**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Mohsin Kamal **Dated:** 13/10/2023

**Semester:** 7th **Section:** BEE 12C

**CS-477 Computer Vision**

Lab 5: Intensity Transformation and Spatial filtering

**Group Members**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Teamwork** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Table of Contents**

[2 Intensity Transformation and Spatial filtering 3](#_Toc148110985)

[2.1 Introduction 3](#_Toc148110986)

[2.2 Objectives 3](#_Toc148110987)

[2.3 Software 3](#_Toc148110988)

[3 Lab Tasks 4](#_Toc148110989)

[3.1 Task 1 4](#_Toc148110990)

[3.2 Task 2 5](#_Toc148110991)

[4 Part 2 – Spatial Filtering Basics (Open Ended) 10](#_Toc148110992)

[4.1 Task 1 – Effect of Averaging and the Size of Averaging Filters 10](#_Toc148110993)

[4.2 Task 2 – Gaussian Smoothing 12](#_Toc148110994)

[4.3 Task 3 – Un-sharp Masking 14](#_Toc148110995)

[4.4 Task 4 – Effect of Averaging and the Size of Averaging Filters 15](#_Toc148110996)

[5 Conclusion 16](#_Toc148110997)

# Intensity Transformation and Spatial filtering

## Introduction

Contrast stretching and histogram equalization are two common image processing techniques used to enhance the contrast of an image. Contrast stretching linearly stretches the range of intensity values in an image to make better use of the available dynamic range. Histogram equalization is a more sophisticated technique that redistributes the intensity values in an image to create a more uniform histogram.

In this lab, we will learn how to create and apply contrast stretching and histogram equalization to images using Python. We will also compare the two techniques and discuss their advantages and disadvantages.

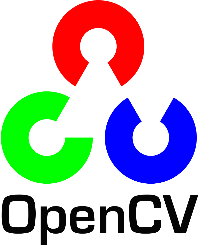
## Objectives

The following are the main objectives of this lab:

* To create and apply contrast stretching on histograms.
* To create and apply histogram equalization.

## Software

OpenCV is a library that focuses on image processing and computer vision. An image is an array of colored squares called pixels. Each pixel has a certain location in the array and color values in BGR format. By referring to the array indices, the individual pixels or a range of pixels can be accessed and modified. OpenCV provides many functions for resizing, rotating, and placing objects in images. Rotation involves computing a 2-D rotation matrix which is applied for the transformation of the image.



# Lab Tasks

## Task 1

1. Load a low contrast image “wiki.jpg”
2. Create an algorithm which applies contrast stretching (pick any implementation you like i.e. either formula or points based)
   1. NOTE: For now, you can select the stretching limits by your own.
3. Apply same technique on “lowcon.tif”
4. Summarize your findings on how to extend or automate the task!

### TASK 1 CODE STARTS HERE ###

img = cv2.imread("lab5\_wiki.jpg", cv2.IMREAD\_GRAYSCALE)

img2 = cv2.imread("lab5\_lowcon.tif", cv2.IMREAD\_GRAYSCALE)

*# Constrast stretching*

*# S = ((Smax - Smin) / (Rmax - Rmin)) \* (R - Rmin) + Smin*

*def* contrast\_stretching(*img*, *rmin*, *rmax*, *smin*, *smax*):

    img = ((smax - smin) / (rmax - rmin)) \* (img - rmin) + smin

    img = img.astype(np.uint8)

    return img

*# empirical values*

c\_img = contrast\_stretching(img, 100, 200, 0, 255)

*# Display the images*

cv2.imshow("wiki", img)

cv2.imshow("contrast\_strecthed", c\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

*# empirical values*

c\_img2 = contrast\_stretching(img2, 80, 160, 0, 255)

*# Display the image*

cv2.imshow("lowcon", img2)

*# Display the image*

cv2.imshow("contrast\_strecthed", c\_img2)

cv2.waitKey(0)

cv2.destroyAllWindows()

### TASK 1 CODE ENDS HERE ###

### TASK 1 SCREENSHOT STARTS HERE ###

### TASK 1 SCREENSHOT ENDS HERE ###

**### TASK 1 Description**

To extend or automate the task of contrast stretching, we can use an algorithm that automatically selects the stretching limits based on the image histogram. One such method is known as **adaptive contrast stretching**. This method uses the image histogram to calculate the minimum and maximum pixel values that should be stretched to the full range of the output image. This ensures that the contrast of the output image is maximized without clipping any pixels.

