Assignment 2

Neeraj Kumar (2020BSZ8607) bsz208607@iitd.ac.in
Krishna Chaitanya Reddy Tamataam(2018MT60785) mt6180785@maths.iitd.ac.in
Chayan Kumar Paul(2020EEZ7528) eez207528.@iitd.ac.in
IIT Delhi

15th, Septemeber 2020

1 Question 1

1.1 Adding Noise

1.1.1 MATLAB code

```
I = imread('lena512-2.bmp');
a = imresize(I,[100,100]);
out = uint8(awgn(double(a) , 0 , 'measured'));
d = 1/(1+(10^(0/10)));
out_0 =  imnoise(out,'salt & pepper',d);

out = uint8(awgn(double(a) , 10 , 'measured'));
d = 1/(1+(10^(10/10)));
out_10 =  imnoise(out,'salt & pepper',d);

out = uint8(awgn(double(a) , 20 , 'measured'));
d = 1/(1+(10^(20/10)));
out_20 =  imnoise(out,'salt & pepper',d);

out = uint8(awgn(double(a) , 30 , 'measured'));
d = 1/(1+(10^(30/10)));
out_30 =  imnoise(out,'salt & pepper',d);

out = uint8(awgn(double(a) , 60 , 'measured'));
d = 1/(1+(10^(60/10)));
out_60 =  imnoise(out,'salt & pepper',d);

figure
ax1 = subplot(2,3,1);
imshow(out_0)
ax2 = subplot(2,3,2);
imshow(out_10)
ax3 = subplot(2,3,3);
imshow(out_20)
ax4 = subplot(2,3,4);
imshow(out_30)
ax5 = subplot(2,3,5);
imshow(out_60)
```

1.1.2 Results

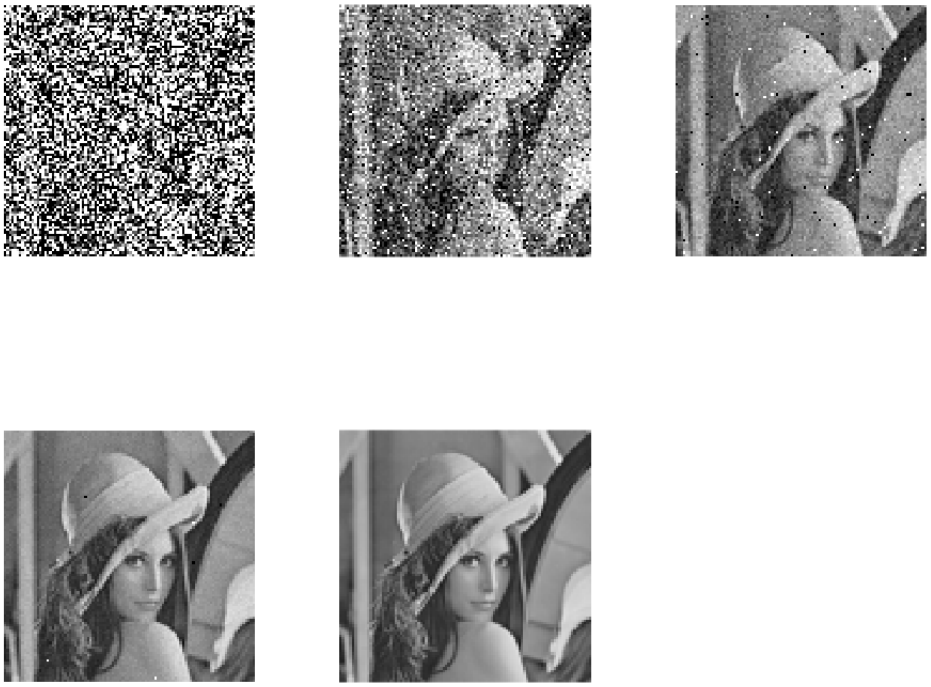

Figure 1: **Image with Gaussian noise and SaltPepper noise added with increasing SNR from left to right** TopLeft : Image with SNR = 0db or both kinds of noises,TopCentre : Image with SNR = 10db or both kinds of noises, TopRight : Image with SNR = 20db for both kinds of noises, BottomLeft : Image with SNR = 30db for both kinds of noises, BottomRight : Image with SNR = 60db for both kinds of noises

1.1.3 Observations

We used the in-bulit awgn command to add gaussian noise of the desired snr. Then we add Salt and Pepper noise of the same snr to the output image coming after adding the gaussian noise (sequentially added the two noises).But for salt and pepper noise we calculated the density of noise $d = 1/(1 + 10^{snr/10})$ where snr is measured in decibels.Increasing the snr will decrease the noise added to the image as seen from left to right in the image shown in Figure 1.

1.2 Smoothing filters

1.2.1 MATLAB code

```
 filter_5 = ones(5)/25;
 filter_7 = ones(7)/49;
 filter_9 = ones(9)/81;
 smooth5_0 = imfilter(out_0, filter_5);
 smooth7_0 = imfilter(out_0, filter_7);
 smooth9_0 = imfilter(out_0, filter_9);
%    monty = cat(3,smooth5_0,smooth7_0,smooth9_0);
%    figure
%    montage(monty)
 smooth5_10 = imfilter(out_10, filter_5);
 smooth7_10 = imfilter(out_10, filter_7);
  smooth9_10 = imfilter(out_10, filter_9);
%    monty = cat(3,smooth5_10,smooth7_10,smooth9_10);
```

```
%    figure
%    montage(monty)
 smooth5_20 = imfilter(out_20, filter_5);
 smooth7_20 = imfilter(out_20, filter_7);
 smooth9_20 = imfilter(out_20, filter_9);
%    monty = cat(3,smooth5_20,smooth7_20,smooth9_20);
%    figure
%    montage(monty)
 smooth5_30 = imfilter(out_30, filter_5);
 smooth7_30 = imfilter(out_30, filter_7);
 smooth9_30 = imfilter(out_30, filter_9);
%    monty = cat(3,smooth5_30,smooth7_30,smooth9_30);
%    figure
%    montage(monty)
 smooth5_60 = imfilter(out_60, filter_5);
 smooth7_60 = imfilter(out_60, filter_7);
 smooth9_60 = imfilter(out_60, filter_9);
%    monty = cat(3,smooth5_60,smooth7_60,smooth9_60);
%    figure
%    montage(monty)

 mean5_0 = mean(mean((smooth5_0 - out_0).^2));
 mean7_0 = mean(mean((smooth7_0 - out_0).^2));
 mean9_0 =  mean(mean((smooth9_0 - out_0).^2));

 mean5_10 = mean(mean((smooth5_10 - out_10).^2));
 mean7_10 = mean(mean((smooth7_10 - out_10).^2));
 mean9_10 =  mean(mean((smooth9_10 - out_10).^2));

 mean5_20 = mean(mean((smooth5_20 - out_20).^2));
 mean7_20 = mean(mean((smooth7_20 - out_20).^2));
 mean9_20 =  mean(mean((smooth9_20 - out_20).^2));

 mean5_30 = mean(mean((smooth5_30 - out_30).^2));
 mean7_30 = mean(mean((smooth7_30 - out_30).^2));
 mean9_30 =  mean(mean((smooth9_30 - out_30).^2));

 mean5_60 = mean(mean((smooth5_60 - out_60).^2));
 mean7_60 = mean(mean((smooth7_60 - out_60).^2));
 mean9_60 =  mean(mean((smooth9_60 - out_60).^2));

X =[0 , 10 , 20 , 30 , 60];
Y = [mean5_0,mean7_0,mean9_0;
     mean5_10,mean7_10,mean9_10;
     mean5_20,mean7_20,mean9_20;
     mean5_30,mean7_30,mean9_30;
     mean5_60,mean7_60,mean9_60];
figure
bar(X,Y)
```

