Digital Image Processing Project.

Presented to: Dr. Marwa N. Refaie

Prepared by: Ahmed Abdel Moneim Abdel Halim.

Group: G1.

ID: 91271.

Introduction To Digital Image Processing:

The digital image processing deals with developing a digital system that performs operations on a digital image. An image is nothing more than a two dimensional signal. It is defined by the mathematical function f(x, y) where x and y are the two coordinates horizontally and vertically and the amplitude of f at any pair of coordinates (x, y) is called the intensity or gray level of the image at the point. When x, y and the amplitude values of f are all finite discrete quantities, we call the image a digital image.

The field of image digital image processing refers to the processing of digital image by means of a digital computer. A digital image is composed of finite number of elements , each of which has a particular location and values of these elements are referred to as picture elements, image elements pixels.

The RGB color system, a color image consists of three (red, green and blue) individual component images.

The Demand for a wide range of applications in environmental, agricultural, military, industry and medical science has increased.

Applications of Digital Image Processing :

1)Image sharpening and restoration.

2)Medical field.

3)Remote Sensing.

4)Transmission and encoding.

5)Machine/Robot vision.

6)Color processing.

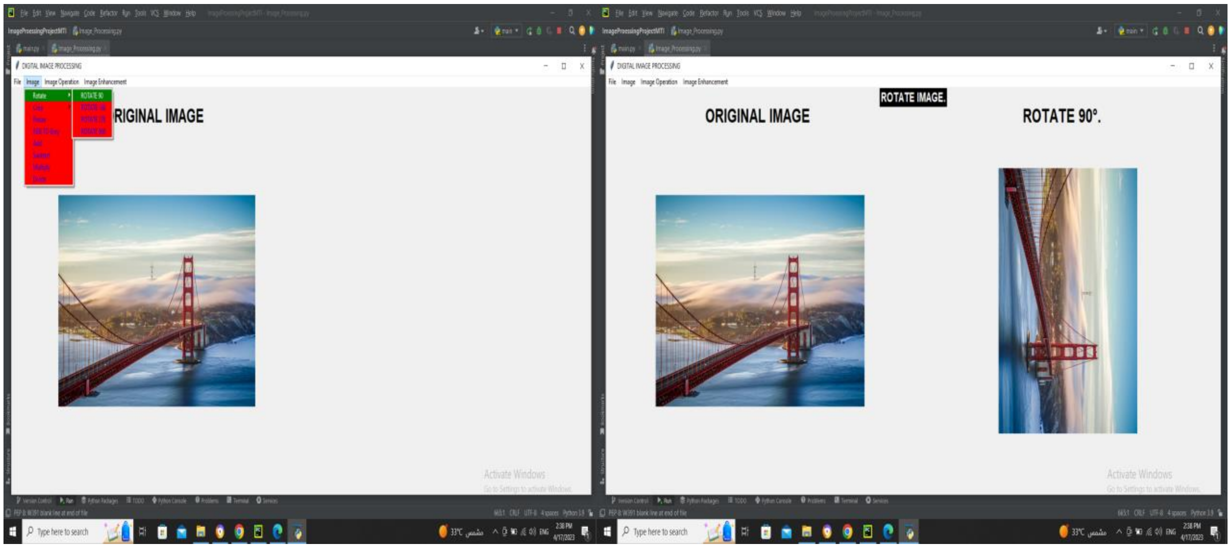
7)Pattern recognition.

Digital Image Processing Techniques:

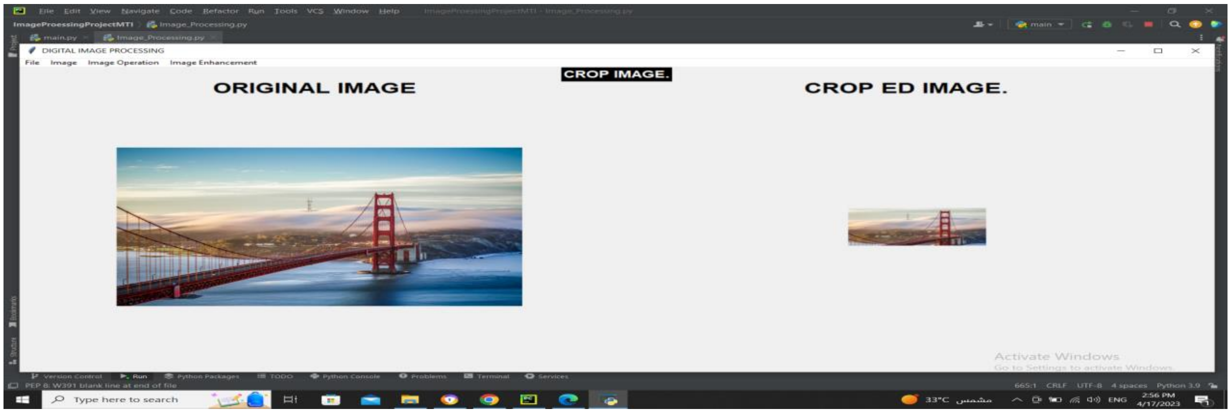
Image:

Rotate:

Rotation is a common operation in digital image processing that involves rotating an image by a certain angle around a specified point. The rotation angle can be positive or negative, and the rotation can be clockwise or counterclockwise.

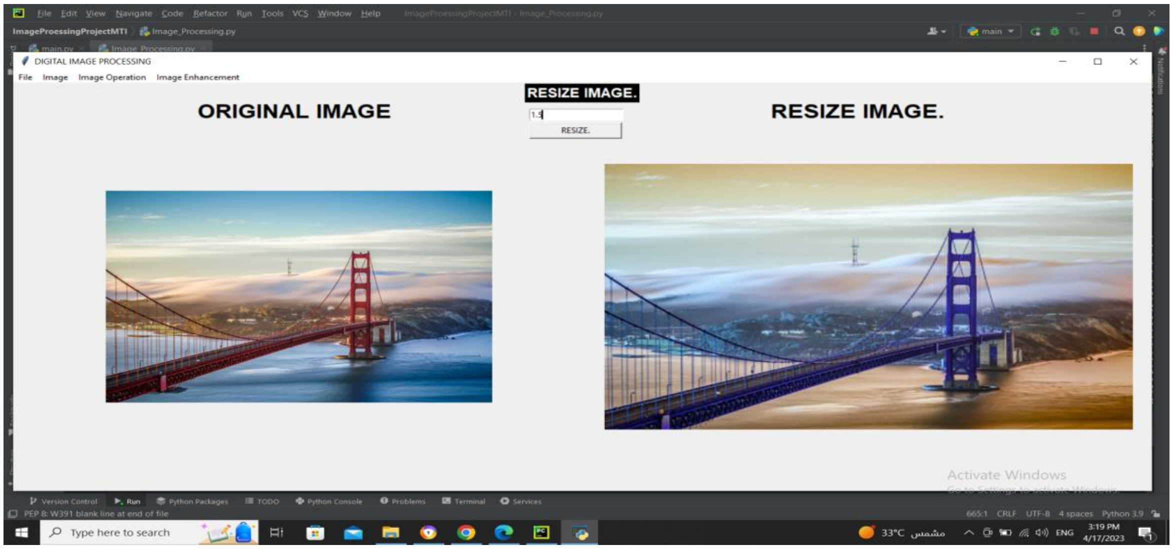


Crop: Cropping refers to the process of removing or trimming out a portion of an image to create a new, smaller image. Removing unwanted parts.



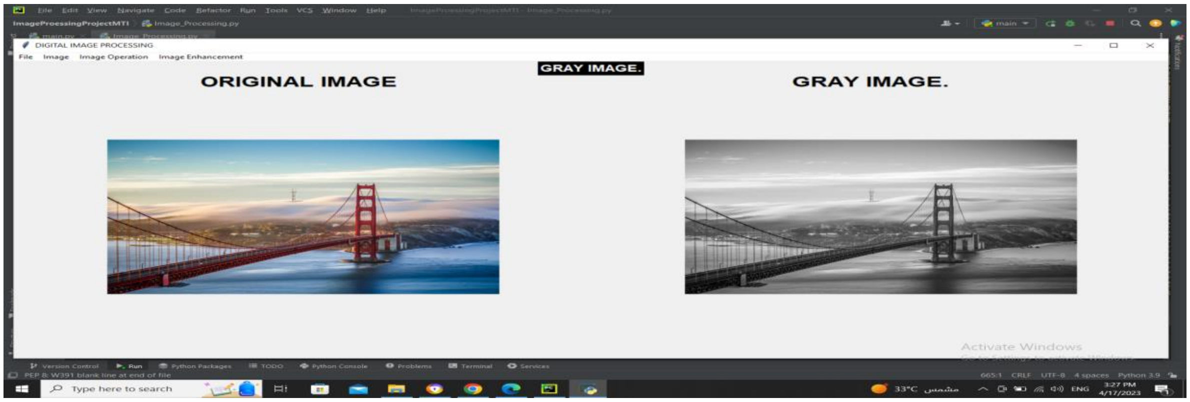
Resize:

Resizing refers to the process of changing the size of an image. This can involve making the image Larger or Smaller, while maintaining its aspect ratio.

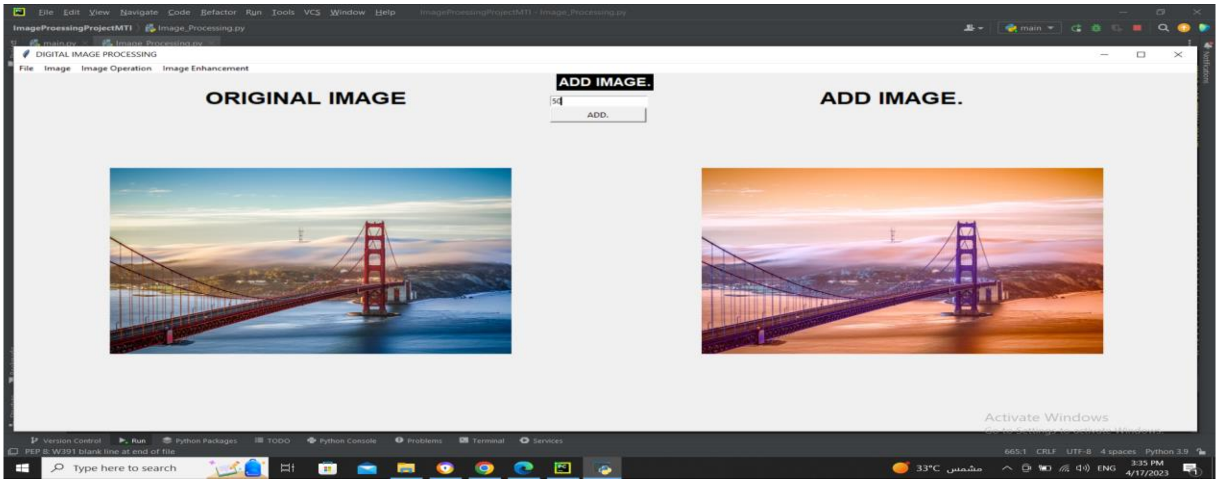

File Edit View Navigate Code Refactor Run Tools VCS Window Help
ImageProessingProjectMTI
Image_Processing.py
main.py
DIGITAL IMAGE PROCESSING
File Image Image Operation Image Enhancement
RESIZE IMAGE.
ORIGINAL IMAGE
1.5
RESIZE.
RESIZE IMAGE.
Activate Windows
Version Control Run Python Packages TODO Python Console Problems Terminal Services
PEP 8: W391 blank line at end of file
665:1 CRLF UTF-8 4 spaces Python 3.9
Type here to search
33°C مشمس
ENG
3:19 PM
4/17/2023

