Image Processing and Pattern Recognition - Lab work

Name: Harthik Mallichetty
Roll no: 222010320050

INDEX

Date	Experiment Number	Experiment	Remarks
11-12-2023	1	Image negative	
	2	Logarithmic transform	
	3	Power Law	
	4	Thresholding	
	5	Clipping	
18-12-2023	6	Contrast stretching	
	7	Intensity slicing	
1-1-2024	8	Nonlinear smoothing filters (min, max, median)	
	9	Linear smoothing filters (Average, Low pass, Weighted average)	
8-1-2024	10	1st order sharpening filters - 1. Robert's 2. Sobel's 3. Prewitt's	
29-1-2024	11	2nd order sharpening filters - Laplacian	

Image Processing and Pattern recognition

Week 1

Aim: Understanding Image transformation methods: Image negative, Logarithmic transform, Power law, Thresholding, Clipping using MATLAB.

Requirements: MATLAB software

Directory used throughout week 1 -
/Users/harthikmallichetty/Downloads/grayscale.jpeg

Image used to demonstrate transforms throughout week 1 - (grayscale.jpeg)

Value - 1204 x 1880 uint8

Image Negative

Procedure

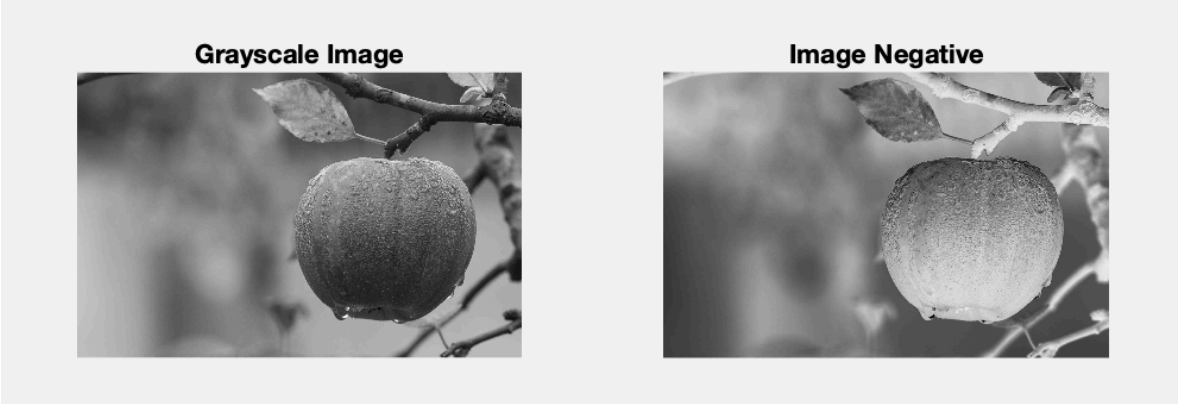
In image processing, finding the negative of an image involves obtaining the complement of pixel intensities. The negative of an image is essentially an inversion of its colors, where the darkest areas become the brightest, and vice versa.