1.2.2 Result

1.2.3 Observations

in Figure 2, We can see that for a more noised image the mean square errors were almost the same for all the three filters. but for a less noise image the mean squared error increased with the filter size. also we could observe a steady decrease of mean square error with increase in snr.

1.3 Edge Detection using Canny Edge Filter

1.3.1 MATLAB code

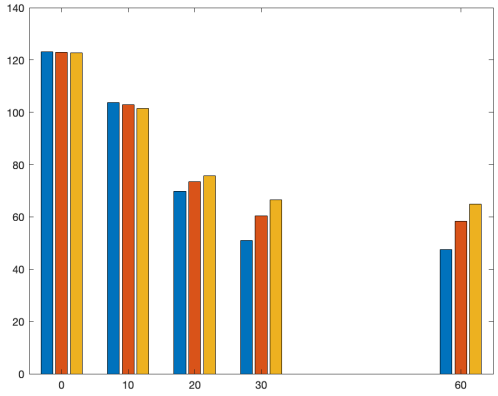

Figure 2: **A bar graph showing the between the mean square error between the noised image and smoothed image for 5*5(blue) , 7*7(red) , 9*9(orange) filter**

```
BW_0 = edge(out_0,'canny');
BW_10 = edge(out_10,'canny');
BW_20 = edge(out_20,'canny');
BW_30 = edge(out_30,'canny');
BW_60 = edge(out_60,'canny');
figure
ax1 = subplot(2,3,1);
imshow(BW_0)
ax2 = subplot(2,3,2);
imshow(BW_10)
ax3 = subplot(2,3,3);
imshow(BW_20)
ax4 = subplot(2,3,4);
imshow(BW_30)
ax5 = subplot(2,3,5);
imshow(BW_60)
```

1.3.2 Results

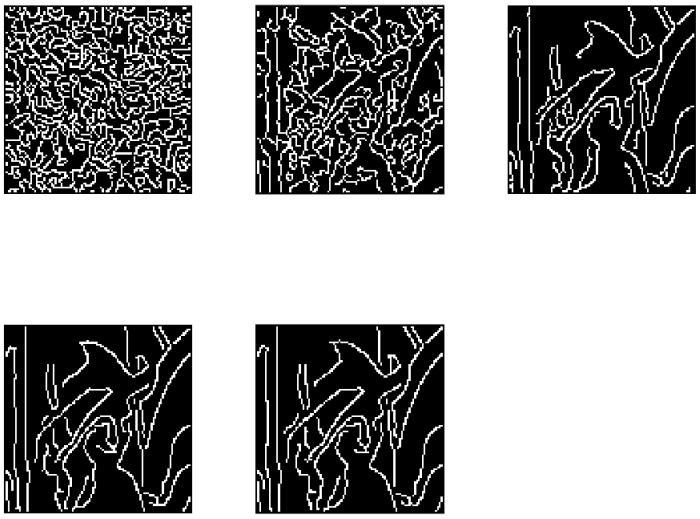

Figure 3: **Canny Edge detector applied on images with different noise Level (0db for both kinds of noises , 10db for both kinds of noises, 20db for both kinds of noises , 30db for both kinds of noises, 60db for both kinds of noises)**

1.3.3 Observations

In Figure 3 We can see the canny edge detector is forming edges around the noise in the left most images so and we are getting better sense of edges as the noise level decreases from left to right. Next we try to use the edge detector after removing the noise.