## Task 2

1. Write a program that equalizes the histogram of a given image. Consider the formula below

A mathematical equation with numbers and symbols

Description automatically generated

where ‘s’ and ‘r’ are the output and input pixel intensities respectively. ‘L’ is the maximum intensity value (for n bit image L = 2^n). The probability of occurrence of the intensity level rj in the image is approximated by

1. Show the comparison of histograms before and after equalization obtained using:
   1. Your Implementation of the Algorithm
   2. OpenCV’s implementation of Histogram Equalization
2. Conclude your findings on following images and analyze the workings of histogram:
   1. dark.tif
   2. bright.tif
   3. lowcon.tif
   4. Wiki.jpg

### TASK 2 CODE STARTS HERE ###

*def* custom\_histogram(*img*):

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    rows, cols = gray.shape

    hist = np.zeros(256)

    for i in range(rows):

        for j in range(cols):

            hist[gray[i, j]] += 1

    hist /= (rows \* cols)

    cdf = deepcopy(hist)

    for i in range(1, 256):

        cdf[i] += cdf[i-1]

    return hist, cdf

img = cv2.imread("lab5\_wiki.jpg")

hist, cdf = custom\_histogram(img)

equalized\_img = equalize\_histogram(img)

eq\_hist, eq\_cdf = custom\_histogram(equalized\_img)

plt.rcParams["figure.figsize"] = (8, 6)

plt.subplot(3, 2, 1)

plt.stem(hist, *markerfmt*="", *linefmt*="r-", *basefmt*="r--")

plt.title("Histogram")

plt.subplot(3, 2, 2)

plt.plot(cdf, *color*="g")

plt.title("CDF")

plt.subplot(3, 2, 3)

plt.stem(eq\_hist, *markerfmt*="", *linefmt*="r-", *basefmt*="r--")

plt.title("Equalized Histogram (Custom)")

plt.subplot(3, 2, 4)

plt.plot(eq\_cdf, *color*="g")

plt.title("Equalized CDF (Custom)")

*# OpenCV's implementation of Histogram Equalization*

img = cv2.imread("lab5\_wiki.jpg")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

equalized\_img = cv2.equalizeHist(gray)

hist = cv2.calcHist([equalized\_img], [0], None, [256], [0, 256])

cdf = np.cumsum(hist)

plt.subplot(3, 2, 5)

plt.stem(hist, *markerfmt*="", *linefmt*="r-", *basefmt*="r--")

plt.title("Equalized Histogram (OpenCV)")

plt.subplot(3, 2, 6)

plt.plot(cdf, *color*="g")

plt.title("Equalized CDF (OpenCV))")

plt.tight\_layout()

plt.show()

**On More Images**

images = ["lab5\_dark.tif", "lab5\_bright.tif", "lab5\_lowcon.tif", "lab5\_wiki.jpg"]

for img\_name in images:

    img = cv2.imread(img\_name)

    hist, cdf = custom\_histogram(img)

    equalized\_img = equalize\_histogram(img)

    eq\_hist, eq\_cdf = custom\_histogram(equalized\_img)

    plt.rcParams["figure.figsize"] = (8, 4)

    plt.subplot(2, 2, 1)

    plt.stem(hist, *markerfmt*="", *linefmt*="r-", *basefmt*="r--")

    plt.title(*f*"{img\_name} - Histogram")

    plt.subplot(2, 2, 2)

    plt.plot(cdf, *color*="g")

    plt.title(*f*"{img\_name} - CDF")

    plt.subplot(2, 2, 3)

    plt.stem(eq\_hist, *markerfmt*="", *linefmt*="r-", *basefmt*="r--")

    plt.title(*f*"{img\_name} - Equalized Histogram")

    plt.subplot(2, 2, 4)

    plt.plot(eq\_cdf, *color*="g")

    plt.title(*f*"{img\_name} - Equalized CDF")

