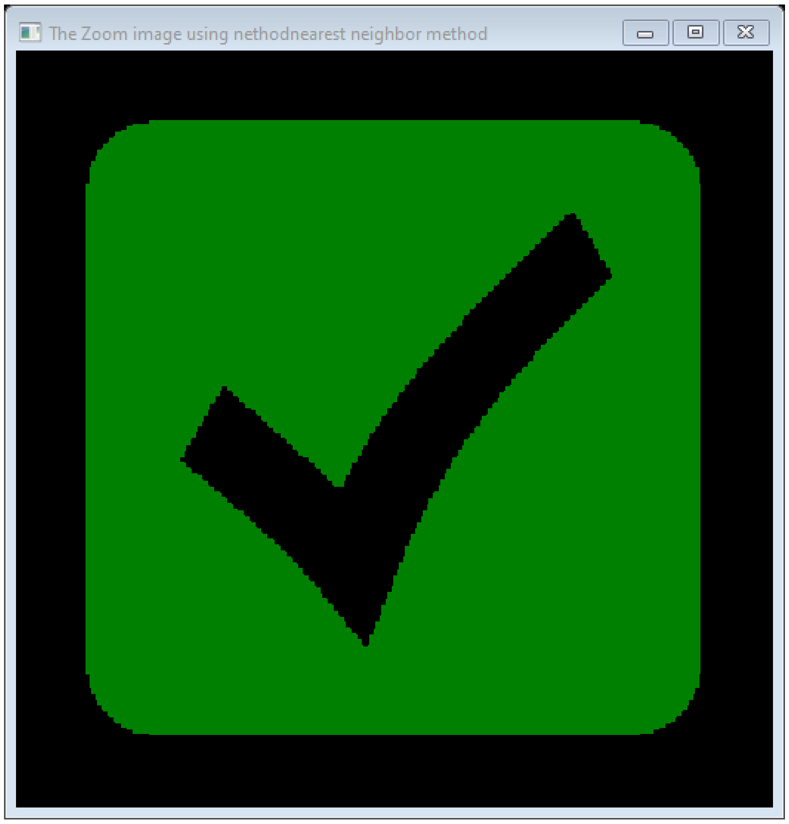

Figure 3-2 The Zoom image using the nearest neighbour method

The original image has a 256X256 pixel dimension. The Zoom image was taken by using the nearest neighbour method that was given the double size of the original image. Obviously, the zoom image is visualized that edges of the right shape are not very smooth or sharp because it is visualized the pixel. Therefore, this analysis concludes that the nearest neighbour method for scaling images is given the hard edges and not given the soft edges because nearest coordinates are taken as rounded coordinates, thus rounding to the nearest neighbour.