1.4 Applying noise removal filters and then applying canny edge filter

1.4.1 MATLAB code

```
x = 3;
y = 3;
eps = 1e-4;
har_0 = uint8((x*y)./imfilter(1./double(out_0+eps),ones(x,y)));
har_10 = uint8((x*y)./imfilter(1./double(out_10+eps),ones(x,y)));
har_20 = uint8((x*y)./imfilter(1./double(out_20+eps),ones(x,y)));
har_30 = uint8((x*y)./imfilter(1./double(out_30+eps),ones(x,y)));
har_60 = uint8((x*y)./imfilter(1./double(out_60+eps),ones(x,y)));
figure
ax1 = subplot(2,3,1);
imshow(edge(har_0,'canny'))
ax2 = subplot(2,3,2);
imshow(edge(har_10,'canny'))
ax3 = subplot(2,3,3);
imshow(edge(har_20,'canny'))
ax4 = subplot(2,3,4);
imshow(edge(har_30,'canny'))
ax5 = subplot(2,3,5);
imshow(edge(har_60,'canny'))
%median Filter
figure
ax11 = subplot(2,3,1);
imshow(medfilt2(out_0))
```

```
ax21 = subplot(2,3,2);
imshow(medfilt2(out_10))
ax31 = subplot(2,3,3);
imshow(medfilt2(out_20))
ax41 = subplot(2,3,4);
imshow(medfilt2(out_30))
ax51 = subplot(2,3,5);
imshow(medfilt2(out_60))

figure
ax1 = subplot(2,3,1);
imshow(edge(medfilt2(out_0),'canny'))
ax2 = subplot(2,3,2);
imshow(edge(medfilt2(out_10),'canny'))
ax3 = subplot(2,3,3);
imshow(edge(medfilt2(out_20),'canny'))
ax4 = subplot(2,3,4);
imshow(edge(medfilt2(out_30),'canny'))
ax5 = subplot(2,3,5);
imshow(edge(medfilt2(out_60),'canny'))
%adaptive median filter code

function b = adaptive_filter(image)
b = image;
Smax=9;
% 100*100 image
for i=1:98
    for j=1:98
        n=b(i:i+2,j:j+2);
        Zmin=min(n(:));
        Zmax=max(n(:));
        Zmed=median(n(:));
        sx=3;
        sy=3;
        % check if pixel value is extreme
        A1=Zmed-Zmin;
        A2=Zmed-Zmax;
        if (A1>0) && (A2<0)
          B1 = Zxy-Zmin;
          B2 = Zxy-Zmax;
          if (B1>0) && (B2<0)
                b(i:i+2,j:j+2) = n(i, j);
              break;
          else
              % if yes replace by median
               b(i:i+2,j:j+2) = Zmed;
              break;
          end
        else
          % increase the size of filter if Zmiin < Zmed < Zmax
          sx=sx+2;
          sy=sy+2;
          if (sx > Smax) && (sy > Smax)
              b(i:i+2,j:j+2) = n(i, j);
          end
        end
    end
end
end
%applying adaptive filtering on image
figure
```

```
ax11 = subplot(2,3,1);
imshow(adaptive_filter(out_0))
ax21 = subplot(2,3,2);
imshow(adaptive_filter(out_10))
ax31 = subplot(2,3,3);
imshow(adaptive_filter(out_20))
ax41 = subplot(2,3,4);
imshow(adaptive_filter(out_30))
ax51 = subplot(2,3,5);
imshow(adaptive_filter(out_60))

figure
ax1 = subplot(2,3,1);
imshow(edge(adaptive_filter(out_0),'canny'))
ax2 = subplot(2,3,2);
imshow(edge(adaptive_filter(out_10),'canny'))
ax3 = subplot(2,3,3);
imshow(edge(adaptive_filter(out_20),'canny'))
ax4 = subplot(2,3,4);
imshow(edge(adaptive_filter(out_30),'canny'))
ax5 = subplot(2,3,5);
imshow(edge(adaptive_filter(out_60),'canny'))
```

1.4.2 Results

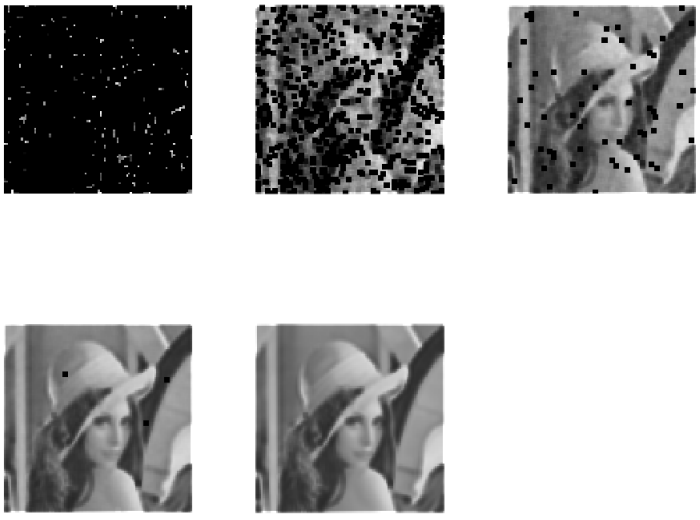

Figure 4: **Applying Harmonic Mean Filter(0db for both kinds of noises , 10db for both kinds of noises, 20db for both kinds of noises , 30db for both kinds of noises, 60db for both kinds of noises)**

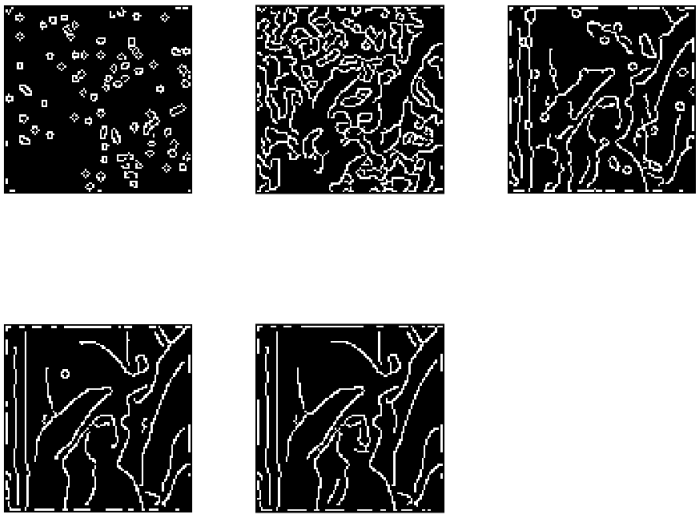

Figure 5: **Canny Edge detector after applying harmonic mean filter(0db for both kinds of noises , 10db for both kinds of noises, 20db for both kinds of noises , 30db for both kinds of noises, 60db for both kinds of noises)**

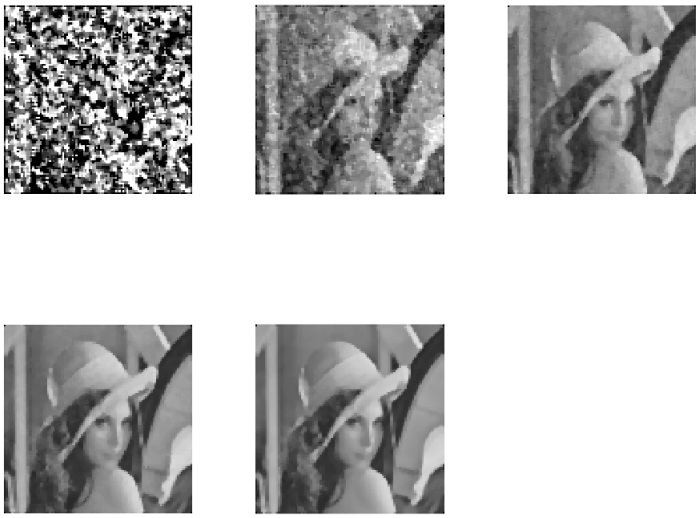

Figure 6: **Applying Median Filter (0db for both kinds of noises , 10db for both kinds of noises, 20db for both kinds of noises , 30db for both kinds of noises, 60db for both kinds of noises)**