Here, to compute image negative, we first find the L value by finding the highest pixel value and define the range by calculating the number of bits the matrix holds. After computing L, we use it to find the complement of each pixel to form a negative matrix.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Grayscale Image");
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
negative=(L-1)-a;
subplot(1,2,2);
imshow(negative);
title("Image Negative");
```

Output



Logarithmic Transform

Procedure

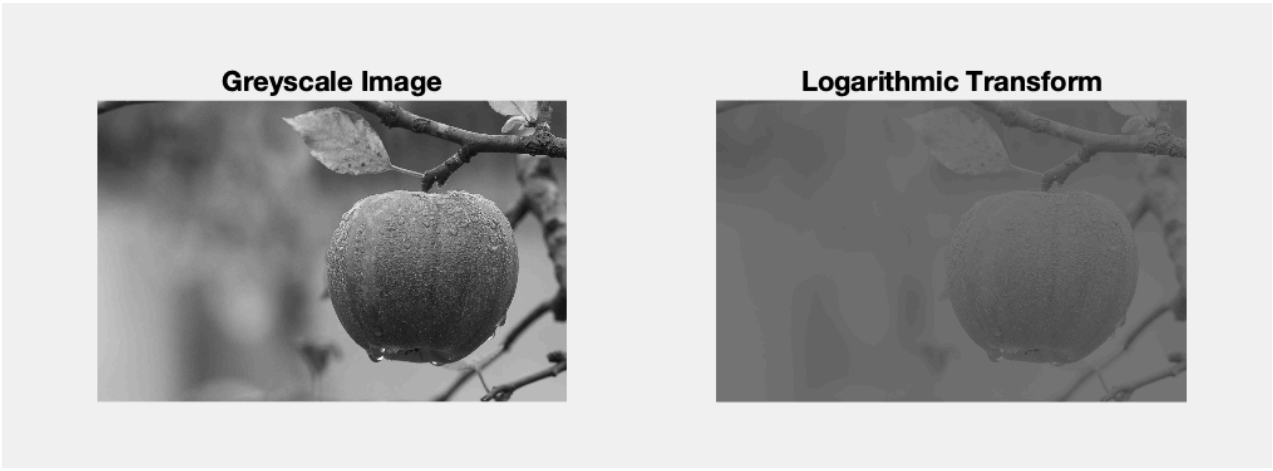
In image processing, logarithmic transformation is a common technique used to enhance the visibility of details in dark regions of an image. This transformation is particularly useful when dealing with images that have a wide range of pixel intensities, especially those with low-light or shadowed areas.

The general form of a logarithmic transformation is given by: $\mathbf{Log_b(x)=y}$
Here, x is the original pixel value, and y is the resulting value.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
c = 0.2;
x = double(1+a);
logimg = c*log10(x);
subplot(1,2,2);
imshow(logimg);
title("Logarithmic Transform");
```

Output



Power Law Transform

Procedure

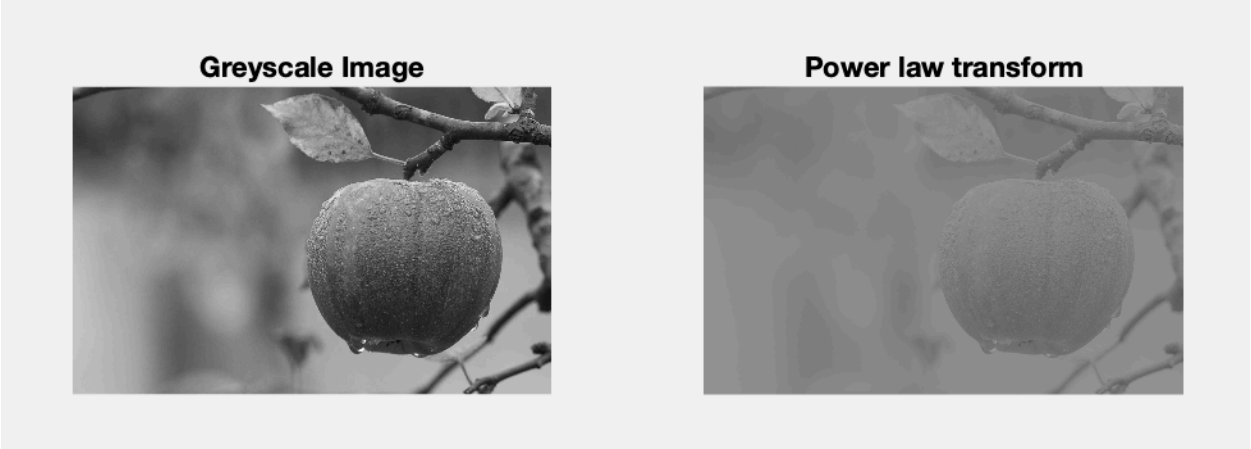
Power-law transformation, also known as power-law normalization or gamma correction, is a technique used in image processing to adjust the contrast of an image. It involves raising the intensity values of the pixels in an image to a certain power, typically denoted by the gamma (γ) parameter.