RGB to Grey:

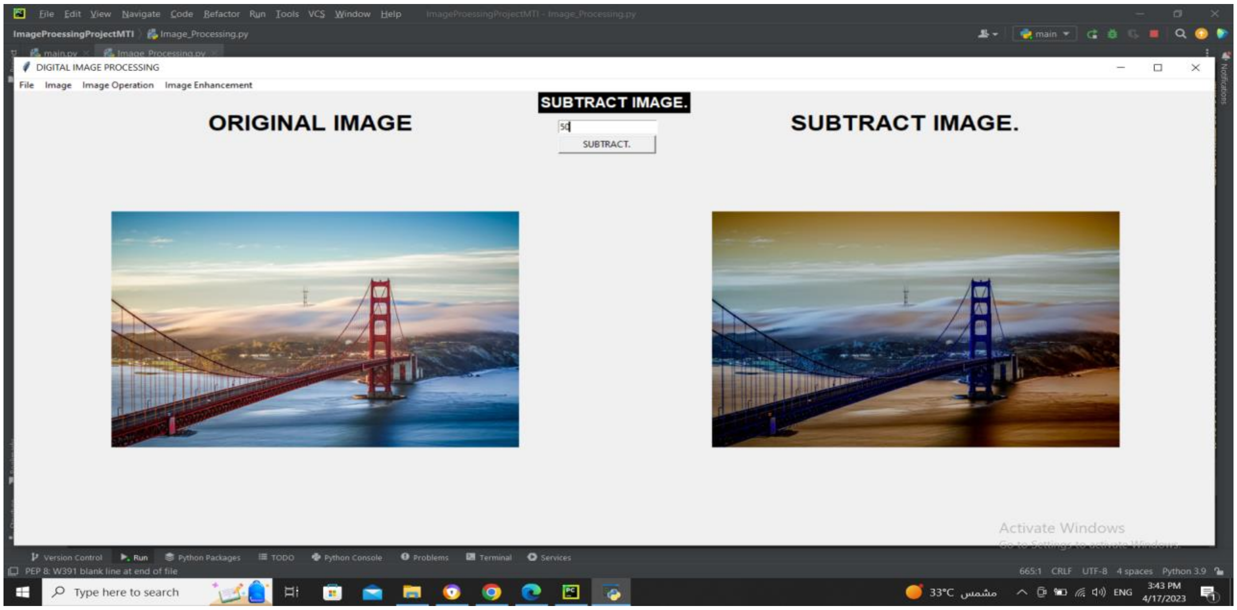
RGB stands for red, green, and blue, which are the primary colors used to create a wide range of colors in digital images. In DIP, converting an RGB Image to grayscale image is a common task that involves reducing the color information in the image from three color channels to just one.


File Edit View Navigate Code Refactor Run Tools VCS Window Help
ImageProessingProjectMTI
Image_Processing.py
DIGITAL IMAGE PROCESSING
File Image Image Operation Image Enhancement
GRAY IMAGE.
ORIGINAL IMAGE
GRAY IMAGE.
Activate Windows
Version Control Run Python Packages TODO Python Console Problems Terminal Services
PEP 8: W391 blank line at end of file
665:1 CRLF UTF-8 4 spaces Python 3.9
Type here to search
33°C مشمس
ENG
3:27 PM
4/17/2023

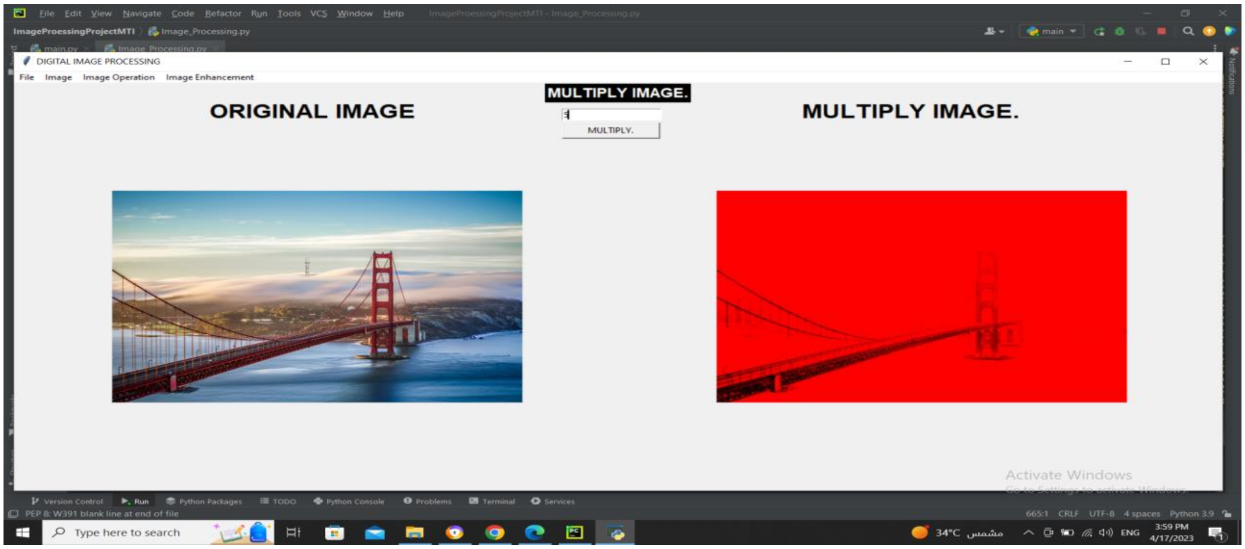
Addition: Addition in digital image processing is a basic arithmetic operation that involves adding the pixel values of two or more images on a pixel-by-pixel basis or constant value.



Subtract: Subtract in digital image processing refers to the process of subtracting the pixel values of one image from another or constant value.



Multiply: multiplication is a common mathematical operation that is applied to the pixels of an image. Multiplication of an image with a constant scaler value is known as image scaling.



Divided: Divided is a common mathematical operation that is applied to the pixels of an image. Divided of an image with a constant value.

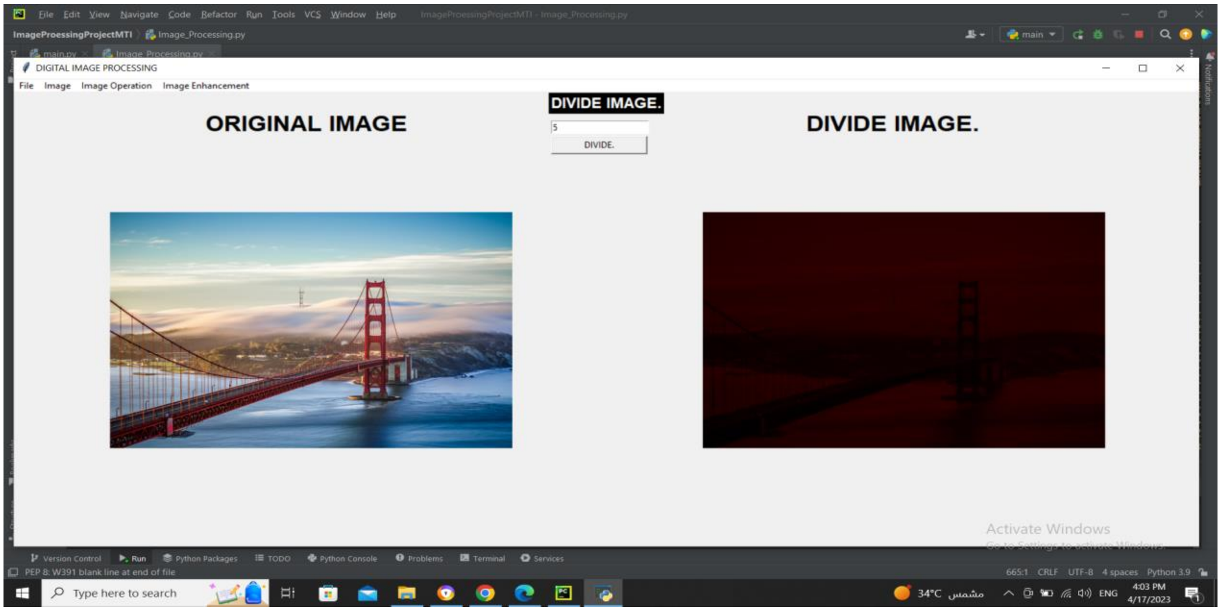
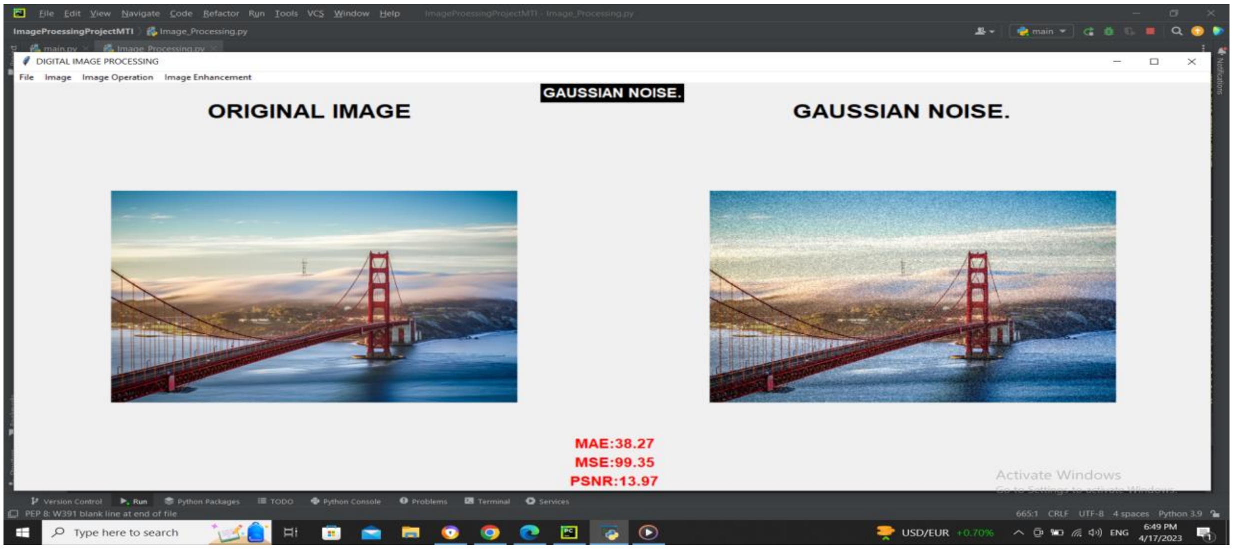