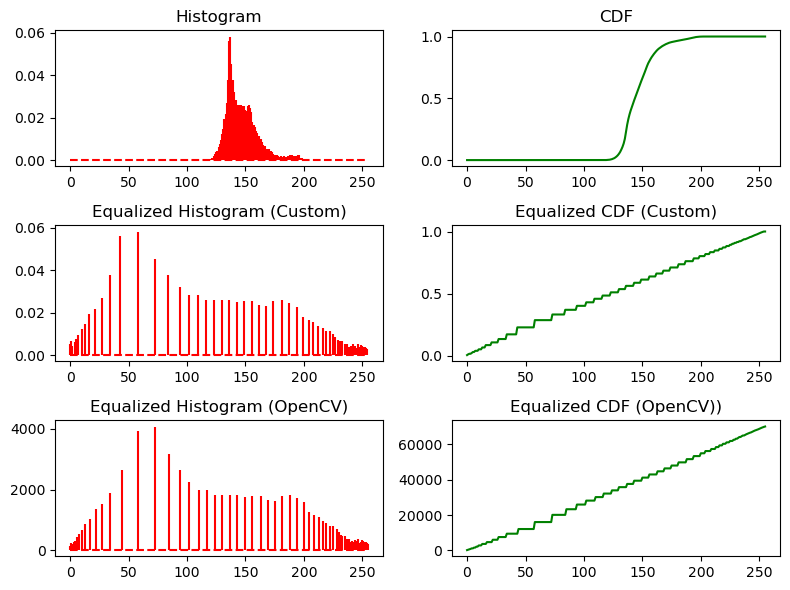
    plt.tight\_layout()

    plt.show()