The general form of the power-law transformation is given by the equation: **T(r)=c*(r^γ)** Here, T(r) is the transformed intensity value, r is the original intensity value, c is a constant scaling factor, γ is the gamma parameter.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
c = 0.2;
g = 0.2;
powerlaw = c*(double(a).^g);
subplot(1,2,2);
imshow(powerlaw);
title("Power law transform");
```

Output



Thresholding Transform

Procedure

Thresholding transformation is a common image processing technique used to segment an image into regions based on intensity levels. The basic idea behind thresholding is to convert a grayscale image into a binary image, where pixels are classified as either foreground or background, depending on whether their intensity values are above or below a certain threshold.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r = 128;
thres = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if a(j,k) >= r
        thres(j,k) = L-1;
     else
        thres(j,k) = 0;
     end
  end
end
subplot(1,2,2);
imshow(thres);
title("Thresholding");
```

Output

Clipping Transform

Procedure

In image processing, clipping transformation is a technique used to limit or restrict pixel values within a certain range. This is often done to prevent overflow or underflow of pixel intensities, especially when performing mathematical operations on images. The most common scenario for clipping is in the context of digital image processing, where pixel values are typically represented as integers within a specific range (e.g., 0 to 255 for an 8-bit image). The process involves setting any pixel value that falls outside the desired range to the closest valid endpoint.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r1 = 100;
r2 = 140;
clip = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
```

```
  end
end
for j = 1:x
  for k = 1:y
    if and(a(j,k) >= r1, a(j,k)<= r2)
       clip(j,k) = L-1;
    else
       clip(j,k) = 0;
    end
  end
end
subplot(1,2,2);
imshow(clip);
title("Clipping");
```

Output

Conclusion: By applying methods on a grayscale image, we understood how transformation techniques work on an image in MATLAB.

Image Processing and Pattern recognition

Week 2

Aim: Understanding contrast stretching and intensity slicing using MATLAB.

Requirements: MATLAB software

Directory used throughout week 2 -
/Users/harthikmallichetty/Downloads/grayscale.jpeg

Image used to demonstrate transforms throughout week 2 - (grayscale.jpeg)

Value - 1204 x 1880 uint8

Contrast Stretching

Contrast stretching, also known as normalization or intensity rescaling, is a technique used in image processing to enhance the contrast of an image. The goal is to spread the range of pixel intensities across the entire dynamic range, making the image visually more appealing and improving the visibility of details. The process involves linearly scaling the pixel values of the image from their original range to a new range.

$$\text{Output pixel value} = \frac{(\text{Input pixel value} - \text{Min intensity}) \times (\text{New Max range})}{\text{Original intensity range}} + \text{New Min range}$$

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
org = 0;
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
r1 = 40;
r2 = 140;
s1 = 120;
s2 = 180;
alp = (s1 - org)/(r1 - org);
bet = (s2 - s1)/(r2 - r1);
gam = (L-1 - s2)/(L-1 - r2);
contstre = zeros(x,y);
for j = 1:x
  for k = 1:y
     pix = a(j,k);
     if and(pix > 0, pix < r1)
        contstre(j,k) = alp*pix;
```

```
      else if and(r1 <= pix, pix < r2)
            contstre(j,k) = (bet*(pix-r2)) + s1;
      else if and(r2 <= pix, pix <= r2)
            contstre(j,k) = (gam*(pix-r2)) + s2;
      end
      end
      end
  end
end
subplot(1,2,2);
imshow(contstre);
title("Contrast Stretching");
```