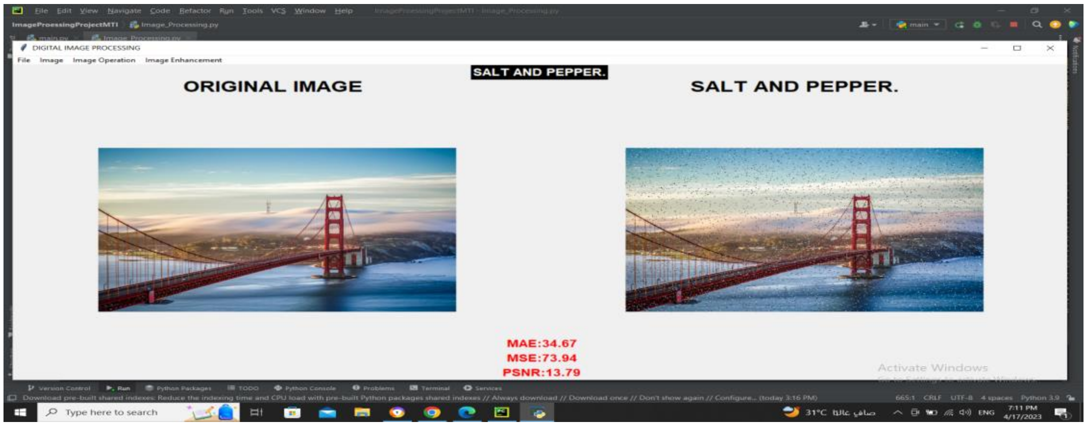
Image Operation:

Gaussian Noise: is a type of noise that is commonly found in digital images.it is a statistical noise that is caused by random variables in the intensity levels of pixels in an image.



```
MAE:38.27
MSE:99.35
PSNR:13.97
```

Salt and pepper Noise: Salt and pepper noise is a common type of noise that can appear in digital images during image acquisition, transmission or processing. It is called “salt and pepper” because it looks like white and black speckles sprinkled on an image.



```
MAE:34.67
MSE:73.94
PSNR:13.79
```

Median Blur: Median Blur is a type of image filtering operation commonly used in digital image processing. It is a non-linear filter that is applied to an image to remove noise and other small details while preserving the edges and overall structure of the image.

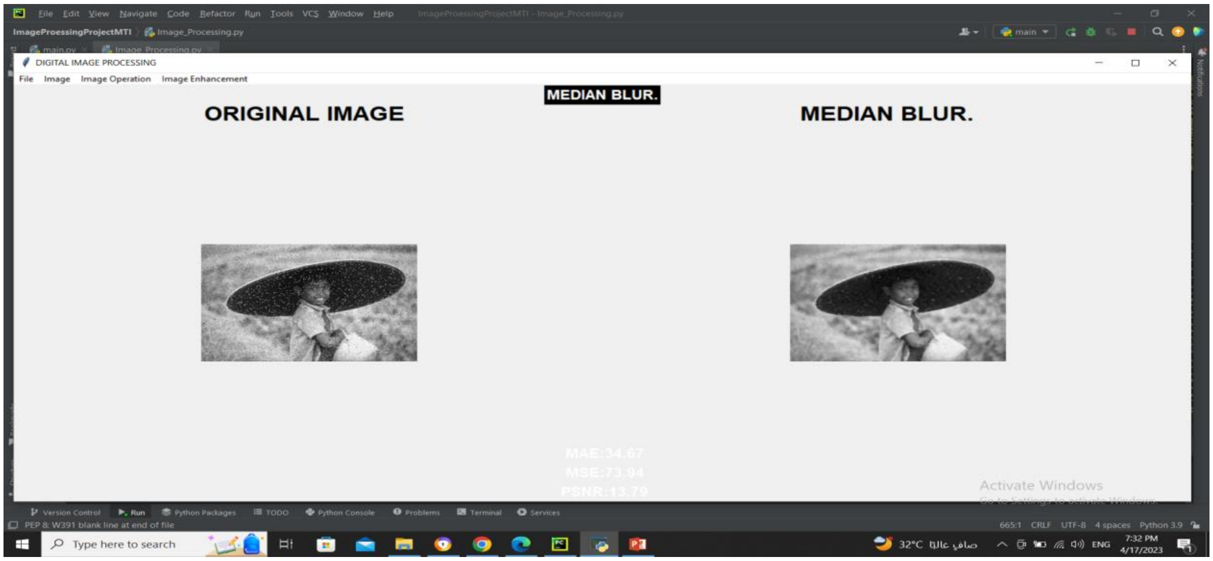
The median blur operation replaces each pixel value in the image with the median value of the pixel values within a specified neighborhood around it.

{0, 0, 0, 0, 0, 0, 5, 6, 6} = 0.

223	186	114
204	161	106
219	194	138

106 114 138 161 $\boxed{186}$ 194 204 219 223

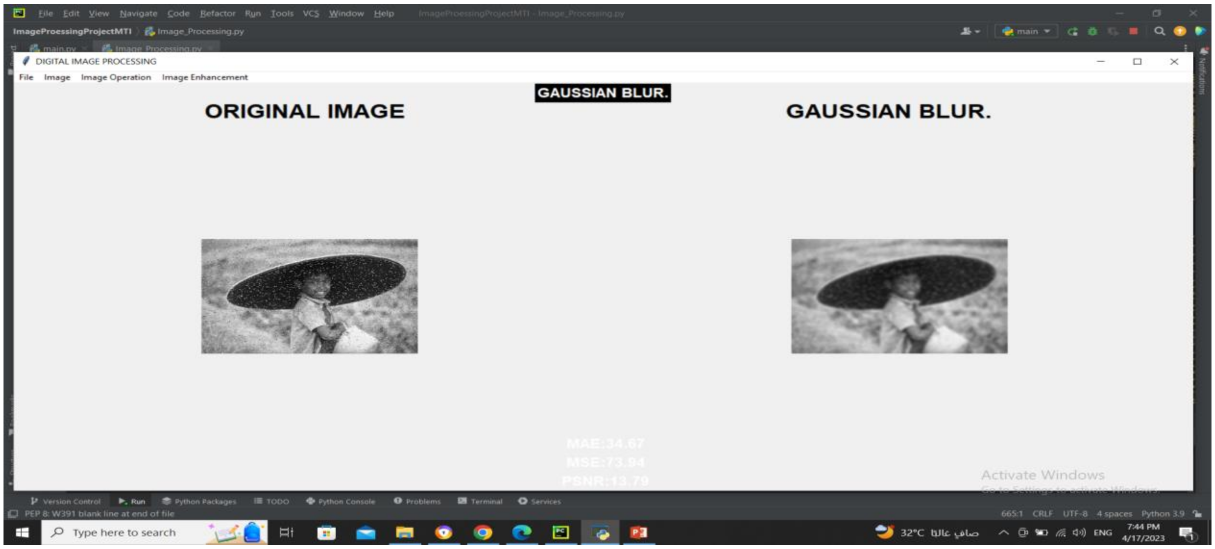
{106, 114, 138, 161, 186, 194, 204, 219, 223} = 186.



MAE:34.67
MSE:73.94
PSNR:13.79

Advantages: Very effective in removing salt and pepper and impulse noise.

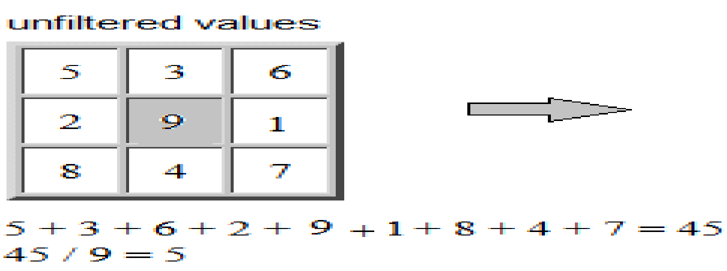
Gaussian Blur: Gaussian blur is a widely used image processing technique in which an image is convolved with a Gaussian filter noise and smooth out the image. It is often applied as a preprocessing step to reduce noise and enhance the quality of an image.



MAE:34.67
MSE:73.94
PSNR:13.79

Advantages: Very effective in <u>Gaussian Noise.</u>

Average Filter: An average filter, also known as a mean filter, is a type of digital image processing filter that is used to smooth or blur an image by reducing the amount of noise and details in the image. The average filter works by replacing each pixel value in the image with the average value of its neighboring pixels.