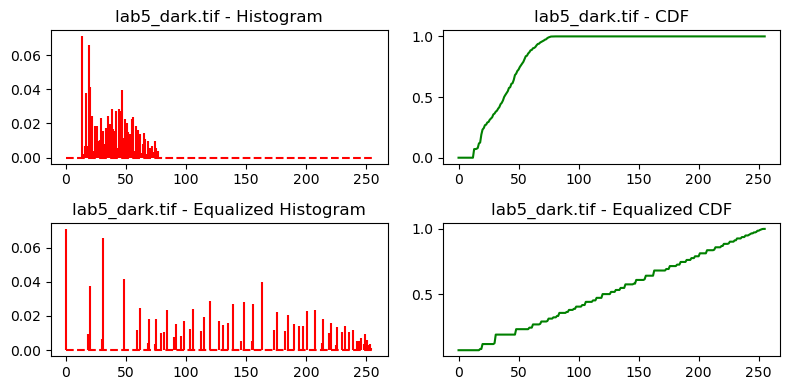
### TASK 2 CODE ENDS HERE ###

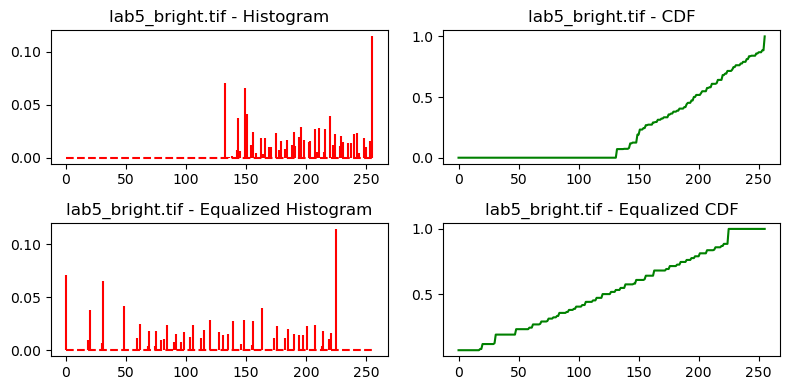
### TASK 2 SCREENSHOT STARTS HERE ###

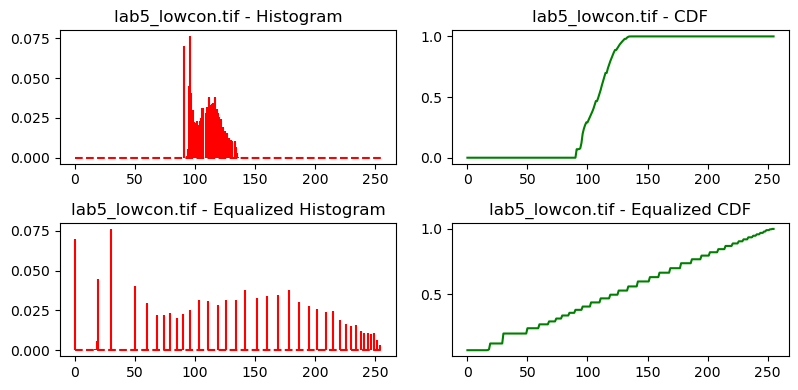
**Comparison with OpenCV**

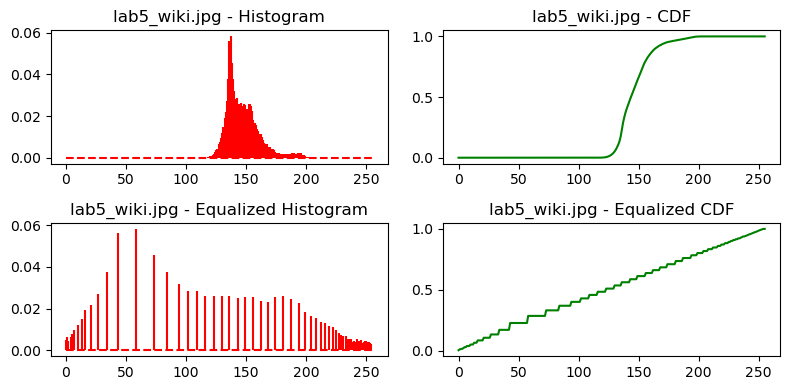


**On More Images**









### TASK 2 SCREENSHOT ENDS HERE ###

**### TASK 2 Description**

# Part 2 – Spatial Filtering Basics (Open Ended)

## Task 1 – Effect of Averaging and the Size of Averaging Filters

Consider the following image. Apply averaging with a filter size of 3\*3, 5\*5, 15\*15, and 35\*35.



img = cv2.imread("lab5\_smoothing.tif", cv2.IMREAD\_GRAYSCALE)

*# 3x3 averaging filter*

avg\_filter = np.ones((3, 3)) / 9

avg\_img = cv2.filter2D(img, -1, avg\_filter)

*# 5x5 averaging filter*

avg\_filter = np.ones((5, 5)) / 25

avg\_img2 = cv2.filter2D(img, -1, avg\_filter)

*# 15x15 averaging filter*

avg\_filter = np.ones((15, 15)) / 225

avg\_img3 = cv2.filter2D(img, -1, avg\_filter)

*# 35x35 averaging filter*

avg\_filter = np.ones((35, 35)) / 1225

avg\_img4 = cv2.filter2D(img, -1, avg\_filter)

plt.rcParams["figure.figsize"] = (6, 6)

plt.subplot(2, 2, 1)

plt.imshow(avg\_img, *cmap*="gray")

plt.title("3x3 Averaging Filter")

plt.subplot(2, 2, 2)

plt.imshow(avg\_img2, *cmap*="gray")

plt.title("5x5 Averaging Filter")

plt.subplot(2, 2, 3)

plt.imshow(avg\_img3, *cmap*="gray")

plt.title("15x15 Averaging Filter")

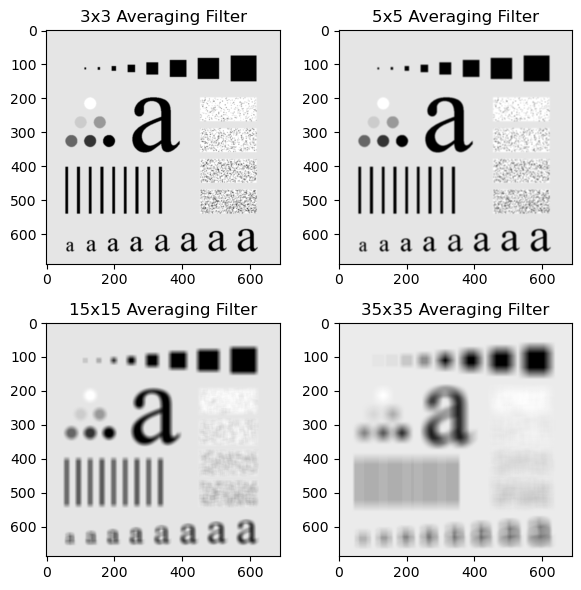
plt.subplot(2, 2, 4)

plt.imshow(avg\_img4, *cmap*="gray")

plt.title("35x35 Averaging Filter")

plt.tight\_layout()