Output

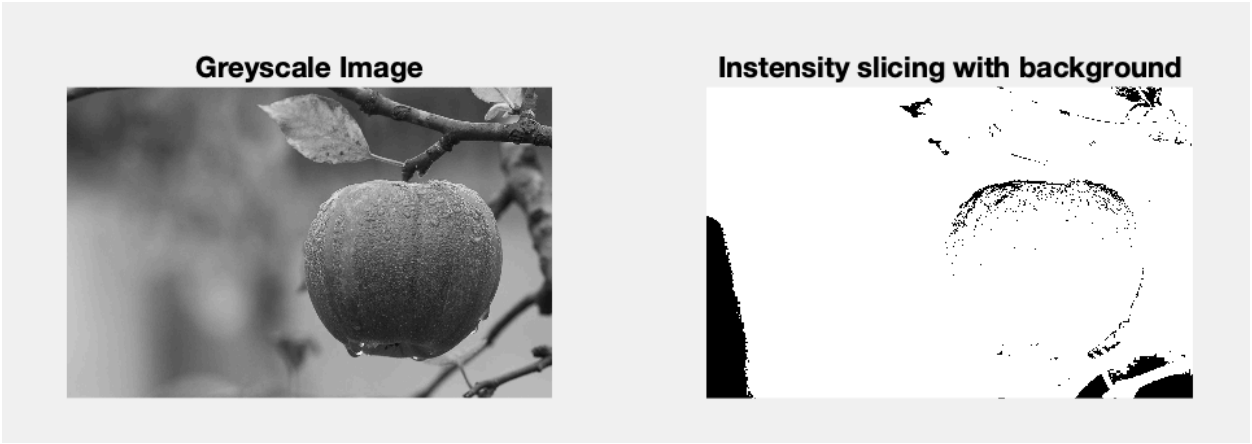
Intensity Slicing

Intensity slicing, also known as thresholding or range mapping, is a simple image processing technique that involves highlighting or isolating specific intensity ranges in an image. The idea is to set pixels with intensity values within a certain range to a specific value (typically white), while pixels outside that range are set to another value (typically black). This technique is often used for segmentation or highlighting specific features in an image.

Code (intensity slicing with background)

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r1 = 20;
r2 = 180;
intsliwbg = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if and(a(j,k) >= r1, a(j,k) <= r2)
        intsliwbg(j,k) = L-1;
     end
  end
end
subplot(1,2,2);
imshow(intsliwbg);
title("Instensity slicing with background");
```

Output



Code (Intensity slicing without background)

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r1 = 100;
r2 = 140;
intsliwobg = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if and(a(j,k) >= r1, a(j,k) <=r2)
        intsliwobg(j,k) = L-1;
     else
        intsliwobg(j,k) = 0;
     end
  end
end
subplot(1,2,2);
```

```
imshow(intsliwobg);
title("Intensity slicing without background");
```

Output

Conclusion: In this experiment we understood how to use MATLAB to successfully apply contrast stretching, intensity slicing with background, intensity slicing without background techniques.

Image Processing and Pattern recognition

Week 3

Aim: Understanding and applying spatial filters using MATLAB.

Requirements: MATLAB software

Directory used throughout week 3 -
/Users/harthikmallichetty/Downloads/grayscale.jpeg

Image used to demonstrate transforms throughout week 3 - (grayscale.jpeg)

Value - 1204 x 1880 uint8

Spatial filters in image processing are used to modify the pixel values of an image based on their spatial arrangement. These filters operate on the pixel values in a local neighborhood around each pixel, and they can be broadly categorized into linear and non-linear spatial filters.

Non-linear spatial filters

Non-linear spatial filters in image processing operate on individual pixels without using convolution. These filters apply a specific function to each pixel independently, typically based on the local pixel values within a specified neighborhood. Non-linear filters are particularly useful for tasks where preserving edges, details, or handling outliers is important.

1. Minimum

Replaces each pixel with the minimum value within its local neighborhood. This filter is useful for tasks such as morphological operations and image contrast enhancement.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
minmat = ordfilt2(a,1,ones(3,3));
subplot(1,2,2);
imshow(minmat);
title("Minimum spatial filter");
```

Output

2. Median

Replaces each pixel in the image with the median value within its local neighborhood. This filter is effective in removing impulse noise, such as salt-and-pepper noise, while preserving edges.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
medianmat = ordfilt2(a,5,ones(3,3));
subplot(1,2,2);
imshow(medianmat);
title("Median spatial filter");
```

Output

3. Maximum

Replaces each pixel with the maximum value within its local neighborhood. This filter is useful for tasks such as morphological operations and image contrast enhancement.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
maxmat = ordfilt2(a,9,ones(3,3));
subplot(1,2,2);
imshow(maxmat);
title("Maximum spatial filter");
```

Output

Linear smoothing spatial filters

Linear smoothing spatial filters are used to blur or smooth an image by reducing high-frequency noise and suppressing details. These filters operate on the principle of convolution, where a kernel (also known as a filter or mask) is applied to each pixel in the image to obtain a weighted average of its neighboring pixel values. The weights in the kernel determine the influence of each neighboring pixel on the output.

1. Mean

Mean filter replaces each pixel with the average value of its local neighborhood. The neighborhood is typically a square or rectangular region. It smoothens the image by reducing intensity variations.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
filter = ones(3)/9;
avg = imfilter(a, filter, 'conv');
subplot(1,2,2);
imshow(avg);
title("Average spatial filter");
```