unfiltered values

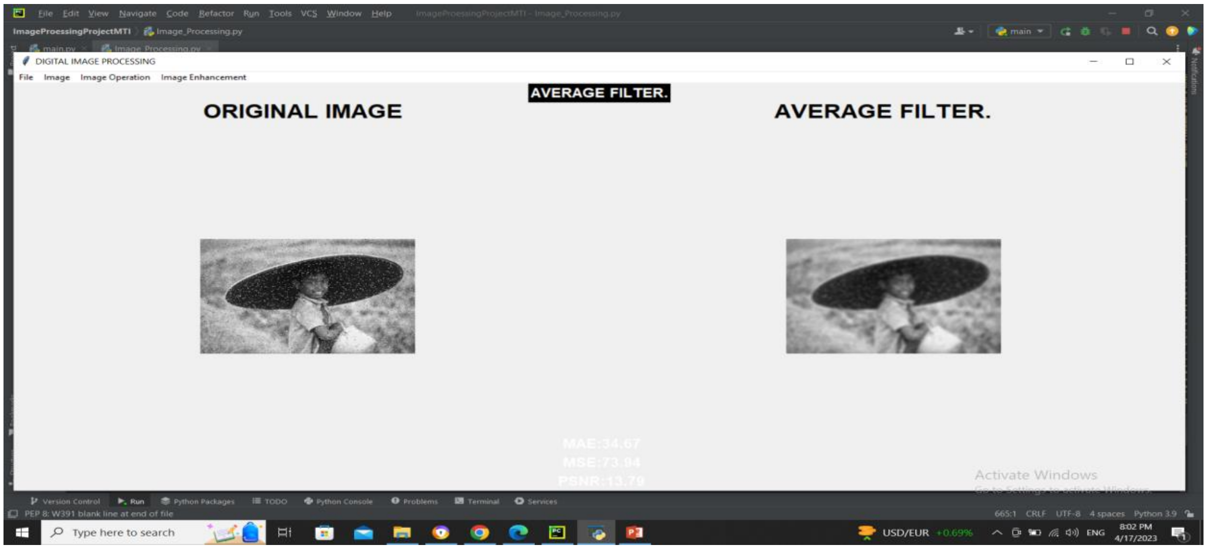
5	3	6
2	9	1
8	4	7

mean filtered

*	*	*
*	5	*
*	*	*

5 + 3 + 6 + 2 + 9 + 1 + 8 + 4 + 7 = 45
45 / 9 = 5

5 + 3 + 6 + 2 + 9 + 1 + 8 + 4 + 7 = 45. 45/9 = 5

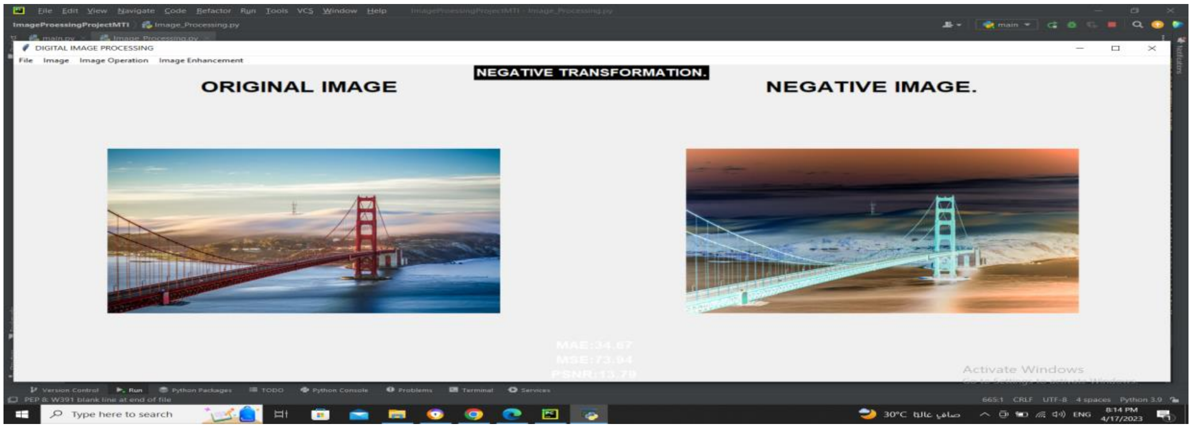


MAE:34.67
MSE:73.94
PSNR:13.79

Advantages: Reduction of irrelevant detail in an image.

Image Enhancement: image enhancement is the procedure of improving the quality and information content of original data before processing.

Negative Transformation: negative transformation refers to subtracting pixel values from (L-1-S), Where L is the maximum possible value of the pixel, and replacing it with the result.



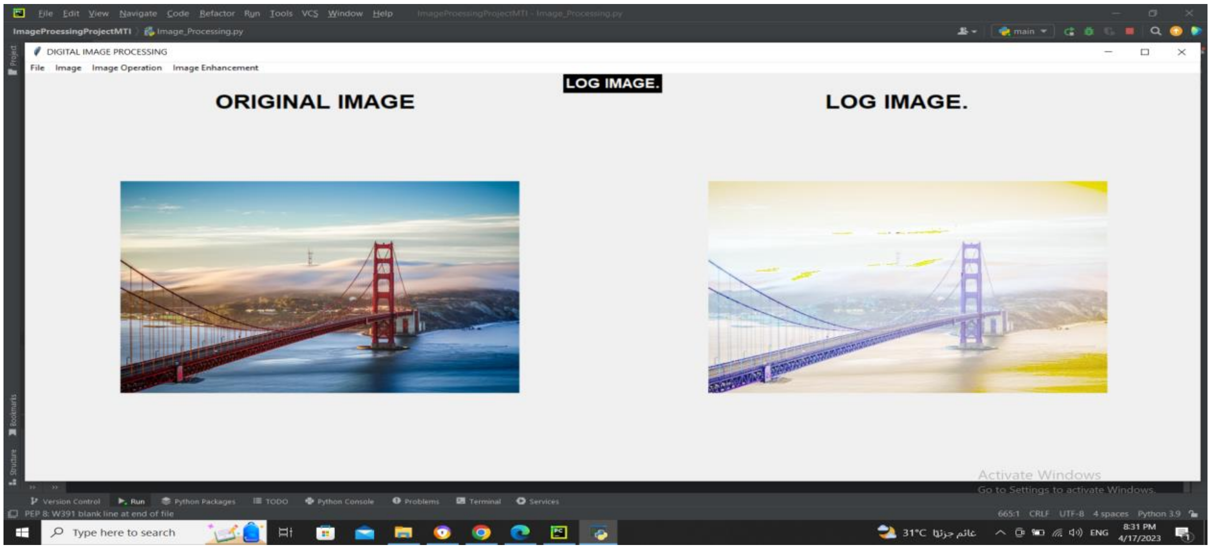
MAE:34.67
MSE:73.94
PSNR:13.79

Log Transformation: Log transformation is a mathematical operation used to enhance the contrast of an image. It involves taking the logarithm of the pixel intensities in the image.

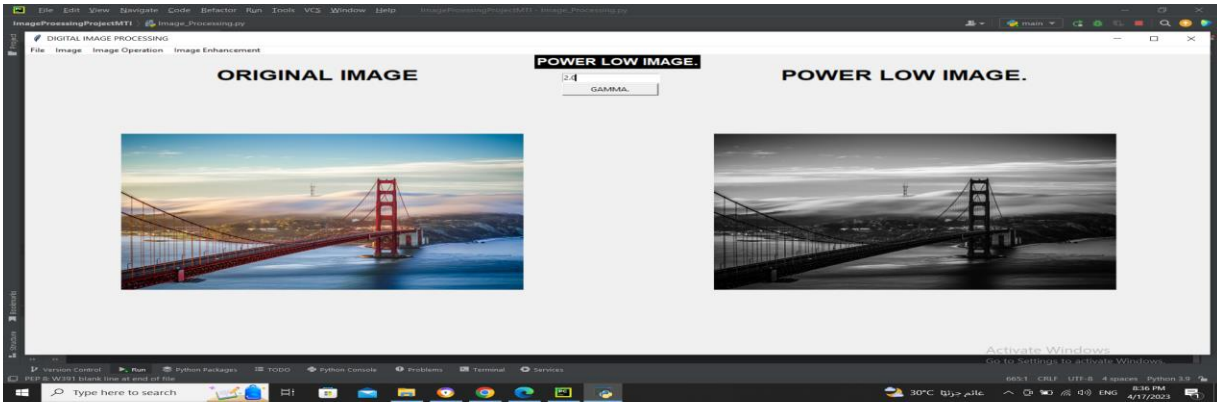
The Transformation function can be Expressed As:

s = c * log(1 + r).

Where s is the transformation pixel value, r is the original pixel value, c is a constant, and log is the natural logarithm.



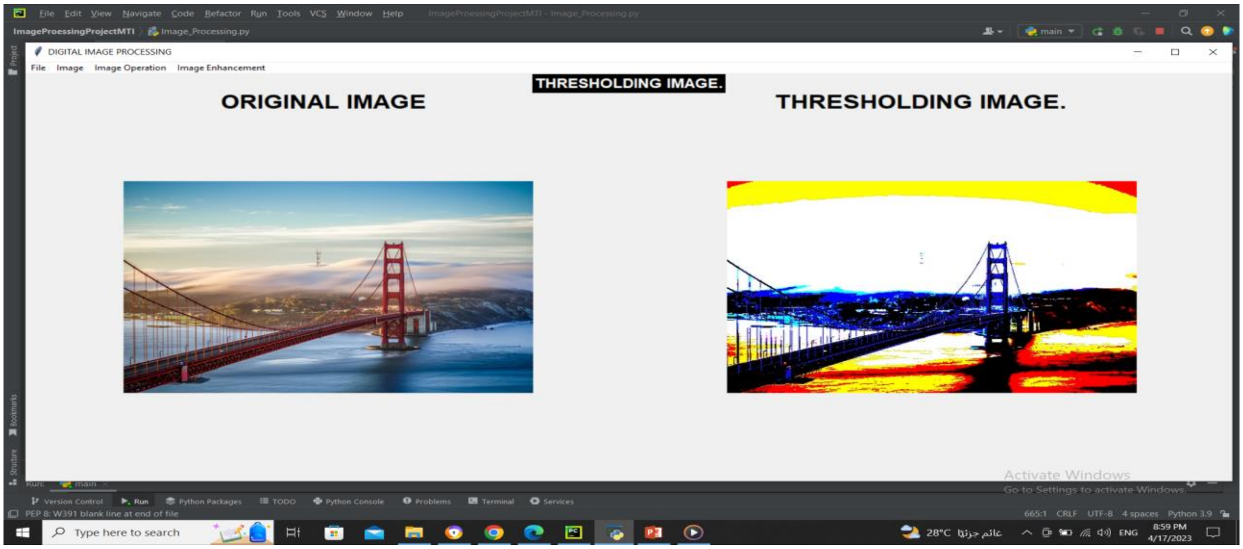
Power Law Transformation: The general form of Power Law (Gamma) Transformation function is. s = c*r^Y. Where, 's' and 'r' are the output and input pixels values, respectively and 'c' and Y are the positive constants.



Thresholding Transformation: Thresholding is a common technique used in image processing to separate objects or regions in an image based on their intensity levels. A thresholding value is chosen, and pixels in the image with instensity values above.

S = {c if r >= m

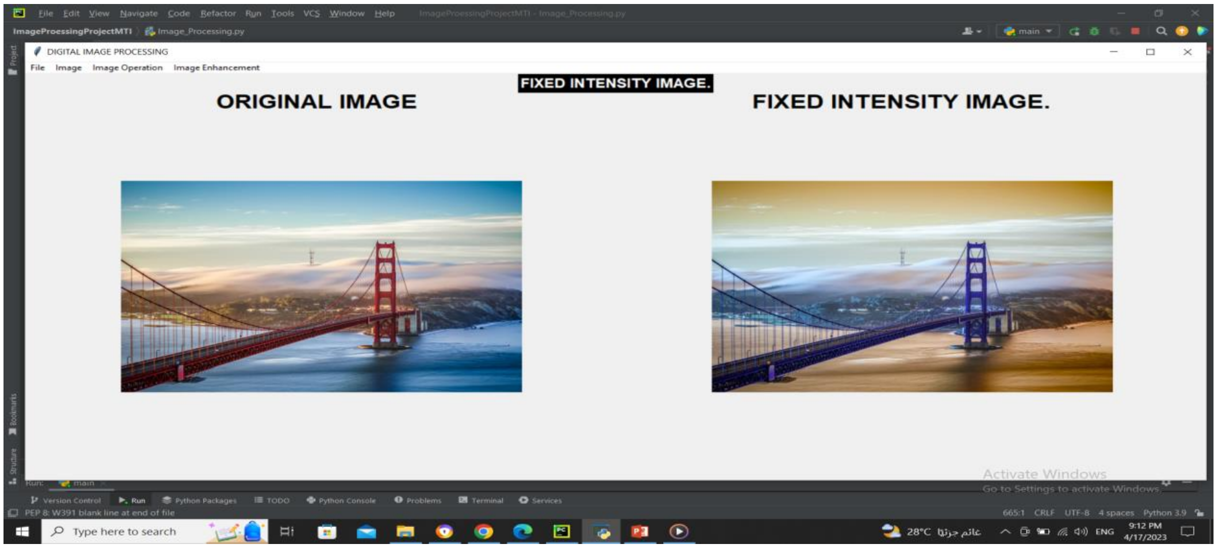
{0 if r < m


File Edit View Navigate Code Refactor Run Tools VCS Window Help
ImageProessingProjectMTI - Image_Processing.py
ImageProessingProjectMTI
Image_Processing.py
DIGITAL IMAGE PROCESSING
File Image Image Operation Image Enhancement
THRESHOLDING IMAGE.
ORIGINAL IMAGE
THRESHOLDING IMAGE.
Activate Windows
Go to Settings to activate Windows.
Run: main
Version Control Run Python Packages TODO Python Console Problems Terminal Services
665:1 CRLF UTF-8 4 spaces Python 3.9
Type here to search
28°C غائم جزئيا
ENG
8:59 PM
4/17/2023