3.2 Exercise 2: Bilinear interpolation

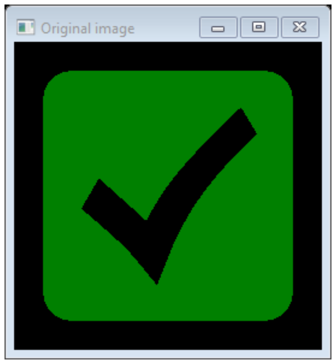

Figure 3-3 The original image

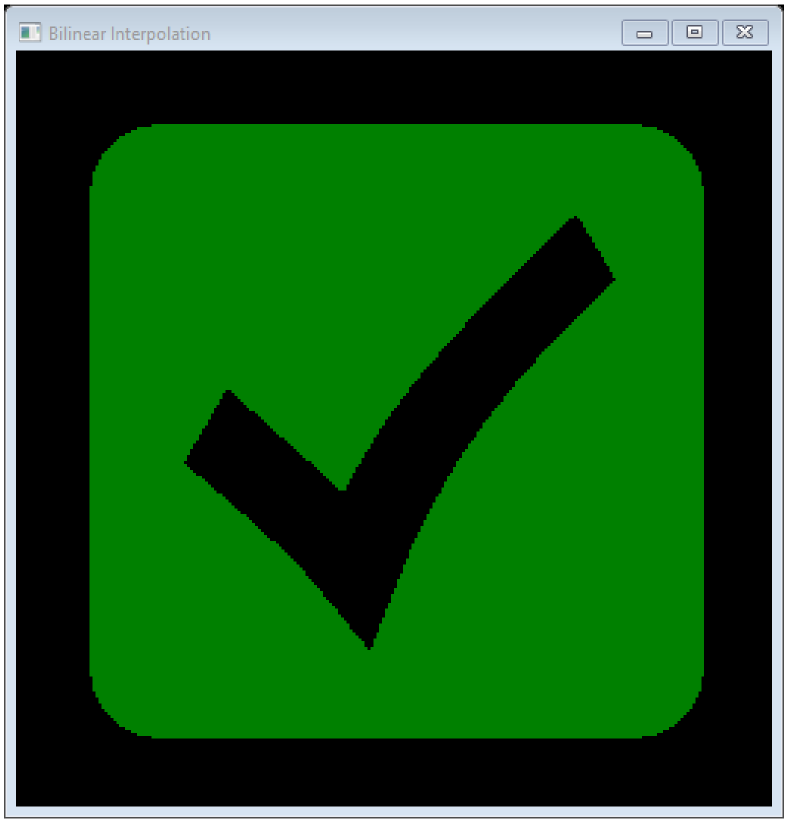

Figure 3-4 The Zoom image using Bilinear Interpolation method

The original image has a 256X256 pixel dimension. The Zoom image which was taken by using the Bilinear Interpolation method was given the double size of the original image. Obviously, the zoom image is visualized that edges of the right shape are smoother or sharper than nearest neighbour method zoomed image because it is less visualized the pixel. Therefore, this analysis concludes that the Bilinear Interpolation method for scaling images is given the little soft edges and not given the proper level lines or hard edges because corresponding coordinates are not taken as rounded or integers, thus floating coordinates of four pixels closest to the corresponding coordinate are used to implement the Bilinear interpolation calculations.

3.3 Exercise 3: Bicubic interpolation

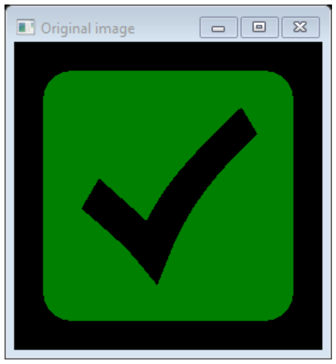

Figure 3-5 The original image

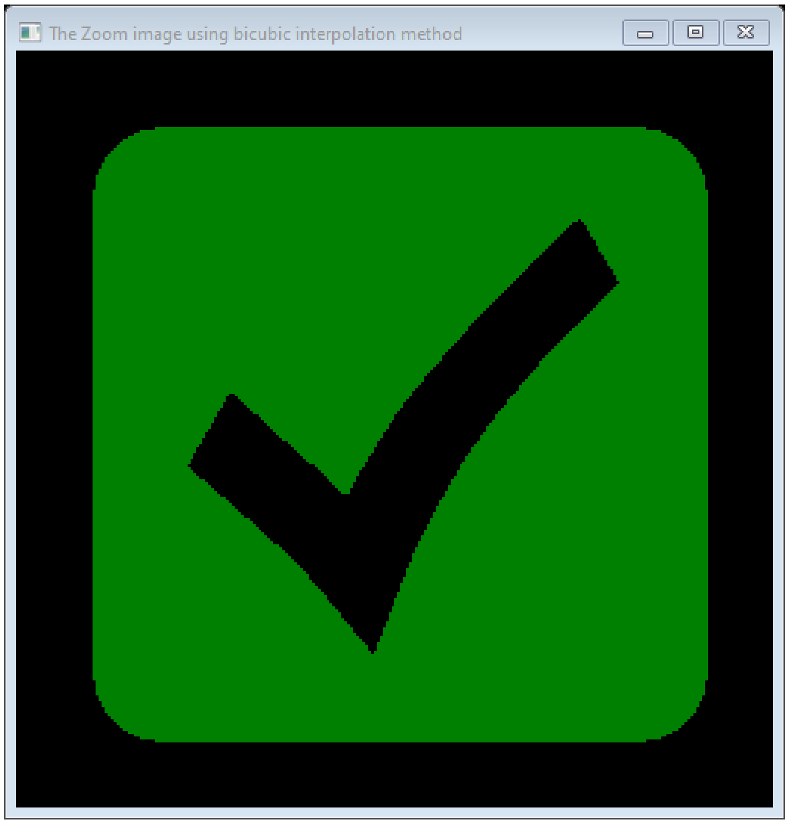

Figure 3-6 The Zoom image using the Bicubic Interpolation method

The original image has a 256X256 pixel dimension. The Zoom image which was taken by using the Bicubic Interpolation method was given the double size of the original image. Obviously, the zoom image is visualized that edges of the right shape are smoother or sharper than nearest neighbour method zoomed image but it less than Bilinear Interpolation method zoomed image because it is less visualized the pixel. Therefore, this analysis concludes that the Bicubic Interpolation method for scaling images is given the little soft edges and not given the proper level lines or hard edges because corresponding coordinates are not taken as rounded or integers, thus floating coordinates of four pixels closest to the corresponding coordinate are considered pixel value change rate between adjacent points. And the two third-order degree polynomial functions with three-time operation could be given a enlarge effect smoothly that is the reason for distributing the colour pixel continuous way. Furthermore, considering two third-order degree polynomial interpolation basis functions were given precious output of enlarge image.

Finally, comparing above mentioned three image scaling interpolation methods, the bilinear interpolation method is the best method for enlarging an image but sometimes soft edges could be jagged in this method and somehow it can be succeeded.