Output

2. Low pass

Low-pass filters allow low-frequency components (smooth variations) to pass through while attenuating high-frequency components. Gaussian filter is a common type of low-pass filter. Smoothing with an emphasis on preserving low-frequency details.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
filter = [1 1 1; 1 2 1; 1 1 1]/10;
lowavg = imfilter(a, filter, 'conv');
subplot(1,2,2);
imshow(lowavg);
title("Low pass spatial filter");
```

Output

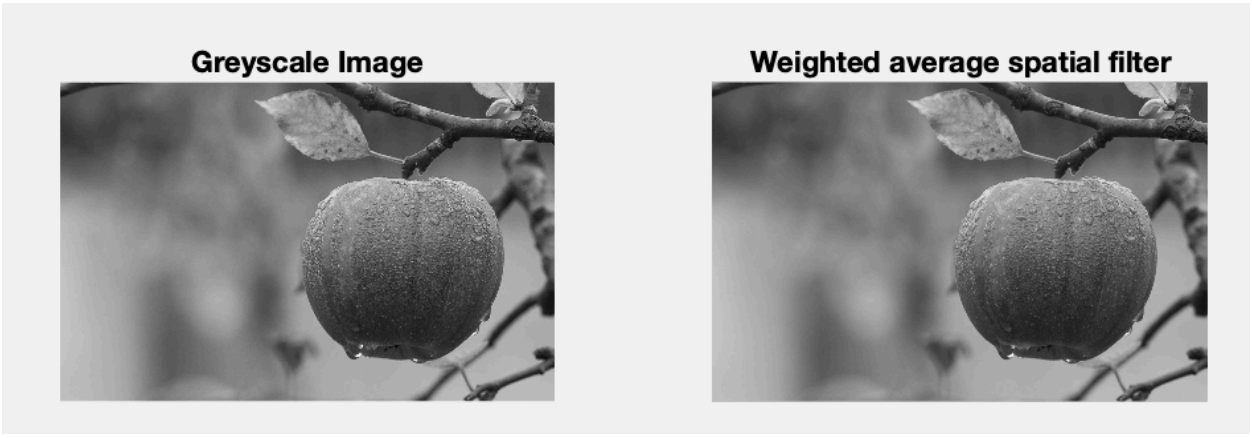
3. Weighted average

Similar to mean filtering, but with different weights assigned to each pixel in the neighborhood. The weights are determined by a predefined kernel. It allows for different levels of emphasis on each pixel in the neighborhood, providing more flexibility in image smoothing.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
filter = [1 2 1; 2 4 2; 1 2 1]/16;
weghavg = imfilter(a, filter, 'conv');
subplot(1,2,2);
imshow(weghavg);
title("Weighted average spatial filter");
```

Output



Conclusion: By applying various filters, we understood how spatial smoothing techniques work on an image in MATLAB.

Image Processing and Pattern recognition

Week 4

Aim: Understanding image sharpening techniques (1st order) - Robert's cross, Sobel's, Prewitt's filters using MATLAB.

Requirements: MATLAB software

Directory used throughout week 4 -
/Users/harthikmallichetty/Downloads/grayscale.jpeg

Image used to demonstrate transforms throughout week 4 - (grayscale.jpeg)

Value - 1204 x 1880 uint8

Sharpening Filters

In image processing, sharpening filters are used to enhance the fine details and edges within an image, making it appear more focused and clear. These filters work by emphasizing high-frequency components of the image, which often correspond to edges or rapid changes in intensity.