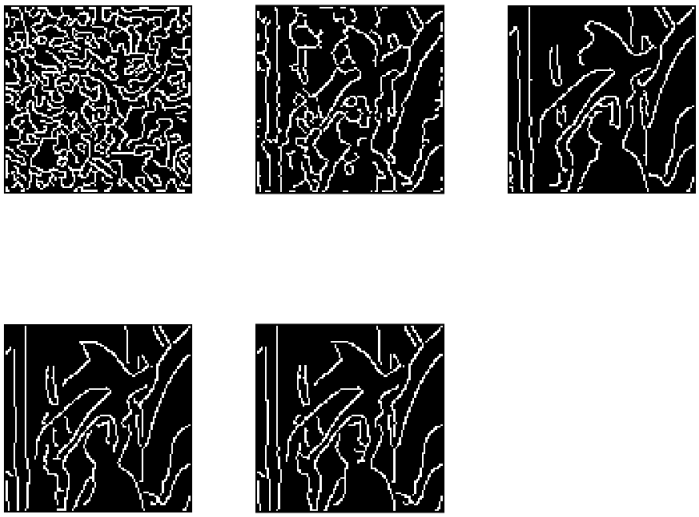

Figure 7: **Canny Edge Detector after applying Median Filter(0db for both kinds of noises , 10db for both kinds of noises, 20db for both kinds of noises , 30db for both kinds of noises, 60db for both kinds of noises)**

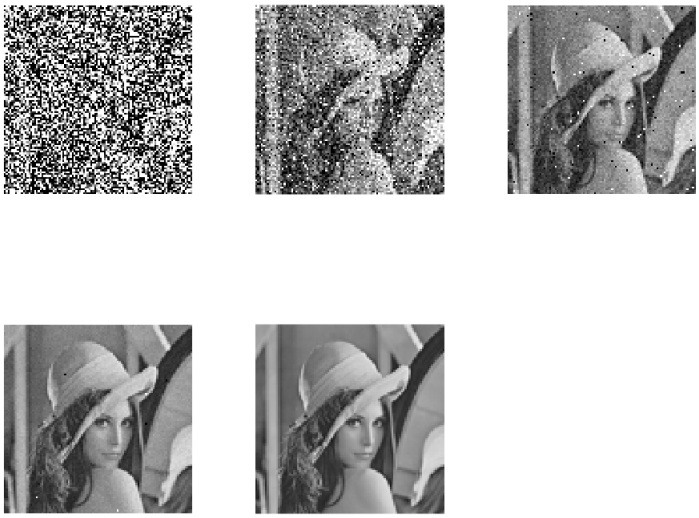

Figure 8: **Applying Adaptive Median Filter(0db for both kinds of noises , 10db for both kinds of noises, 20db for both kinds of noises , 30db for both kinds of noises, 60db for both kinds of noises)**

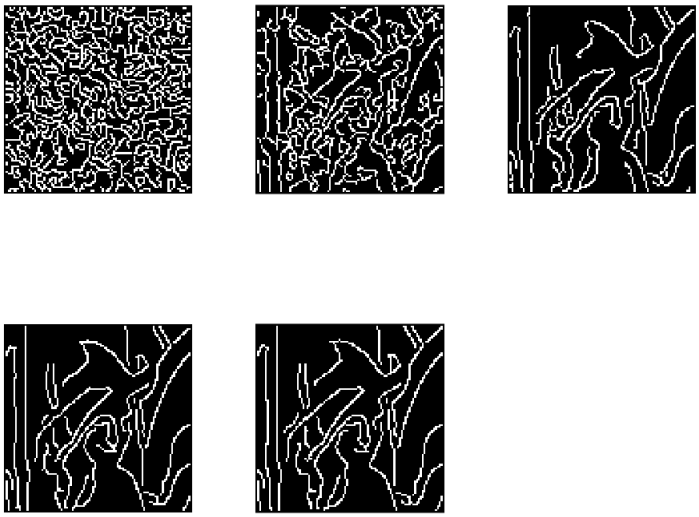

Figure 9: **Applying Canny Edge detector after applying adaptive median filter (0db for both kinds of noises , 10db for both kinds of noises, 20db for both kinds of noises , 30db for both kinds of noises, 60db for both kinds of noises)**

1.4.3 Observation

- In Figure 4 and 5 and Firstly the harmonic filter was able to remove the Gaussian noise but the salt and pepper got worsened into black spots in the image and hence the edge detector worked poorly.
- In Figure 6 and 7 Secondly the median filter is good at removing the salt and pepper low spatial noise density but not so effective in removing it for a high spatial noise density. but still performed a better job in removing the salt and pepper noise as compared to harmonic filter. It is easy to form edges by canny detector if the image has Gaussian noise as there is a Gaussian filter applied to remove the Gaussian noise.
- In Figure 8 and 9 ,Adaptive median filter was designed with a maximum window size of 9. This algorithm has two stages A where the maximum median and minimum is calculated and the window size is adjusted(max = 9) until the $minimum < median < maximum$ and is replaced by the median value if the window size reaches its maximum value. then the algorithm goes to the next stage where if the $minimum < I(x, Y) < maximum$ then the pixel value remains the same else is replaced by the median value.
- In Figure 8 and 9 Adaptive median filter is able to remove more Salt and Pepper noise with high spatial density and also makes the image less distorted.with more noise being able to be moved canny edge detector is left with some gaussian noise to remove(which is inbuilt in it) and then detect the edges. we could observe that after applying adaptive median filter detects the edges better compared to the other filters.