PEP 8: W391 blank line at end of file

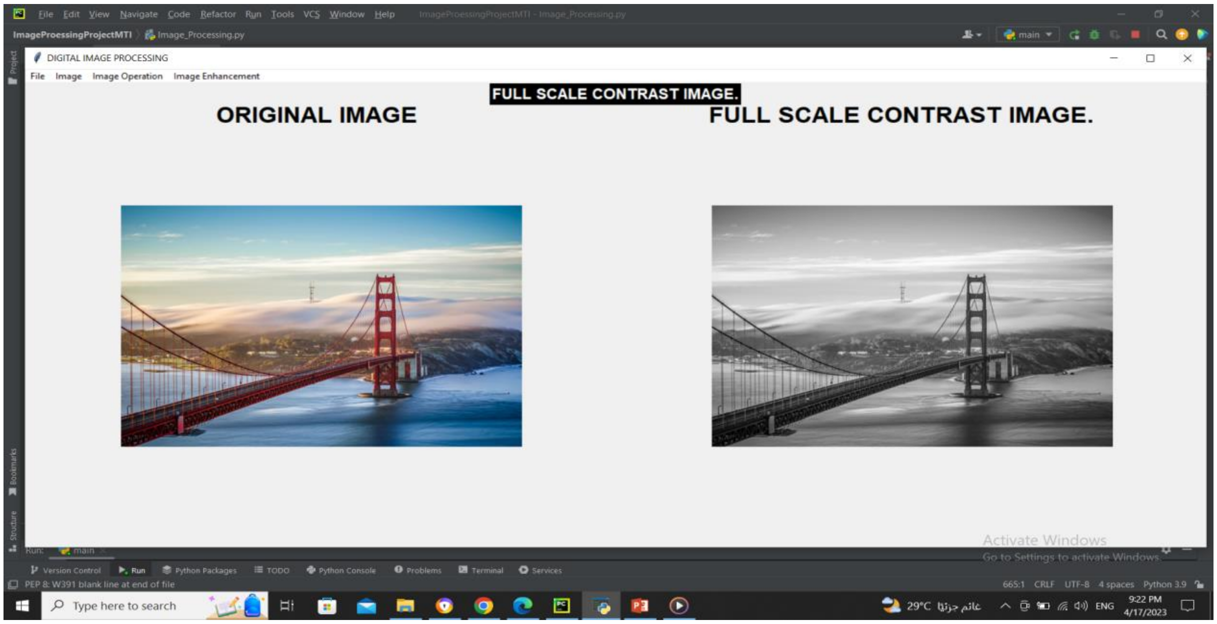
Fixed Intensity Transformation: Fixed intensity transformations in image processing are a class of image processing techniques that modify the intensity values of pixels in an image using a fixed transformation function.

Equation: $s\ =\ round\left(\frac{1}{L-1}\,r^2\right)$



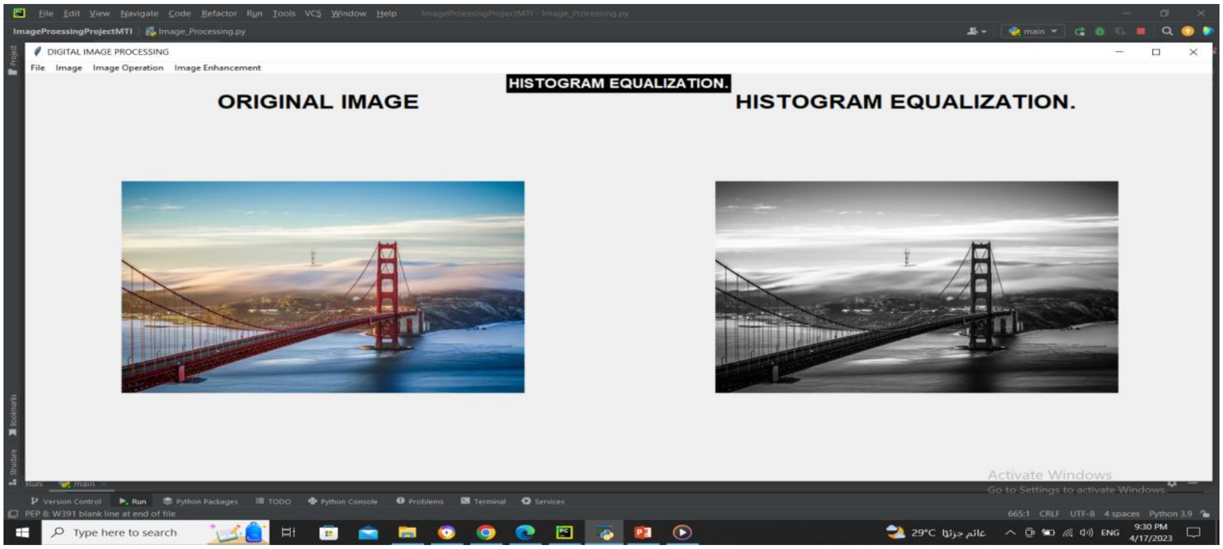
Full Scale Contrast Transformation: is the type of image processing technique that the contrast of an image by stretching the range of intensity values to cover the entire available range. This technique is often used to enhance the visual appearance of an image or to make certain details more visible.

Equation: $S = round\left((2^B - 1) \cdot \frac{r - rmin}{rmax - rmin}\right)$.


DIGITAL IMAGE PROCESSING
File Image Image Operation Image Enhancement
FULL SCALE CONTRAST IMAGE.
ORIGINAL IMAGE
FULL SCALE CONTRAST IMAGE.

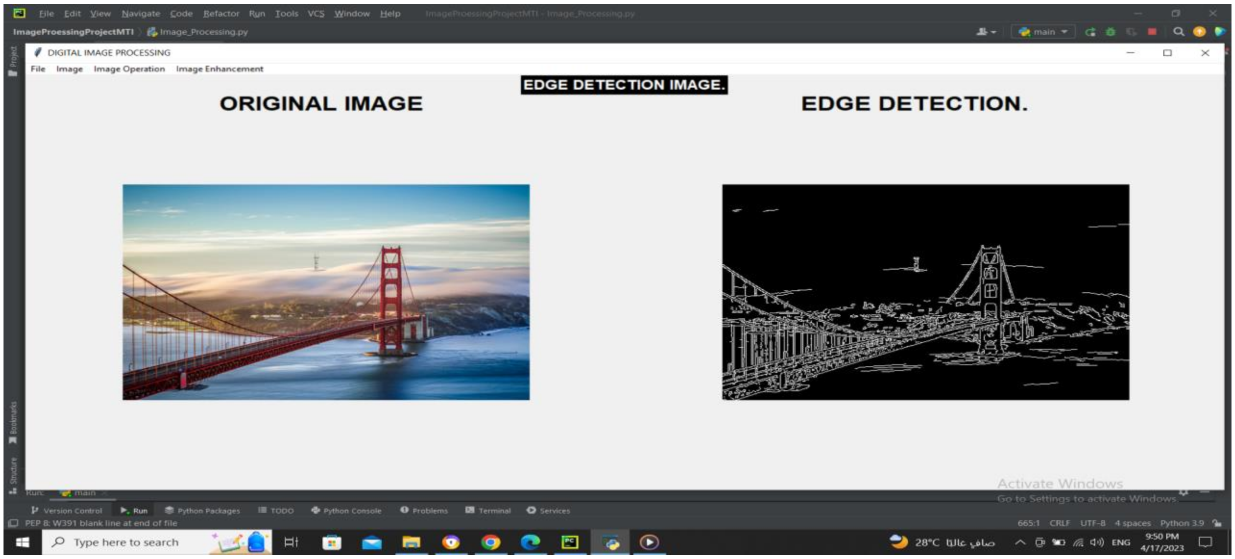
Histogram Equalization:

Histogram Equalization is a technique used to in image processing to improve the contrast of an image. The goal of histogram equalization is to transform the pixel values of an image so that the histogram of the output image is as close as possible to a uniform distribution.is a computer image processing technique used to improve contrast in images.



Edge Detection:

Edge Detection is a technique used in image processing to identify and locate the boundaries of objects within an image and Identify edges.



The Code:

Main:

```
#Import tkinter library.
from tkinter import *
from PIL import Image
from PIL import ImageTk
from Image_Processing import Application
def click_Button():
    window.destroy()
    #Go-To-Application.
    ui = Application()
    ui.mainloop()
####################################################
window = Tk()
window.title('DIGITAL IMAGE PROCESSING APPLICATION')
window.eval('tk::PlaceWindow . center')
width = 1250  # Width.
height = 650  # Height.
screen_width = window.winfo_screenwidth()  # Width of the screen.
screen_height = window.winfo_screenheight()  # Height of the screen.
# Calculate Starting X and Y coordinates for Window.
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
window.geometry('%dx%d+%d+%d' % (width, height, x, y))
# the Maximum size-Don't-Change.
# tkinter fixed window size.
window.maxsize(width=width,height=height)
window.config(bg='#013A63',highlightthickness=0)
print(width)
print(height)
#Create Image Background.
#image = Image.open('BG-scaled.jpg')
image = Image.open('SanFranciscoAhmed.jpg')
image = image.resize((1350, 800))
image = ImageTk.PhotoImage(image)
canvas = Canvas(width=width,height=height,highlightthickness=0)
#images = PhotoImage(file = image)
canvas.create_image(width/2,height/2,image=image)
canvas.pack()
#Label.
l1 = Label(canvas, text='DIGITAL IMAGE PROCESSING APPLICATION',
           font=('Arial',44, 'bold'),
           pady=25, justify=CENTER,bg='#ff4d4d',fg='#3333ff',highlightthickness=0)
l1.place(relx=0.5, rely=0.35, anchor='center')
#Button.
b1 = Button(canvas, text='CONTINUE', bg='#ff4d4d', fg='#3333ff',
              font=('Arial',30,'bold'),activebackground='#012A4A',
            highlightthickness=0,command=click_Button)
b1.place(relx=0.5, rely=0.75, anchor='center')
```

```
window.mainloop()
##############################################
##############################################
```