1. Robert's cross

The Roberts Cross operator is a simple edge detection algorithm used in image processing and computer vision. The operator works by performing a convolution of the image with a pair of 2x2 convolution kernels. These kernels for Gx and Gy are:

1	0
0	-1

0	1
-1	0

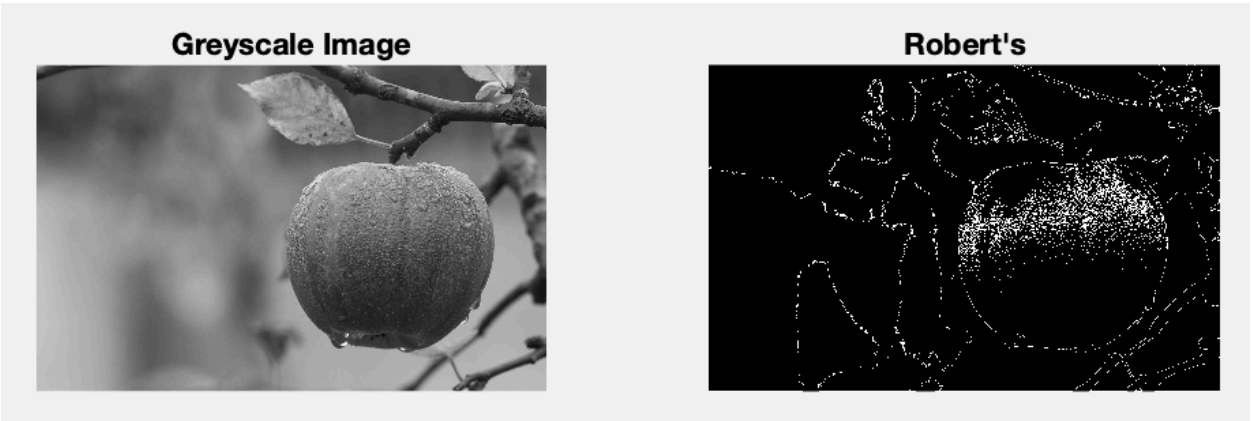
After using these filters to convolve, we use |G| = |Gx| +|Gy| to get the image.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r = 128;
thres = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if a(j,k) >= r
        thres(j,k) = L-1;
     else
        thres(j,k) = 0;
     end
  end
end
roberts_x = [1 0;0 -1];
roberts_y = [0 1;-1 0];
Gx = imfilter(thres, roberts_x, 'conv');
Gy = imfilter(thres, roberts_y, 'conv');
```

```
edge_rob = abs(Gx) + abs(Gy);
subplot(1,2,2);
imshow(edge_rob);
title("Robert's");
```

Output



2. Sobel's

Sobel operator is a linear filter that is frequently used for edge detection. By applying this operator, you can highlight the edges in different directions (horizontal, vertical, or diagonal), providing a sharpening effect. The kernels for Gx and Gy are:

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

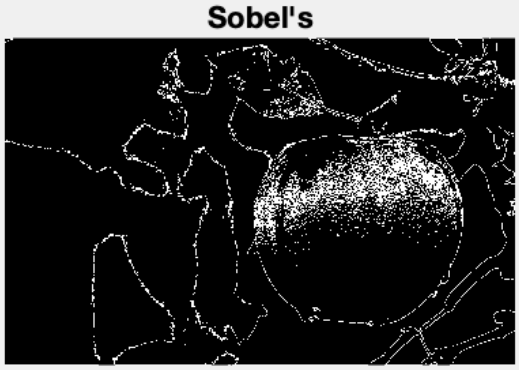
After convolving the filters over the image we use |G| = |Gx| + |Gy|.

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r = 128;
thres = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if a(j,k) >= r
        thres(j,k) = L-1;
     else
        thres(j,k) = 0;
     end
  end
end
sobel_x = [-1 0 1; -2 0 2; -1 0 1];
sobel_y = [1 2 1; 0 0 0; -1 -2 -1];
```

```
x = imfilter(thres, sobel_x, 'conv');
y = imfilter(thres, sobel_y, 'conv');
edge_sob = abs(x) + abs(y);
subplot(1,2,2);
imshow(edge_sob);
title("Sobel's");
```

Output



3. Prewitt's

Prewitt operator is a linear filter that is frequently used for edge detection. By applying this operator, you can highlight the edges in different directions (horizontal, vertical, or diagonal), providing a sharpening effect. The kernels for Gx and Gy are:

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

After convolving the filters over the image we use |G| = |Gx| + |Gy|.