2 Question 2

2.1 Laplace filter

2.1.1 Python Code

```
import cv2
import numpy as np

# load the image and scale to 0..1
image = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE).astype(float) / 255.0
```

```
plt.imshow(image,cmap='gray')

# vertical edge detector without doagonal element
kernel_laplace_Without_diagonal = np.array([[0, 0, 0],
                    [1, -2, 1],
                    [0, 0, 0]])
filtered = cv2.filter2D(src=image, kernel=kernel_laplace_Without_diagonal,
    ddepth=-1)
filtered = (filtered * 255).astype("uint8")


# horizontal edge detector without diaginal element
kernel_laplace_Without_diagonal = np.array([[0, 1, 0],
                    [0, -2, 0],
                    [0, 1, 0]])
filtered = cv2.filter2D(src=image, kernel=kernel_laplace_Without_diagonal,
    ddepth=-1)
filtered = (filtered * 255).astype("uint8")


# vertical edge detector without diaginal element
kernel_laplace_Without_diagonal = np.array([[0, 1, 0],
                    [0, -2, 0],
                    [0, 1, 0]])
filtered = cv2.filter2D(src=image, kernel=kernel_laplace_Without_diagonal,
    ddepth=-1)
filtered = (filtered * 255).astype("uint8")
plt.imshow(filtered,cmap='gray')



# horizontal edge detector with diagonal element
kernel_laplace_Without_diagonal = np.array([[1, 1, 0],
                    [0, -4, 0],
                    [0, 1, 1]])
filtered = cv2.filter2D(src=image, kernel=kernel_laplace_Without_diagonal,
    ddepth=-1)
filtered = (filtered * 255).astype("uint8")

plt.imshow(filtered,cmap='gray')

#  edge detector without diagonal element
kernel_laplace_Without_diagonal = np.array([[0, 1, 0],
                    [1, -4, 1],
                    [0, 1, 0]])
filtered = cv2.filter2D(src=image, kernel=kernel_laplace_Without_diagonal,
    ddepth=-1)
filtered = (filtered * 255).astype("uint8")

plt.imshow(filtered,cmap='gray')

#  edge detector with diagonal element
kernel_laplace_With_diagonal = np.array([[1, 1, 1],
                    [1, -8, 1],
                    [1, 1, 1]])
filtered = cv2.filter2D(src=image, kernel=kernel_laplace_With_diagonal, ddepth
    =-1)
filtered = (filtered * 255).astype("uint8")

plt.imshow(filtered,cmap='gray')
```

2.1.2 Results

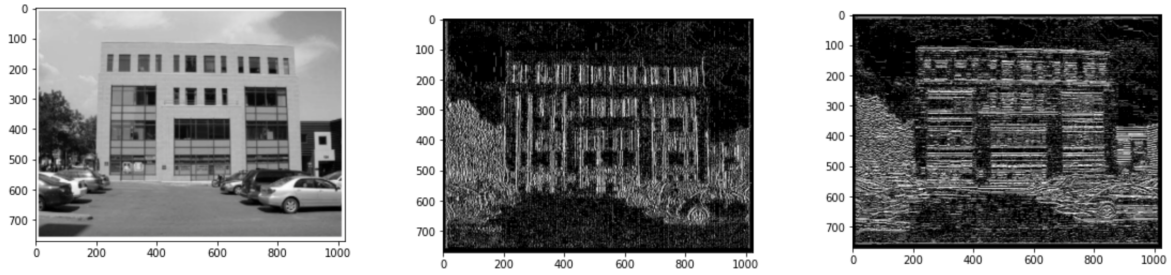

Figure 10: **Left: Original image Centre: Vertical edges without diagonal element Right :horizontal edges without diagonal element**

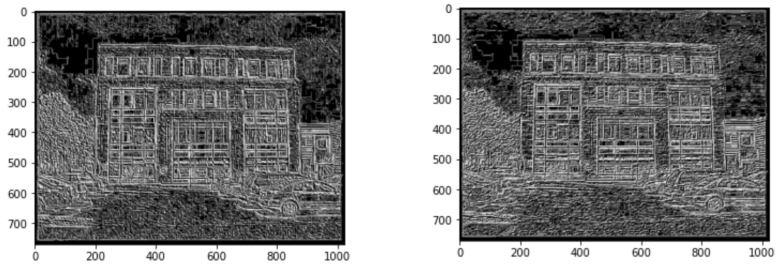

Figure 11: **Left: vertical edges with diagonal element,Left: horizontal edges with diagonal element**

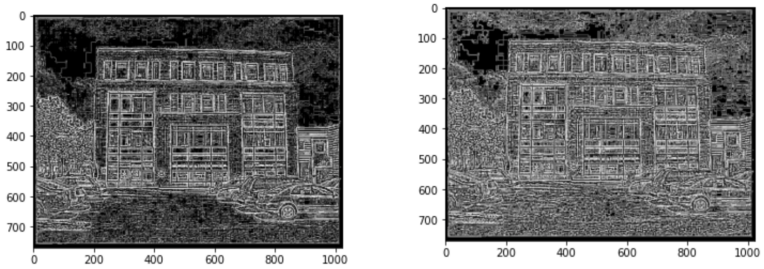

Figure 12: **Left: Joint laplacian filter without diagonal element, Right: laplacian filter with diagonal element with kernel size 3**

2.1.3 Observations

- In Figure 10 , the image and their vertical and horizontal edges are being plotted using laplcaian operator without diagonal element. The centre of image shows the prominent vertical edges and right side shows the horizontal prominent edges of the original image.
- Figure 11, thr vertical edges and horizontal edges have been p[lotted using laplacian operator having diagonal element. The edges along the diagonal elements can also be seen in the image clearly.
- Figure 12, joint laplacian filter is applied in which the Left image is having the vertical and horizontal edges without diagonal element . The right side consists of diagonal element and we can clearly see the more. o of edges and edges along diagonal directions.