Image Processing:

```
##Library########################
from tkinter import *
from tkinter import ttk
from tkinter.font import Font
from tkinter import filedialog
from tkinter import messagebox
from PIL import Image, ImageTk
import numpy as np
import cv2 as cv
import cv2
import random
from PIL import Image
import numpy as np
from imgaug import augmenters as iaa
###############################
#########Rotate################
def rotate_the_Image_90(image):
    img = cv.cvtColor(image,cv.COLOR_BGR2RGB)
    New_image = cv.rotate(img, cv.ROTATE_90_CLOCKWISE)
    return New_image
def rotate_the_Image_180(image):
    img = cv.cvtColor(image,cv.COLOR_BGR2RGB)
    New_image = cv.rotate(img, cv.ROTATE_180)
    return New_image
def rotate_the_Image_270(image):
    img = cv.cvtColor(image,cv.COLOR_BGR2RGB)
    New_image = cv.rotate(img, cv.ROTATE_90_COUNTERCLOCKWISE)
    return New_image
def CROP_IMAGE(image):
    def resize_image(img):
        scale_percent = 50  # percent of original size.
        width = int(img.shape[1] * scale_percent / 100)
        height = int(img.shape[0] * scale_percent / 100)
        dim = (width, height)
        # resize image.
        resized = cv.resize(img, dim, interpolation=cv.INTER_AREA)
        return resized
    img = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    cropping = False
    x_start, y_start, x_end, y_end = 0, 0, 0, 0
    image = resize_image(img)
    oriImage = image.copy()
    def mouse_crop(event, x, y, flags, param):
        # grab references to the global variables.
        global x_start, y_start, x_end, y_end, cropping
        # if the left mouse button was DOWN, start RECORDING.
        # (x, y) coordinates and indicate that cropping is being.
        if event == cv.EVENT_LBUTTONDOWN:
            x_start, y_start, x_end, y_end = x, y, x, y
            cropping = True
        # Mouse is Moving.
        elif event == cv.EVENT_MOUSEMOVE:
            if cropping == True:
                x_end, y_end = x, y
        # if the left mouse button was released.
        elif event == cv.EVENT_LBUTTONUP:
            # record the ending (x, y) coordinates.
            x_end, y_end = x, y
            cropping = False  # cropping is finished.
            refPoint = [(x_start, y_start), (x_end, y_end)]
            if len(refPoint) == 2:  # when two points were found.
                roi = oriImage[refPoint[0][1]:refPoint[1][1],
refPoint[0][0]:refPoint[1][0]]
                cv.imshow("Cropped", roi)
                cv.imwrite("CroppedImage.jpg",roi)
    cv.namedWindow("image")
    cv.setMouseCallback("image", mouse_crop)
    while True:
        i = image.copy()
        if not cropping:
            cv.imshow("image", image)
            break
        elif cropping:
            cv.rectangle(i, (x_start, y_start), (x_end, y_end), (255, 0, 0), 2)
            cv.imshow("image", i)
        cv.waitKey(1)
#######RisizeImage################
def risize_image(image,number):
    import cv2
    img = image
    scale_percent = 100*number
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
    cv2.imwrite("resized_Image.jpg", resized)
#$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$#
def addValueImage(image,value):
    add = cv2.add(image,value)
    cv2.imwrite("NewAddImage.jpg", add)
def subtractValueImage(image,value):
    subtract = image - value
    cv2.imwrite("NewSubtractImage.jpg", subtract)
def multiplyValueImage(image,value):
    multiply = cv2.multiply(image, value)
    cv2.imwrite("NewMultiplyImage.jpg", multiply)
def divideValueImage(image, value):
    divide = cv2.divide(image, value)
    cv2.imwrite("NewDivideImage.jpg", divide)
def rgb2gray(img):
    image = img
    grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    cv2.imwrite("rgb_to_gray.jpg",grayscale)
def Mean_Absolute_error(path1,path2):
    img1 = cv2.imread(path1)
    img2 = cv2.imread(path2)
    abs_diff = cv2.absdiff(img1, img2)
    mean_abs_error = np.mean(abs_diff)
    return mean_abs_error
def Mean_square_error(path1,path2):
    img1 = cv2.imread(path1)
    img2 = cv2.imread(path2)
    sq_diff = np.square(cv2.absdiff(img1, img2))
    mean_sq_error = np.mean(sq_diff)
    return mean_sq_error
def Peak_signal_to_noise_ratio(path1,path2):
    img_original = cv2.imread(path1)
    img_degraded = cv2.imread(path2)
    psnr = cv2.PSNR(img_original, img_degraded)
    return psnr
######Negative Transformation##
def imgNe(img):
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    # Invert the image using cv2.bitwise_not.
    img_neg = cv.bitwise_not(img)
    return img_neg
#Negative.
def Negative_of_Image(image):
    img = image
    img_inv = 255 - img
    cv2.imwrite("Negative_of_Image.jpg",img_inv)
#Log transformation.
def Log_transformation(img):
    image = img
    # Apply log transformation method.
    c = 255 / np.log(1 + np.max(image))
    log_image = c * (np.log(image + 1))
    # float value will be converted to int.
    log_image = np.array(log_image, dtype=np.uint8)
    cv2.imwrite("Log_of_Image.jpg",log_image)
####Power Law (Gamma).####
def Power_Law_transformation(image,Gamma):
    img = image
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gamma = Gamma
    output = 255 * ((gray_img / 255) ** gamma)
    cv2.imwrite("Power_Law_Image.jpg",output)
####Thresholded.##########
def Thresholded_transformation(image):
    img = image
    ret, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
    cv2.imwrite("Thresholded_Image.jpg",thresh)
####Fixed intensity transformation.###
def Fixed_intensity_transformation(image):
    img = image
    alpha = 2.0
    beta = 50
    new_img = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)
    cv2.imwrite("Fixed_intensity.jpg",img)
def Full_Scale_Contrast_Stretch(path):
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    min_val, max_val, _, _ = cv2.minMaxLoc(img)
    out = np.zeros_like(img)
    out = ((img - min_val) / (max_val - min_val)) * 255
    out = np.uint8(out)
    cv2.imwrite('Full_Scale_Contrast.jpg', out)
#Average Filter.
def Average_Filter(image):
    img = image
    # Apply average filter with a kernel size of 5x5.
    kernel_size = (5, 5)
    blur = cv2.blur(img, kernel_size)
    cv2.imwrite('AverageFilterImage.jpg', blur)
#Histogram Equalization of image.
def Histogram_Equalization(path):
    img = cv2.imread(path, 0)
    # Perform histogram Equalization.
    equ = cv2.equalizeHist(img)
    cv2.imwrite('histogramEqualizationImage.jpg',equ)
def Edge_Detection(path):
    img = cv2.imread(path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 100, 200)
    cv2.imwrite('Edge_DetectionImage.jpg',edges)
##Aoolication#########################################
class Application(Tk):
    img = None
    img_is_found = False
    ifile = ''
    def window_size(self):
        width = 1500  # Width.
        height = 650  # Height.
        screen_width = self.winfo_screenwidth()  # Width of the screen.
        screen_height = self.winfo_screenheight()  # Height of the screen.
        # Calculate Starting X and Y coordinates for Window.
        x = (screen_width / 2) - (width / 2)
        y = (screen_height / 2) - (height / 2)
        self.geometry('%dx%d+%d+%d' % (width, height, x, y))
    def createWidgets(self):
        self.menuBar = Menu(master=self,background='#ffd700', fg='white')
        self.filemenu = Menu(self.menuBar,
tearoff=0,bg="red",fg="blue",activebackground="green")
        self.filemenu.add_command(label="Open", command=self.choose)
        self.filemenu.add_command(label="Save", command=self.save_image)
        self.filemenu.add_command(label="Exit", command=self.destroy)
        self.menuBar.add_cascade(label="File", menu=self.filemenu)
        self.Image_menu = Menu(self.menuBar,
tearoff=0,bg="red",fg="blue",activebackground="green")
        self.rotate_Img = Menu(self.Image_menu,
tearoff=0,bg="red",fg="blue",activebackground="green")
```

```
        self.crop_Img = Menu(self.Image_menu,
tearoff=0,bg="red",fg="blue",activebackground="green")
        self.rotate_Img.add_command(label="ROTATE 90",command=self.ROTATE_90)
        self.rotate_Img.add_command(label="ROTATE 180",command=self.ROTATE_180)
        self.rotate_Img.add_command(label="ROTATE 270",command=self.ROTATE_270)
        self.rotate_Img.add_command(label="ROTATE 360",command=self.ROTATE_360)
        self.crop_Img.add_command(label="Crop", command=self.Crop_Image)
        self.crop_Img.add_command(label="Print", command=self.Print_Croped_Image,
state="disabled")
        self.Image_menu.add_cascade(label='Rotate' , menu = self.rotate_Img)
        self.Image_menu.add_cascade(label='Crop', menu = self.crop_Img)
        self.Image_menu.add_command(label='Resize', command = self.resize_Image)
        self.Image_menu.add_command(label="RGB TO Grey", command = self.rgb2Gray)
        self.Image_menu.add_command(label='Add',command= self.add_Image)
        self.Image_menu.add_command(label="Subtract",command = self.subtract_Image)
        self.Image_menu.add_command(label="Multiply",command = self.multiply_Image)
        self.Image_menu.add_command(label="Divide",command = self.divide_Image)
        self.menuBar.add_cascade(label="Image", menu=self.Image_menu)
        self.ImageOperation_menu = Menu(self.menuBar,
tearoff=0,bg="red",fg="blue",activebackground="green")#Create Class Object.
        self.ImageOperation_menu.add_command(label='Gaussian
Noise',command=self.Add_gaussian_Noise)
        self.ImageOperation_menu.add_command(label='Salt &
Paper',command=self.Add_saltpepperNoise)
        self.ImageOperation_menu.add_command(label='Median Blur',command=self.medianBlur)
        self.ImageOperation_menu.add_command(label='Gaussian
Blur',command=self.GaussianBlur)
        self.ImageOperation_menu.add_command(label='Average
Filter',command=self.Average_Filter)
        self.menuBar.add_cascade(label="Image Operation", menu=self.ImageOperation_menu)
        self.ImageEnhancement_menu = Menu(self.menuBar,
tearoff=0,bg="red",fg="blue",activebackground="green")
        self.ImageEnhancement_menu.add_command(label='Negative Transformation',
command=self.negtaive_Img)
        self.ImageEnhancement_menu.add_command(label='Log
Transformation',command=self.Log_Image)
        self.ImageEnhancement_menu.add_command(label='Power Low
Transformation',command=self.Power_Low_Image)
        self.ImageEnhancement_menu.add_command(label='Thresholding
Transformation',command=self.Thresholding_Image)
        self.ImageEnhancement_menu.add_command(label='Fixed Intensity
Transformation',command=self.Fixed_intensity_Image)
        self.ImageEnhancement_menu.add_command(label='Full Scale Contrast
Transformation',command=self.Full_Scale_Contrast)