1	1	1
0	0	0
-1	-1	-1

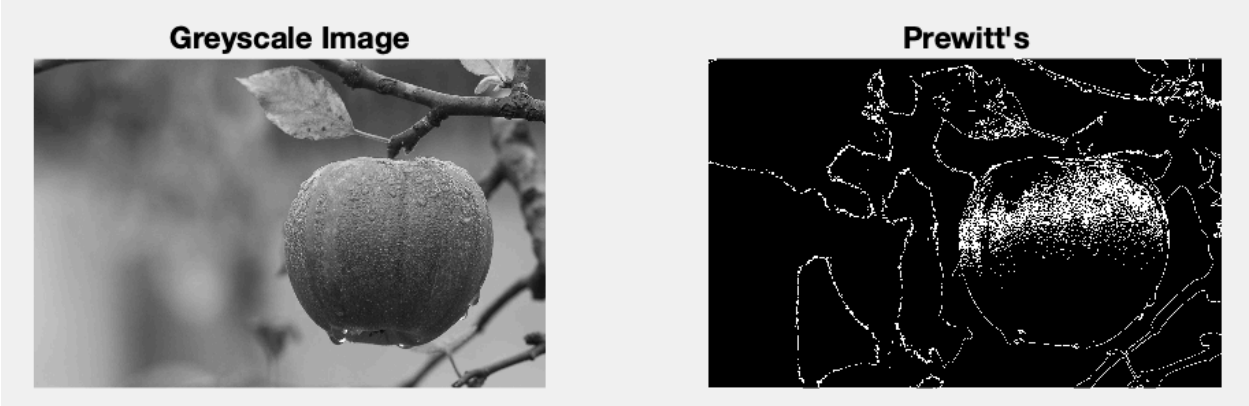
-1	0	1
-1	0	1
-1	0	1

Code

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r = 128;
thres = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if a(j,k) >= r
        thres(j,k) = L-1;
     else
```

```
        thres(j,k) = 0;
      end
    end
end
prewitt_x = [-1 0 1; -2 0 2; -1 0 1];
prewitt_y = [1 2 1; 0 0 0; -1 -2 -1];
x = imfilter(thres, prewitt_x, 'conv');
y = imfilter(thres, prewitt_y, 'conv');
edge_prew = abs(x) + abs(y);
subplot(1,2,2);
imshow(edge_prew);
title("Prewitt's");
```

Output



Conclusion: By applying various filters, we understood how spatial sharpening techniques work on an image in MATLAB.

Image Processing and Pattern recognition

Week 5

Aim: Understanding image sharpening technique (2nd order) - Laplacian filter using MATLAB.

Requirements: MATLAB software

Directory used throughout week 5 - /Users/harthikmallichetty/Downloads/grayscale.jpeg

Image used to demonstrate transforms throughout week 5 - (grayscale.jpeg)

Value - 1204 x 1880 uint8

Sharpening Filters

In image processing, sharpening filters are used to enhance the fine details and edges within an image, making it appear more focused and clear. These filters work by emphasizing high-frequency components of the image, which often correspond to edges or rapid changes in intensity.

Laplacian

A Laplacian filter, also known as a Laplacian operator or Laplacian of Gaussian (LoG), is a mathematical operator used in image processing for edge detection and image sharpening. It is particularly effective in highlighting rapid intensity changes or edges in an image. The Laplacian operator is defined as the sum of the second derivatives of the image with respect to the spatial coordinates. In discrete form, it can be approximated using a convolution operation with a kernel or filter. The Laplacian operator is often represented by the following kernels:

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

-1	2	-1
2	-4	2
-1	2	-1

$$\nabla^2 f(x,y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

Code - Filter 1

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r = 140;
thres = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if a(j,k) >= r
        thres(j,k) = L-1;
```