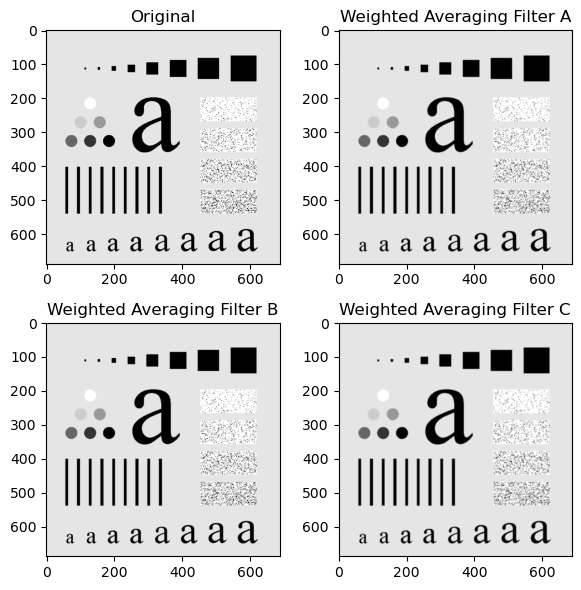
plt.show()



**What do you observe when increasing the size of the filter and why?**

We can see that as we increase the size of the filter, the image gets more and more blurred. This is because the averaging filter is a low pass filter, and it removes the high frequency components from the image. Additionally, the nosie regions in the image are denoised, to a certain extent, as we increase the size of the filter.

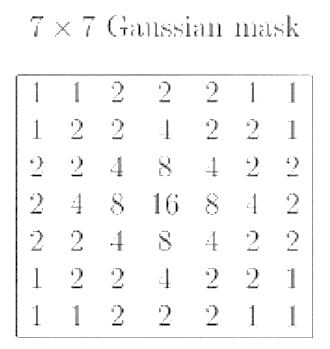
**Apply different weighted averaging filters on the same image and note down the effect they have on the input image.**

****

Compared to averaging filters, the image is less blurred in case of weighted averaging filters, however, the noise regions are denoised, again, to a certain extent, but more effectively than averaging filters.

## Task 2 – Gaussian Smoothing

Apply the following Gaussian filter to the image given above. Here, the σ = 1.4. What impact do you think happens when the value of σ is increased? Don’t forget the normalizing factor while applying the given Gaussian filter.



img = cv2.imread("lab5\_smoothing.tif", cv2.IMREAD\_GRAYSCALE)

*# Gaussian filter 7x7*

gaussian\_filter = np.array(

    [

        [1, 1, 2, 2, 2, 1, 1],

        [1, 2, 2, 4, 2, 2, 1],

        [2, 2, 4, 8, 4, 2, 2],

        [2, 4, 8, 16, 8, 4, 2],

        [2, 2, 4, 8, 4, 2, 2],

        [1, 2, 2, 4, 2, 2, 1],

        [1, 1, 2, 2, 2, 1, 1],

    ], *dtype*=np.float32

)

gaussian\_filter /= 140

gaussian\_img = cv2.filter2D(img, -1, gaussian\_filter)

plt.rcParams["figure.figsize"] = (6, 6)

plt.subplot(1, 2, 1)

plt.imshow(img, *cmap*="gray")

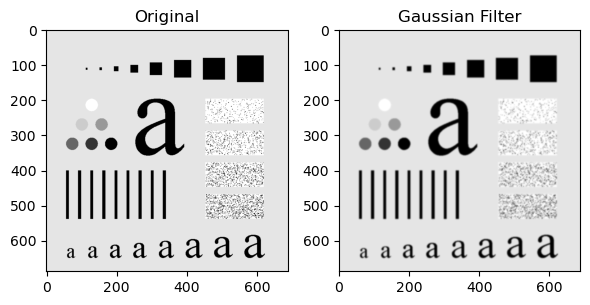
plt.title("Original")

plt.subplot(1, 2, 2)

plt.imshow(gaussian\_img, *cmap*="gray")