        self.ImageEnhancement_menu.add_command(label='Negative',
command=self.Negative_Image)
        self.ImageEnhancement_menu.add_command(label='Histogram
Equalization',command=self.Histogram_Equalization)
        self.ImageEnhancement_menu.add_command(label='Edge
Detection',command=self.Edge_Detection)
        self.menuBar.add_cascade(label="Image Enhancement",
menu=self.ImageEnhancement_menu)
    def choose(self):
        self.ifile = filedialog.askopenfilename(parent=self, title='Choose a file')
        if self.ifile:
            self.path = Image.open(self.ifile)
            self.image2 = ImageTk.PhotoImage(self.path)
            self.label.configure(image=self.image2)
            self.label.image = self.image2
            self.img = np.array(self.path)
            self.img = self.img[:, :, ::-1].copy()
            self.img_is_found = True
            self.ifile = self.ifile
            self.save_image(self.img)
    def negtaive_Img(self):
        if self.img_is_found:
            self.labelTop.configure(text="NEGATIVE TRANSFORMATION.",bg="black")
            img_after = imgNe(self.img)
            img_after = ImageTk.PhotoImage(Image.fromarray(img_after))
            self.label2.configure(image=img_after)
            self.label2.image = img_after
            self.Restoration_Filter.configure(text='NEGATIVE IMAGE.')
            self.Restoration_Filter.text = 'NEGATIVE IMAGE.'
            self.destroyLabel()
            self.destroyButton()
    def ROTATE_90(self):
        if self.img_is_found:
            self.labelTop.configure(text="ROTATE IMAGE.",bg="black")
            img_after = self.img
            img_after = rotate_the_Image_90(self.img)
            img_after = ImageTk.PhotoImage(Image.fromarray(img_after))
            self.label2.configure(image=img_after)
            self.label2.image = img_after
            self.Restoration_Filter.configure(text='ROTATE 90°.')
            self.Restoration_Filter.text = 'ROTATE 90°.'
            self.destroyLabel()
            self.destroyButton()
    def ROTATE_180(self):
        if self.img_is_found:
            self.labelTop.configure(text="ROTATE IMAGE.",bg="black")
            img_after = self.img
            img_after = rotate_the_Image_180(self.img)
            img_after = ImageTk.PhotoImage(Image.fromarray(img_after))
            self.label2.configure(image=img_after)
            self.label2.image = img_after
            self.Restoration_Filter.configure(text='ROTATE 180°.')
            self.Restoration_Filter.text = 'ROTATE 180°.'
            self.destroyLabel()
            self.destroyButton()
    def ROTATE_270(self):
        if self.img_is_found:
            self.labelTop.configure(text="ROTATE IMAGE.",bg="black")
            img_after = self.img
            img_after = rotate_the_Image_270(self.img)
            img_after = ImageTk.PhotoImage(Image.fromarray(img_after))
            self.label2.configure(image=img_after)
            self.label2.image = img_after
            self.Restoration_Filter.configure(text='ROTATE 270°.')
            self.Restoration_Filter.text = 'ROTATE 270°.'
            self.destroyLabel()
            self.destroyButton()
    def ROTATE_360(self):
        if self.img_is_found:
            self.labelTop.configure(text="ROTATE IMAGE.",bg="black")
            img_after = self.img
            img_after = ImageTk.PhotoImage(Image.fromarray(img_after))
            self.label2.configure(image=img_after)
            self.label2.image = img_after
            self.Restoration_Filter.configure(text='ROTATE 360°.')
            self.Restoration_Filter.text = 'ROTATE 360°.'
            self.destroyLabel()
            self.destroyButton()
    def Crop_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="CROP IMAGE.",bg="black")
            img_after = self.img
            img_after = CROP_IMAGE(self.img)
            self.crop_Img.entryconfig("Print",state="normal")
            self.destroyLabel()
            self.destroyButton()
    def Print_Croped_Image(self):
        img_after = cv.imread("CroppedImage.jpg")
        img_after = ImageTk.PhotoImage(Image.fromarray(img_after))
        self.label2.configure(image=img_after)
        self.label2.image = img_after
        self.Restoration_Filter.configure(text='CROP ED IMAGE.')
        self.Restoration_Filter.text = 'CROP ED IMAGE.'
        self.destroyLabel()
        self.destroyButton()
    #################################################
    def resize_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="RESIZE IMAGE.",bg="black")
            self.Entry_resize.place(x=680,y=40)
            self.Button_resize.place(x=680,y=62)
            self.Button_resize.configure(command=self.resize_Image2,text="RESIZE.")
            self.destroyLabel()
    def resize_Image2(self):
        img_after = self.img
        img_risize = risize_image(img_after, float(self.Entry_resize.get()))
        image_resize = cv2.imread("resized_Image.jpg")
        image_resize = ImageTk.PhotoImage(Image.fromarray(image_resize))
        self.label2.configure(image=image_resize)
        self.label2.image = image_resize
        self.Restoration_Filter.configure(text='RESIZE IMAGE.')
        self.Restoration_Filter.text = 'RESIZE IMAGE.'
        self.destroyLabel()
    def add_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="ADD IMAGE.",bg="black")
            self.Entry_resize.place(x=680,y=40)
            self.Button_resize.place(x=680,y=62)
            self.Button_resize.configure(command=self.add_Image2,text="ADD.")
            self.destroyLabel()
    def add_Image2(self):
        img_after = self.img
        img_add = addValueImage(img_after, float(self.Entry_resize.get()))
        image_addd = cv2.imread("NewAddImage.jpg")
        image_Add = ImageTk.PhotoImage(Image.fromarray(image_addd))
        self.label2.configure(image=image_Add)
        self.label2.image = image_Add
        self.Restoration_Filter.configure(text='ADD IMAGE.')
        self.Restoration_Filter.text = 'ADD IMAGE.'
        self.destroyLabel()
    def subtract_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="SUBTRACT IMAGE.",bg="black")
            self.Entry_resize.place(x=680,y=40)
            self.Button_resize.place(x=680,y=62)
            self.Button_resize.configure(command=self.subtract_Image2,text="SUBTRACT.")
            self.destroyLabel()
    def subtract_Image2(self):
        img_after = self.img
        img_subtract = subtractValueImage(img_after, float(self.Entry_resize.get()))
        image_subtract = cv2.imread("NewSubtractImage.jpg")
        image_subtract = ImageTk.PhotoImage(Image.fromarray(image_subtract))
        self.label2.configure(image=image_subtract)
        self.label2.image = image_subtract
        self.Restoration_Filter.configure(text='SUBTRACT IMAGE.')
        self.Restoration_Filter.text = 'SUBTRACT IMAGE.'
        self.destroyLabel()
    def multiply_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="MULTIPLY IMAGE.",bg="black")
            self.Entry_resize.place(x=680,y=40)
            self.Button_resize.place(x=680,y=62)
            self.Button_resize.configure(command=self.multiply_Image2,text="MULTIPLY.")
            self.destroyLabel()
    def multiply_Image2(self):
        img_after = self.img
        img_multiply = multiplyValueImage(img_after, float(self.Entry_resize.get()))
        image_multiply = cv2.imread("NewMultiplyImage.jpg")
        image_multiply = ImageTk.PhotoImage(Image.fromarray(image_multiply))
        self.label2.configure(image=image_multiply)
        self.label2.image = image_multiply
        self.Restoration_Filter.configure(text='MULTIPLY IMAGE.')
        self.Restoration_Filter.text = 'MULTIPLY IMAGE.'
        self.destroyLabel()
    #cv2.divide
    def divide_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="DIVIDE IMAGE.",bg="black")
            self.Entry_resize.place(x=680,y=40)
            self.Button_resize.place(x=680,y=62)
            self.Button_resize.configure(command=self.divide_Image2,text="DIVIDE.")
            self.destroyLabel()
    def divide_Image2(self):
        img_after = self.img
        img_divide = divideValueImage(img_after, float(self.Entry_resize.get()))
        image_divide = cv2.imread("NewDivideImage.jpg")
        image_divide = ImageTk.PhotoImage(Image.fromarray(image_divide))
        self.label2.configure(image=image_divide)
        self.label2.image = image_divide
        self.Restoration_Filter.configure(text='DIVIDE IMAGE.')
        self.Restoration_Filter.text = 'DIVIDE IMAGE.'
        self.destroyLabel()
    def rgb2Gray(self):
        if self.img_is_found:
            self.labelTop.configure(text="GRAY IMAGE.",bg="black")
            img_after = self.img
            img_gray = rgb2gray(img_after)
            image_gray = cv2.imread("rgb_to_gray.jpg")
            image_gray = ImageTk.PhotoImage(Image.fromarray(image_gray))
            self.label2.configure(image=image_gray)
            self.label2.image = image_gray
            self.Restoration_Filter.configure(text='GRAY IMAGE.')
            self.Restoration_Filter.text = 'GRAY IMAGE.'
            self.destroyLabel()
            self.destroyButton()
    #################################################
    def Add_saltpepperNoise(self):
        if self.img_is_found:
            self.labelTop.configure(text="SALT AND PEPPER.",bg="black")
            cv2.imwrite("NoSPNoiseImage.jpg",self.img)
            # salt and pepper noise.
            im_arr = np.asarray(self.path)
            aug = iaa.SaltAndPepper(p=0.05)
            im_arr = aug.augment_image(im_arr)
            im = ImageTk.PhotoImage(Image.fromarray(im_arr))#.convert('RGB')
            self.label2.configure(image=im)
            self.label2.image = im
            self.Restoration_Filter.configure(text='SALT AND PEPPER.')
            self.Restoration_Filter.text = 'SALT AND PEPPER.'
            cv2.imwrite("SPNoiseImage.jpg",im_arr)
            MEA=Mean_Absolute_error("NoSPNoiseImage.jpg","SPNoiseImage.jpg")
            MSA=Mean_square_error("NoSPNoiseImage.jpg","SPNoiseImage.jpg")
            PSNR=Peak_signal_to_noise_ratio("NoSPNoiseImage.jpg","SPNoiseImage.jpg")
            self.label_MAE.place(x=630, y=560)
            self.label_MSE.place(x=630, y=590)
            self.label_PSNR.place(x=630, y=620)
            self.label_MAE.configure(text = f"MAE:{round(float(MEA),2)}",fg="red")
            self.label_MSE.configure(text = f"MSE:{round(float(MSA),2)}",fg="red")
            self.label_PSNR.configure(text= f"PSNR:{round(float(PSNR),2)}",fg="red")
            self.destroyButton()
    def Add_gaussian_Noise(self):
        if self.img_is_found:
            self.labelTop.configure(text="GAUSSIAN NOISE.",bg="black")