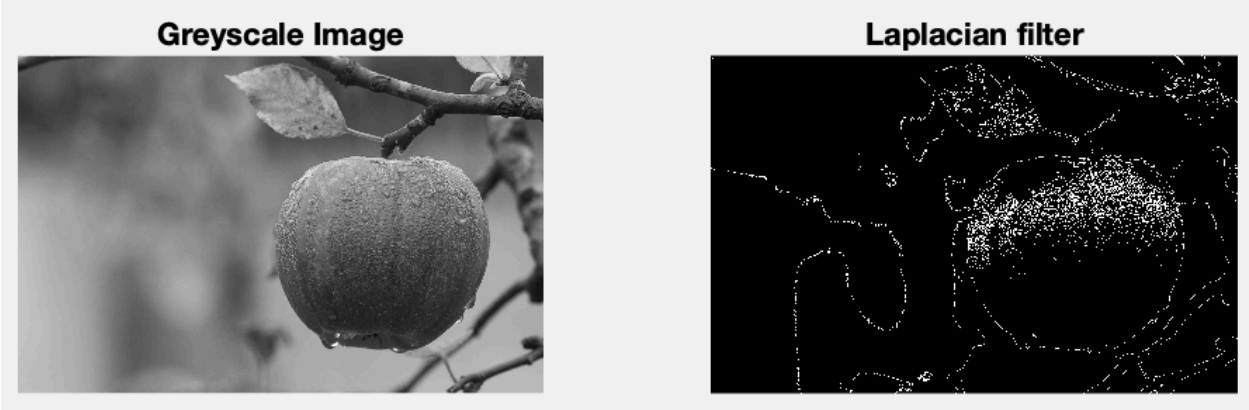
```
        else
            thres(j,k) = 0;
        end
    end
end
filter = [0 1 0; 1 -4 1; 0 1 0];
threshed = imfilter(thres, filter, 'conv');
subplot(1,2,2);
imshow(threshed);
title("Laplacian filter");
```

Output

Code - Filter 2

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r = 140;
thres = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if a(j,k) >= r
        thres(j,k) = L-1;
     else
        thres(j,k) = 0;
     end
  end
end
filter = [1 1 1; 1 -8 1; 1 1 1];
threshed = imfilter(thres, filter, 'conv');
subplot(1,2,2);
imshow(threshed);
title("Laplacian filter");
```

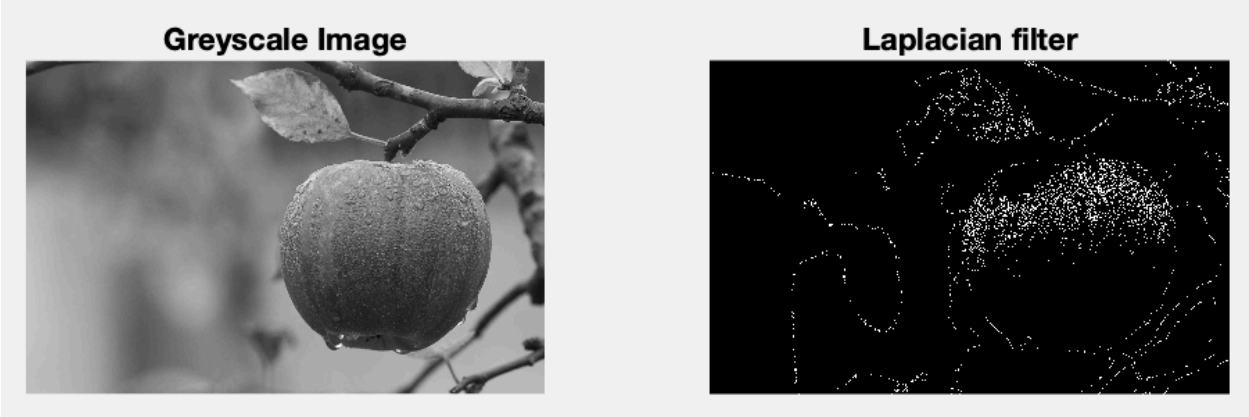
Output



Code - Filter 3

```
a = imread("/Users/harthikmallichetty/Downloads/grayscale.jpeg");
subplot(1,2,1);
imshow(a);
title("Greyscale Image");
[x,y] = size(a);
r = 140;
thres = zeros(x,y);
maxpix = max(max(a));
for i = 1:32
  if maxpix<(2^i)
     L=2^i;
     break;
  else
     i=i+1;
  end
end
for j = 1:x
  for k = 1:y
     if a(j,k) >= r
        thres(j,k) = L-1;
     else
        thres(j,k) = 0;
     end
  end
end
filter = [-1 2 -1; 2 -4 2; -1 2 -1];
threshed = imfilter(thres, filter, 'conv');
subplot(1,2,2);
imshow(threshed);
title("Laplacian filter");
```

Output



Conclusion: By applying the Laplacian filters, we understood how spatial sharpening technique works for 2nd order filtering on an image using MATLAB.