3.4 Exercise 4: Otsu's method for Image Thresholding

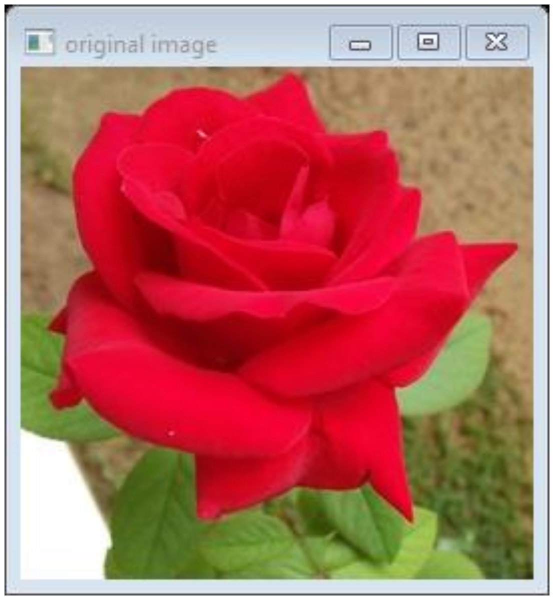

Figure 3-7 The original flower image

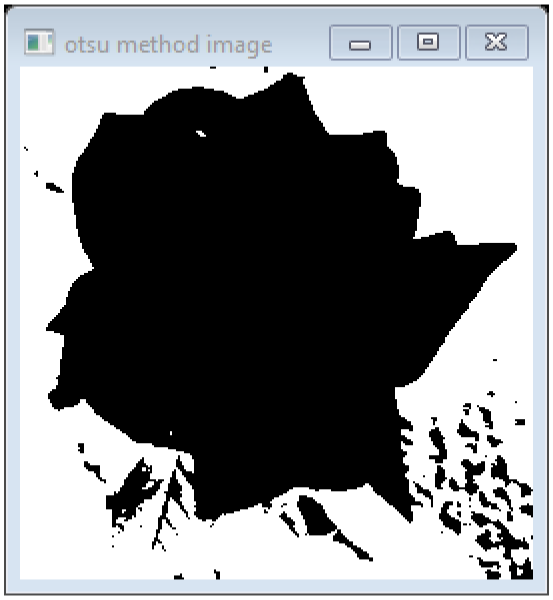

Figure 3-8 threshold flower image that was taken by using OTSU method

The variance maximum threshold: 84

The original image has a 256X256 pixel dimension. The binary image which was taken by using the OTSU threshold method was given the same size as the original image. Obviously, the resultant image has only visualized the threshold of two parts of binary colours such as black which has 255-pixel intensity value and white which has pixel intensity 0 value. When analyzing the resultant image which has the variance maximum threshold 84 and using that value it is more clearly bounded (threshold) of the background and foreground. Therefore, the red colour is contained the black colour and the green and light ranged colour represents white. This resultant image analysis is concluded that the grey value of the largest variance (maximum threshold) is 84 which is almost closed to the white pixel value intensity range and it is not closed to the dark colour pixel intensity range hence threshold flower image has less whit colour and more black colour.

4 REFERENCES

[1] *Khalaf, O., Romero, C., Azhagu Jaisudhan Pazhani, A. and Vinuja, G., 2021. VLSI Implementation of a High-Performance Nonlinear Image Scaling Algorithm. Journal of Healthcare Engineering, 2021, pp.1-10.*

[2] *Programmersought.com. 2021. Nearest neighbor (nearest neighbor)-----python - Programmer Sought. [online] Available at: <https://www.programmersought.com/article/75886157719/ > [Accessed 17 October 2021].*

[3] *Programmersought.com. 2021. Bilinear interpolation algorithm and python implementation - Programmer Sought. [online] Available at: <https://www.programmersought.com/article/42893824093/ > [Accessed 17 October 2021].*

[4] *Programmersought.com. 2021. Bilinear interpolation in image processing - Programmer Sought. [online] Available at: <https://www.programmersought.com/article/47011291323/ > [Accessed 17 October 2021].*

[5] *Angel, A., 2021. Nearest Neighbor Interpolation. [online] Imageeprocessing.com. Available at: < https://www.imageeprocessing.com/2017/11/nearest-neighbor-interpolation.html> [Accessed 17 October 2021].*

[6] *Programmersought.com. 2021. How to understand Otsu's binarization algorithm (Otsu algorithm) Python programming practice in image processing - Programmer Sought. [online] Available at: < https://www.programmersought.com/article/46874062664/> [Accessed 18 October 2021].*

[7] *Rosebrock, A., 2021. OpenCV Thresholding (cv2.threshold) - PyImageSearch. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold/ > [Accessed 18 October 2021].*

[8] *Medium. 2021. Bicubic Interpolation Techniques For Digital Imaging. [online] Available at: <https://medium.com/hd-pro/bicubic-interpolation-techniques-for-digital-imaging-7c6d86dc35dc > [Accessed 18 October 2021].*