            cv2.imwrite("NoGNoiseImage.jpg", self.img)
            im_arr = np.asarray(self.path)#.convert('gray')
            # gaussian noise.
            aug = iaa.AdditiveGaussianNoise(loc=0, scale=0.1*200)#0.1*255)
            im_arr = aug.augment_image(im_arr)
            im = ImageTk.PhotoImage(Image.fromarray(im_arr))#.convert('RGB')
            self.label2.configure(image=im)
            self.label2.image = im
            self.Restoration_Filter.configure(text='GAUSSIAN NOISE.')
            self.Restoration_Filter.text = 'GAUSSIAN NOISE.'
            cv2.imwrite("GNoiseImage.jpg", im_arr)
            MEA=Mean_Absolute_error("NoGNoiseImage.jpg", "GNoiseImage.jpg")
            MSA=Mean_square_error("NoGNoiseImage.jpg", "GNoiseImage.jpg")
            PSNR=Peak_signal_to_noise_ratio("NoGNoiseImage.jpg", "GNoiseImage.jpg")
            self.label_MAE.place(x=630,y=560)
            self.label_MSE.place(x=630,y=590)
            self.label_PSNR.place(x=630,y=620)
            self.label_MAE.configure(text=f"MAE:{round(float(MEA), 2)}",fg="red")
            self.label_MSE.configure(text=f"MSE:{round(float(MSA), 2)}",fg="red")
            self.label_PSNR.configure(text=f"PSNR:{round(float(PSNR),2)}",fg="red")
            self.destroyButton()
    def medianBlur(self):
        if self.img_is_found:
            self.labelTop.configure(text="MEDIAN BLUR.",bg="black")
            im_arr = np.asarray(self.path)
            img = cv.medianBlur(im_arr, 3)
            im = ImageTk.PhotoImage(Image.fromarray(img))  # .convert('RGB')
            self.label2.configure(image=im)
            self.label2.image = im
            self.Restoration_Filter.configure(text='MEDIAN BLUR.')
            self.Restoration_Filter.text = 'MEDIAN BLUR.'
            self.destroyLabel()
            self.destroyButton()
    def GaussianBlur(self):
        if self.img_is_found:
            self.labelTop.configure(text="GAUSSIAN BLUR.",bg="black")
            im_arr = np.asarray(self.path)
            img = cv.GaussianBlur(im_arr,(5,5),cv.BORDER_DEFAULT)
            im = ImageTk.PhotoImage(Image.fromarray(img))  # .convert('RGB')
            self.label2.configure(image=im)
            self.label2.image = im
            self.Restoration_Filter.configure(text='GAUSSIAN BLUR.')
            self.Restoration_Filter.text = 'GAUSSIAN BLUR.'
            self.destroyLabel()
            self.destroyButton()
    def Average_Filter(self):
        if self.img_is_found:
            self.labelTop.configure(text="AVERAGE FILTER.",bg="black")
            img = Average_Filter(self.img)
            image = cv2.imread("AverageFilterImage.jpg")
            im = ImageTk.PhotoImage(Image.fromarray(image))
            self.label2.configure(image=im)
            self.label2.image = im
            self.Restoration_Filter.configure(text='AVERAGE FILTER.')
            self.Restoration_Filter.text = 'AVERAGE FILTER.'
            self.destroyLabel()
            self.destroyButton()
    #Image EnhanceMent:
    def Negative_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="NEGATIVE IMAGE.",bg="black")
            img_after = self.img
            img_Negative = Negative_of_Image(img_after)
            image_Neg = cv2.imread("Negative_of_Image.jpg")
            image_Neg = ImageTk.PhotoImage(Image.fromarray(image_Neg))
            self.label2.configure(image=image_Neg)
            self.label2.image = image_Neg
            self.Restoration_Filter.configure(text='NEGATIVE IMAGE.')
            self.Restoration_Filter.text = 'NEGATIVE IMAGE.'
            self.destroyLabel()
            self.destroyButton()
    def Log_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="LOG IMAGE.",bg="black")
            img_after = self.img
            img_Log = Log_transformation(img_after)
            image_Log = cv2.imread("Log_of_Image.jpg")
            image_Log_T = ImageTk.PhotoImage(Image.fromarray(image_Log))
            self.label2.configure(image=image_Log_T)
            self.label2.image = image_Log_T
            self.Restoration_Filter.configure(text='LOG IMAGE.')
            self.Restoration_Filter.text = 'LOG IMAGE.'
            self.destroyLabel()
            self.destroyButton()
    def Power_Low_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="POWER LOW IMAGE.",bg="black")
            self.Entry_resize.place(x=680, y=40)
            self.Button_resize.place(x=680, y=62)
            self.Button_resize.configure(command=self.Power_Low_Image2, text="GAMMA.")
            self.destroyLabel()
    def Power_Low_Image2(self):
        img_after = self.img
        img_Power_Low =
Power_Law_transformation(img_after,float(self.Entry_resize.get()))
        image_Power_Low = cv2.imread("Power_Law_Image.jpg")
        image_Power_L = ImageTk.PhotoImage(Image.fromarray(image_Power_Low))
        self.label2.configure(image=image_Power_L)
        self.label2.image = image_Power_L
        self.Restoration_Filter.configure(text='POWER LOW IMAGE.')
        self.Restoration_Filter.text = 'POWER LOW IMAGE.'
        self.destroyLabel()
    def Thresholding_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="THRESHOLDING IMAGE.",bg="black")
            img_after = self.img
            img_Thresholding = Thresholded_transformation(img_after)
            image_Thres = cv2.imread("Thresholded_Image.jpg")
            image_Thres = ImageTk.PhotoImage(Image.fromarray(image_Thres))
            self.label2.configure(image=image_Thres)
            self.label2.image = image_Thres
            self.Restoration_Filter.configure(text='THRESHOLDING IMAGE.')
            self.Restoration_Filter.text = 'THRESHOLDING IMAGE.'
            self.destroyLabel()
            self.destroyButton()
    def Fixed_intensity_Image(self):
        if self.img_is_found:
            self.labelTop.configure(text="FIXED INTENSITY IMAGE.",bg="black")
            img_after = self.img
            img_Fixed_I = Fixed_intensity_transformation(img_after)
            image_Fixed = cv2.imread("Fixed_intensity.jpg")
            image_Fixed = ImageTk.PhotoImage(Image.fromarray(image_Fixed))
            self.label2.configure(image=image_Fixed)
            self.label2.image = image_Fixed
            self.Restoration_Filter.configure(text='FIXED INTENSITY IMAGE.')
            self.Restoration_Filter.text = 'FIXED INTENSITY IMAGE.'
            self.destroyLabel()
            self.destroyButton()
    def Full_Scale_Contrast(self):
        if self.img_is_found:
            self.labelTop.configure(text="FULL SCALE CONTRAST IMAGE.",bg="black")
            img_after = self.img
            cv2.imwrite('Full_Scale_Contrast_image.jpg', img_after)
            img_Full_Scale = Full_Scale_Contrast_Stretch('Full_Scale_Contrast_image.jpg')
            image_Full_S = cv2.imread("Full_Scale_Contrast.jpg")
            image_Full_S = ImageTk.PhotoImage(Image.fromarray(image_Full_S))
            self.label2.configure(image=image_Full_S)
            self.label2.image = image_Full_S
            self.Restoration_Filter.configure(text='FULL SCALE CONTRAST IMAGE.')
            self.Restoration_Filter.text = 'FULL SCALE CONTRAST IMAGE.'
            self.destroyLabel()
            self.destroyButton()
    def Histogram_Equalization(self):
        if self.img_is_found:
            self.labelTop.configure(text="HISTOGRAM EQUALIZATION.",bg="black")
            img_after = self.img
            cv2.imwrite("HistogramEqualImage.jpg",img_after)
            img_Histogram = Histogram_Equalization("HistogramEqualImage.jpg")
            image_Histo = cv2.imread("histogramEqualizationImage.jpg")
            image_Histo = ImageTk.PhotoImage(Image.fromarray(image_Histo))
            self.label2.configure(image=image_Histo)
            self.label2.image = image_Histo
            self.Restoration_Filter.configure(text='HISTOGRAM EQUALIZATION.')
            self.Restoration_Filter.text = 'HISTOGRAM EQUALIZATION.'
            self.destroyLabel()
            self.destroyButton()
    def Edge_Detection(self):
        if self.img_is_found:
            self.labelTop.configure(text="EDGE DETECTION IMAGE.",bg="black")
            img_after = self.img
            cv2.imwrite("EDImage.jpg", img_after)
            img_E = Edge_Detection("EDImage.jpg")
            img_E_D = cv2.imread('Edge_DetectionImage.jpg')
            img_after_ED = ImageTk.PhotoImage(Image.fromarray(img_E_D))
            self.label2.configure(image=img_after_ED)
            self.label2.image = img_after_ED
            self.Restoration_Filter.configure(text='EDGE DETECTION.')
            self.Restoration_Filter.text = 'EDGE DETECTION.'
            self.destroyLabel()
            self.destroyButton()
    def save_image(self,image):
        cv.imwrite("psf_image.jpg", image)
    def destroyLabel(self):
        self.label_MAE.configure(fg="white")
        self.label_MSE.configure(fg="white")
        self.label_PSNR.configure(fg="white")
    def destroyButton(self):
        self.Entry_resize.place_forget()#Place_forget().
        self.Button_resize.place_forget()#Place_forget().
    def __init__(self):#Constructor.
        Tk.__init__(self)
        self.title('DIGITAL IMAGE PROCESSING')
        self.window_size()
        self.createWidgets()
        self.config(menu=self.menuBar)
        self.Img_original = Label(self, width=30, text='ORIGINAL IMAGE',
font=Font(size=23, weight='bold'))
        self.Restoration_Filter = Label(self, width=30, text = None, font=Font(size=23,
weight='bold'))
        self.labelTop = Label(text = None,font=('Arial',15, 'bold'),fg="white")
        self.labelTop.pack(side=TOP,padx=25)
        ####################################
        ####################################
        ####################################
        self.label = Label(image = None, width=700, height=420)
        self.label2 = Label(image = None, width=700, height=420)
        self.label.pack(side=LEFT, padx=25)
        self.Img_original.place(relx=0.055, rely=0.035)
        self.label2.pack(side=RIGHT, padx=25)
        self.Restoration_Filter.place(relx=0.55, rely=0.035)
        #####################################
        self.Entry_resize = Entry(self)
        self.Button_resize = Button(self,text="RESIZE
IMAGE",width=16,command=self.resize_Image2)
        #####################################
        self.label_MAE = Label(self,text="",width=20,font=('Arial',15, 'bold'))
        self.label_MSE = Label(self,text="",width=20,font=('Arial',15, 'bold'))
        self.label_PSNR = Label(self,text="",width=20,font=('Arial',15, 'bold'))

##################################################################################

##################################################################################
```