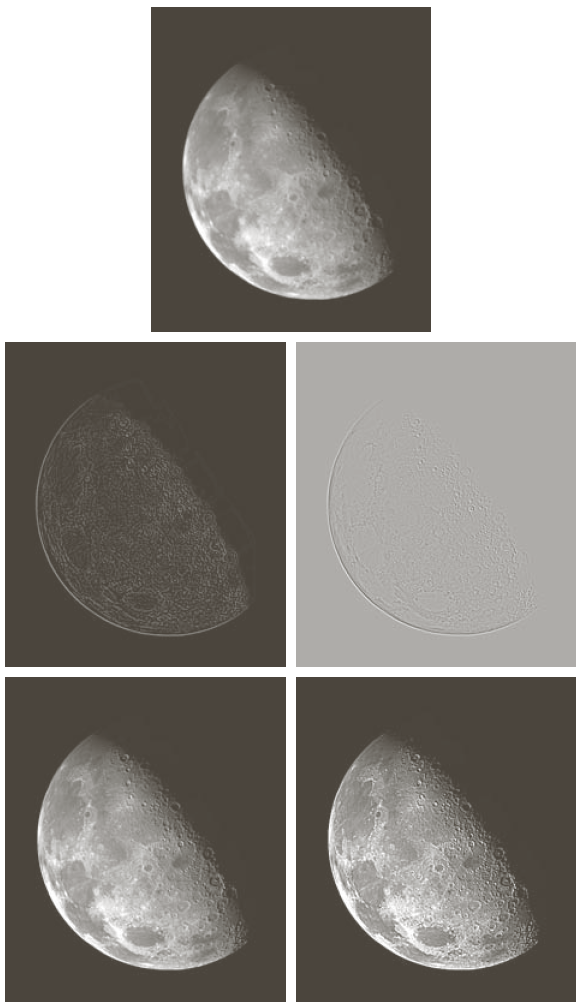
plt.title("Gaussian Filter")

plt.tight\_layout()



As the value of sigma increases, the image becomes more blurred. This is because the value of sigma determines the standard deviation of the Gaussian distribution. The higher the standard deviation, the more the values are spread out from the mean, the bigger the filter size and hence more blurred the image is.

## Task 3 – Un-sharp Masking



img = cv2.imread("lab5\_unsharp.png", cv2.IMREAD\_GRAYSCALE)

*# 3x3 averaging filter*

avg\_filter = np.ones((3, 3)) / 9

avg\_img = cv2.filter2D(img, -1, avg\_filter)

*# Gaussian filter 7x7*

gaussian\_filter = np.array(

    [

        [1, 1, 2, 2, 2, 1, 1],

        [1, 2, 2, 4, 2, 2, 1],

        [2, 2, 4, 8, 4, 2, 2],

        [2, 4, 8, 16, 8, 4, 2],

        [2, 2, 4, 8, 4, 2, 2],

        [1, 2, 2, 4, 2, 2, 1],

        [1, 1, 2, 2, 2, 1, 1],

    ], *dtype*=np.float32

)

gaussian\_filter /= 140

gaussian\_img = cv2.filter2D(img, -1, gaussian\_filter)

*# Unsharp masking*

unsharp\_img = cv2.addWeighted(img, 3, avg\_img, -2, 0)

unsharp\_img2 = cv2.addWeighted(img, 3, gaussian\_img, -2, 0)

plt.rcParams["figure.figsize"] = (8, 6)

plt.subplot(2, 3, 1)

plt.imshow(img, *cmap*="gray")

plt.title("Original")

plt.subplot(2, 3, 2)

plt.imshow(unsharp\_img, *cmap*="gray")

plt.title("Unsharp Masking (3x3)")