2.2 Robert Operator

2.2.1 Matlab Code

```
import cv2
import numpy as np
```

```
img=cv2.imread('building.jpg',0)      #reading image using openCV

# cv2.imshow('Building',img)

plt.imshow(img,cmap='gray')

## Defining Kernel
kernelx = np.array([[1, 0], [0, -1]])
kernely = np.array([[0, 1], [-1, 0]])

#kernel with different size
# for 3X3
kernelx = np.array([[-1, 0,1], [-1,0,1],[-1,0,1]])
kernely = np.array([[1,1,1],[0,0,0], [-1,-1,-1]])

## #for 5X5 kernel size
kernelx = np.array
    ([[-2,-1,0,1,2],[-2,-1,0,1,2],[-2,-1,0,1,2],[-2,-1,0,1,2],[-2,-1,0,1,2]])
kernely = np.array
    ([[2,2,2,2,2],[1,1,1,1,1],[0,0,0,0,0],[-1,-1,-1,-1,-1],[-2,-2,-2,-2,-2]])

#Applying filter in X and Y direction
img_robertx = cv2.filter2D(img, -1, kernelx)
img_roberty = cv2.filter2D(img, -1, kernely)

#Taking the absolute value
grad = cv2.addWeighted(img_robertx, 0.5, img_roberty, 0.5, 0)

plt.imshow(grad,cmap='gray')
```

2.2.2 Results

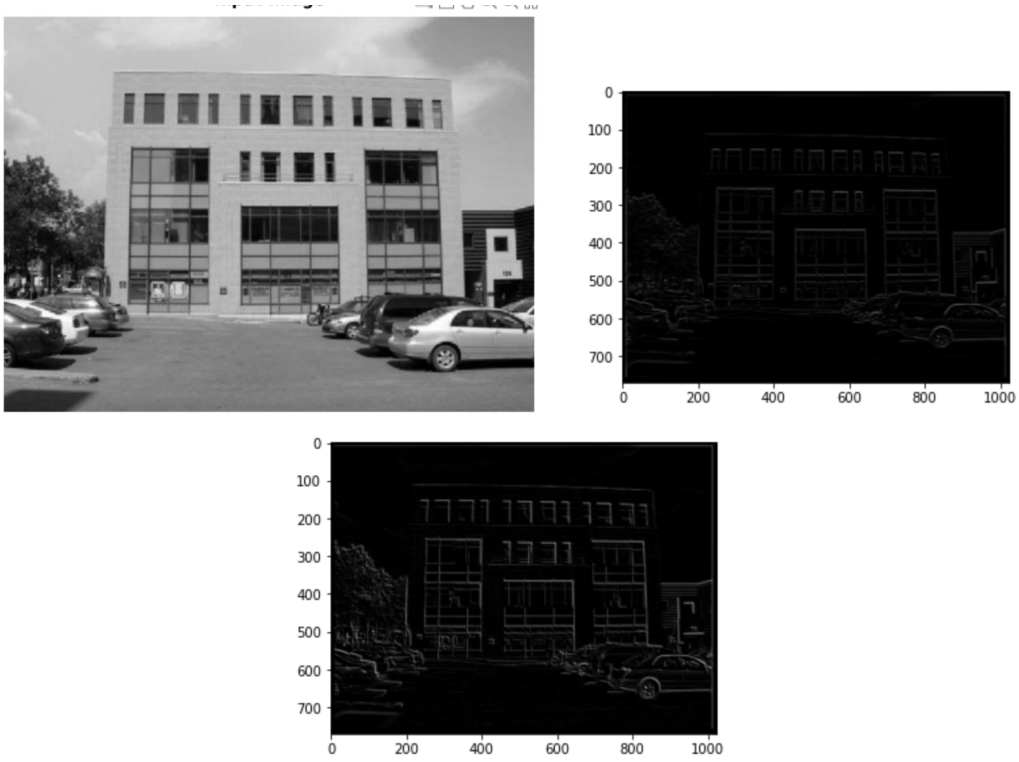

Figure 13: **Left: original image , Right: Robert operator with kernel size =2 , Bottom: Robert operator with kernel size =3**

2.2.3 Observations

- Figure 13, the robert edge operator is aplied with kernel size of 2 and 3 . In the right upper part , robert edges operator with kernel size 2 is applied and captures the very higher frequency component so the edge captured is very dim. In the bottom part , robert edge with kernel size 5 , captures the edges relatively more clearly with more higher frequency components .

2.3 Sobel Gradient operator

2.3.1 Python Code

```
img_blur = cv2.imread('building.jpg')
gray = cv2.cvtColor(img_blur, cv2.COLOR_BGR2GRAY)

sobelx = cv2.Sobel(src=gray, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=3)
sobely = cv2.Sobel(src=gray, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=3)
magnitude = np.sqrt((sobelx ** 2) + (sobely ** 2))
magnitude = np.uint8(magnitude)
```

2.3.2 Results

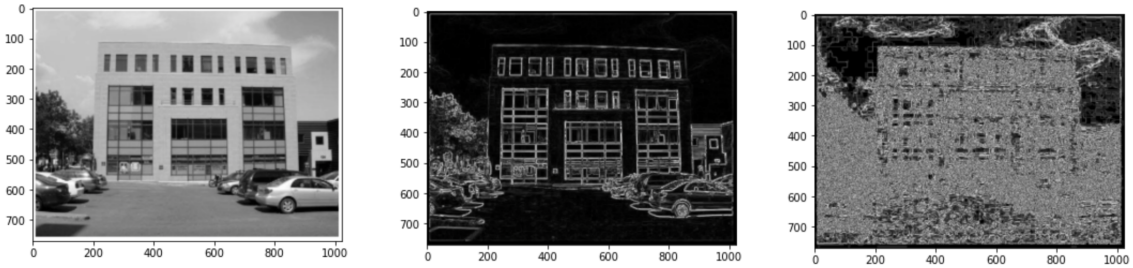

Figure 14: **Left: original image , Centre : Sobel operator edge Image with kernel size 3 ,Right : Sobel operator edge Image with kenel size =5**

2.3.3 Observations

- In Figure 14, we have plotted the edges using the sobel operator with kernel size 3 and 5 respectively. The kernel size gives better edge for the original image with kernel size 3 . With larger kernel size , edges are too many which is due to capture of more frequencies at the higher side.