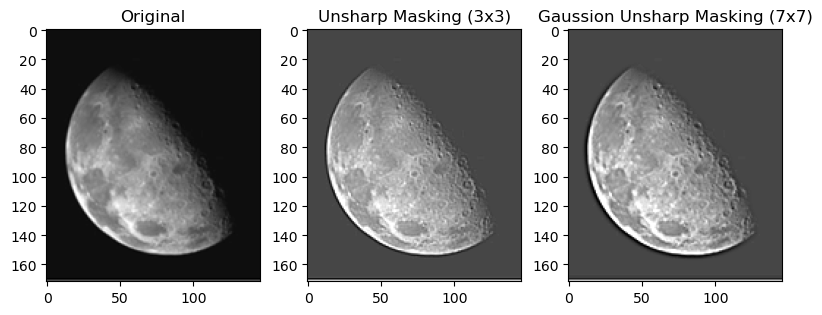
plt.subplot(2, 3, 3)

plt.imshow(unsharp\_img2, *cmap*="gray")

plt.title("Gaussion Unsharp Masking (7x7)")

plt.tight\_layout()

plt.show()



Using a 3x3 averaging filter for unsharp masking will result in a more subtle sharpening effect than using a 7x7 averaging filter. This is because the 3x3 averaging filter will preserve more of the high-frequency components of the image.

## Task 4 – Effect of Averaging and the Size of Averaging Filters

Download the following image "[two\_cats.jpg](http://www.cs.uregina.ca/Links/class-info/425-nova/Lab3/Exercise/two_cats/two_cats.jpg). (A): Use a spatial filter to get the horizontal edges of the image. (B): Use a spatial filter to get the vertical edges of the image. (C): Add the horizontal edge matrix to the vertical edge matrix to yield the following results (the image on the right).

*# Load the image*

img = cv2.imread("lab5\_two\_cats.jpg", cv2.IMREAD\_GRAYSCALE)

*# Define horizontal and vertical edge detection filters*

h\_filter = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

v\_filter = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

*# Apply the filters to the image*

h\_edges = cv2.filter2D(img, -1, h\_filter)

v\_edges = cv2.filter2D(img, -1, v\_filter)

*# Add the horizontal and vertical edges to get the final result*

edges = cv2.add(h\_edges, v\_edges)

*# Display the result*

plt.rcParams["figure.figsize"] = (6, 6)

plt.subplot(2, 2, 1)

plt.imshow(img, *cmap*="gray")

plt.title("Original")

plt.subplot(2, 2, 2)

plt.imshow(h\_edges, *cmap*="gray")

plt.title("Horizontal Edges")

plt.subplot(2, 2, 3)

plt.imshow(v\_edges, *cmap*="gray")

plt.title("Vertical Edges")

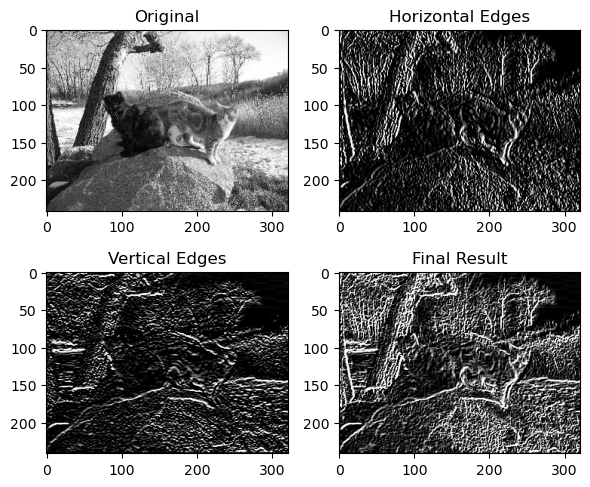
plt.subplot(2, 2, 4)

plt.imshow(edges, *cmap*="gray")

plt.title("Final Result")

plt.tight\_layout()

plt.show()



# Conclusion

In this lab, we learned how to create and apply contrast stretching and histogram equalization to images using Python. We also compared the two techniques and discussed their advantages and disadvantages. Contrast stretching is a simple but effective technique for enhancing the contrast of an image. It is easy to implement and can be used to improve the contrast of a wide range of images. However, contrast stretching can sometimes lead to over-enhancement, which can make the image look artificial. Histogram equalization is a more sophisticated technique for enhancing the contrast of an image. It is more effective than contrast stretching at revealing details in low-contrast images. However, histogram equalization can sometimes lead to noise amplification and loss of saturation in color images.