2.4 High boosted filtering

2.4.1 Matlab Code

```
img = rgb2gray(imread('building.jpg'));

% cwe created a gauusian filter
hgauss = fspecial('gaussian',5,2.5);

% we blur or smoothen the image with the gaussian filter.
blur_image= imfilter(img,hgauss);

% subtract blurred image from original
diffenr_image = img - blur_image;

% add the difference to the orginal image and k is taken 4 to get the high
    boosted image.
highboost_img = img + 4*diffenr_image;
imwrite(highboost_img,'highboost_img.jpg')
```

```
#plot different image
imshow(img);
```

2.4.2 Results

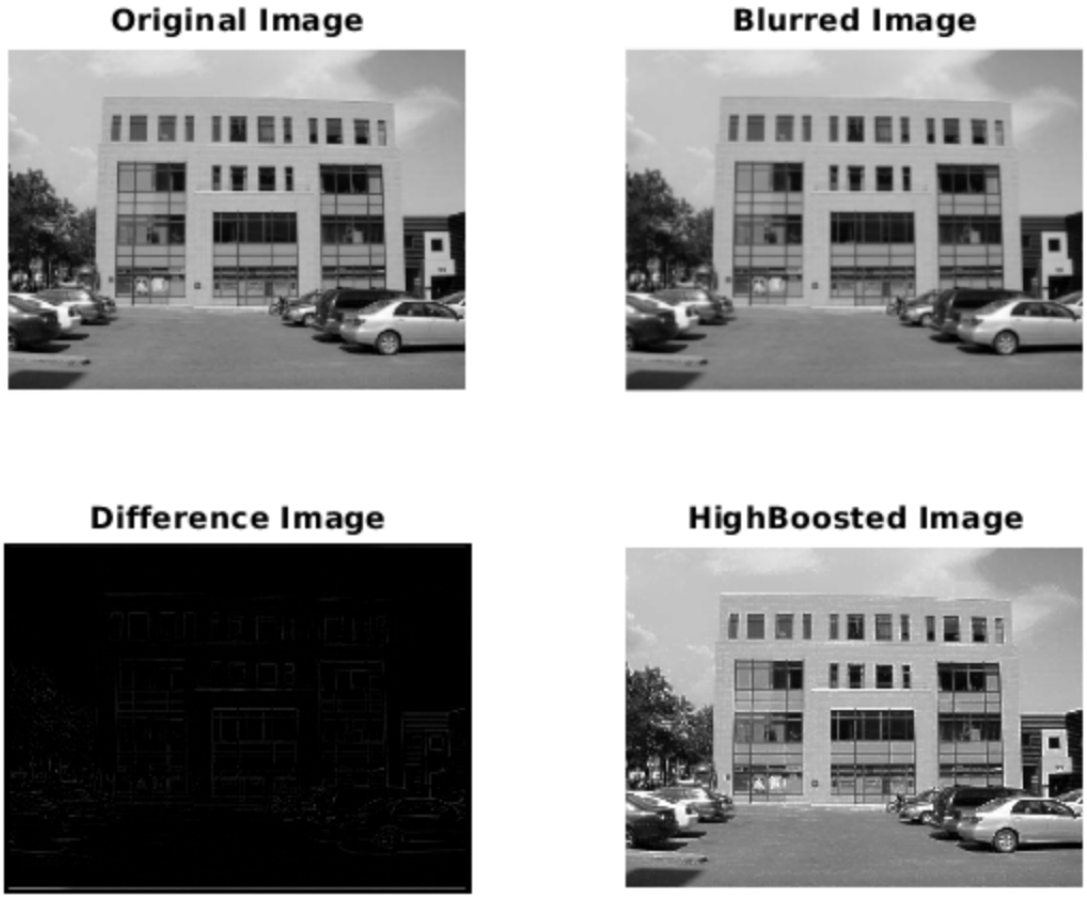

Figure 15: **original image , blurred image , difference image and high boosted image**

2.4.3 Observation

- High boosted image in Figure 15 leads to sharpening of the image . The points at which a change of slope occurs in the signal are now emphasized.

2.5 Different size operator

2.5.1 Python Code

```
#load image
img_blur = cv2.imread('building.jpg')
gray = cv2.cvtColor(img_blur, cv2.COLOR_BGR2GRAY)

#apply sobel operator
sobelx = cv2.Sobel(src=gray, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5)
sobely = cv2.Sobel(src=gray, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5)
magnitude = np.sqrt((sobelx ** 2) + (sobely ** 2))
magnitude = np.uint8(magnitude)

#plot images
```

```
plt.imshow(magnitude,cmap='gray')

#apply laplacian operator with different kernel size
s = cv2.Laplacian(gray, cv2.CV_64F, ksize=5)
s = np.uint8(s)

plt.imshow(s,cmap='gray')


# robert edge operator with different filter size
import cv2
import numpy as np

img=cv2.imread('building.jpg',0)     #reading image using openCV

# cv2.imshow('Building',img)

plt.imshow(img,cmap='gray')

# # Defining Kernel
kernelx = np.array([[1, 0], [0, -1]])
kernely = np.array([[0, 1], [-1, 0]])

#kernel with different size
# for 3X3
kernelx = np.array([[-1, 0,1], [-1,0,1],[-1,0,1]])
kernely = np.array([[1,1,1],[0,0,0], [-1,-1,-1]])

# #for 5X5
kernelx = np.array
    ([[-2,-1,0,1,2],[-2,-1,0,1,2],[-2,-1,0,1,2],[-2,-1,0,1,2],[-2,-1,0,1,2]])
kernely = np.array
    ([[2,2,2,2,2],[1,1,1,1,1],[0,0,0,0,0],[-1,-1,-1,-1,-1],[-2,-2,-2,-2,-2]])

#Applying filter in X and Y direction
img_robertx = cv2.filter2D(img, -1, kernelx)
img_roberty = cv2.filter2D(img, -1, kernely)

#Taking the absolute value
grad = cv2.addWeighted(img_robertx, 0.5, img_roberty, 0.5, 0)

plt.imshow(grad,cmap='gray')
```

2.5.2 Results

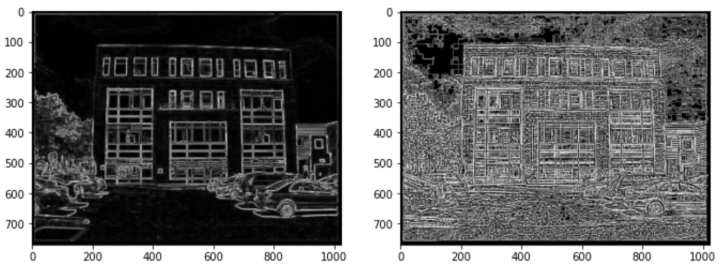

Figure 16: **Left: Sobel image with kernel size =3, Right: Laplace Image kernel size =3**

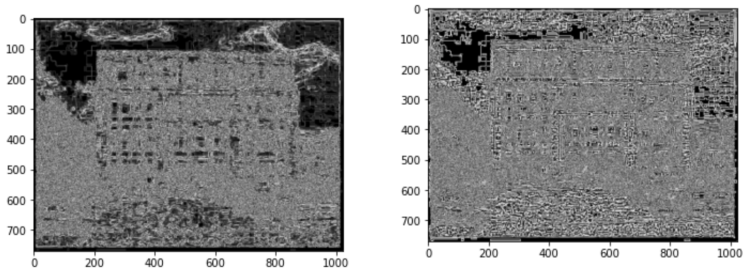

Figure 17: **Left: Sobel image with kernel size =5, Right: Laplace Image kernel size =5**

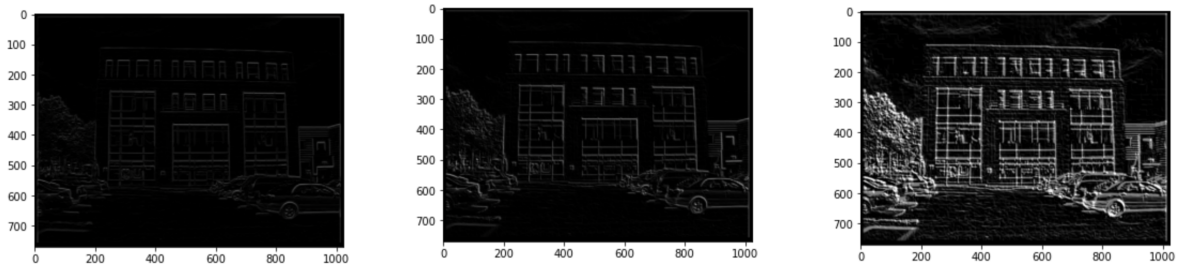

Figure 18: **Left: Robert image with kernel size =2, Centre : Robert Image kernel size =3, Right: Robert Image kernel size =5**

2.5.3 Observations

- In Figure 16 the sobel and laplcaian operator is being used with kernel size of 3. the sobel is generating the edges and in laplacian , the edges captured are more finer and in horizontal , vertical and diagonal direction.
- In Figure 17 the sobel and laplcaian operator is being used with kernel size of 5. more higher frequency components are being captured due to which we see more edges . The noises and less prominent are edges are being captured which is sometimes not necessary for edge detection.
- In Figure 18 the robert gross edge operator is being implemented with kernel size 2,3, and 5. with the increasing kernel size , the no of higher frequenct components also gets increased , so we see more no of edges int he figure.

2.6 best edge detector operator

2.6.1 Results

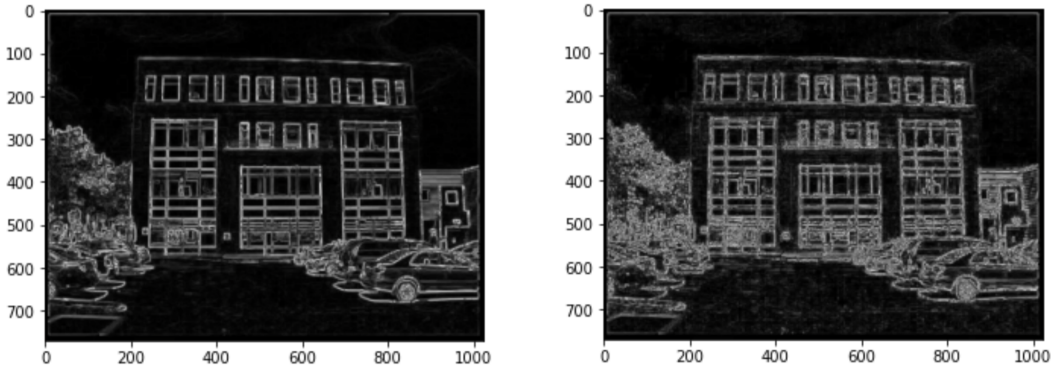

Figure 19: **Left: SObel operator in the original image , Right: Sobel operator in high boosted image**

2.6.2 Observations

- In the left part of Figure 19 , **We found the sobel operator on original image giving the best results** . The edges generated are clearly showing that noisy edges are not captured. We have also applied the sobel operartor on high boosted image in right part of Figure 19 . The noises are also being captured which can be clearly mentioned in the diagram near the car region in right part of Figure 19.

3 Question 3

3.1 Python Code

```
import cv2
import random
from matplotlib import pyplot as plt
import numpy as np
import glob
import os
import pandas as pd


# extract the frames from the video at K fps
def select_frames(video_path):
    cap = cv2.VideoCapture(video_path)

    n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = cap.get(cv2.CAP_PROP_FPS)

    # k is the frame per sec
    K = 10
    if n_frames <= K: #There are not enough frames in the video
        rand_frames_idx = [1]*n_frames
    else:
        rand_frames_idx = [0]*n_frames
        i = 0

        while(i < K):
            idx = random.randint(0, n_frames-1)
            if rand_frames_idx[idx] == 0:
                rand_frames_idx[idx] = 1
                i += 1

    frames_list = []

    # Read until video is completed or no frames needed
    ret = True
    frame_idx = 0
    while(ret and frame_idx < n_frames):

        ret, frame = cap.read()

        if ret and rand_frames_idx[frame_idx] == 1:
            RGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frames_list.append(RGB)

        frame_idx += 1

    cap.release()
```

```
    return frames_list


k=0
destin = 'data_vid/'

# loading the video and caslling the extractr frunction and saving all the
    frames in the folder
for name in glob.glob('/home/neerajku/neerajAvatar/video.mp4'):



            frames_list = select_frames(name)
            print(len(frames_list))

            for i in range(len(frames_list)):

                    name2 = destin+'frame'+format(i, '06d')+'.jpg'

                    plt.imsave(name2,frames_list[i] )




img = cv2.imread('data_vid/frame000003.jpg',0)
img1 = cv2.imread('data_vid/frame000004.jpg',0)

plt.imshow(img,cmap='gray', vmin=0, vmax=255)
plt.imshow(img1,cmap='gray', vmin=0, vmax=255)

frameDelta = cv2.absdiff(img1,img)

plt.imshow(frameDelta,cmap='gray', vmin=0, vmax=255)
```

3.2 Results

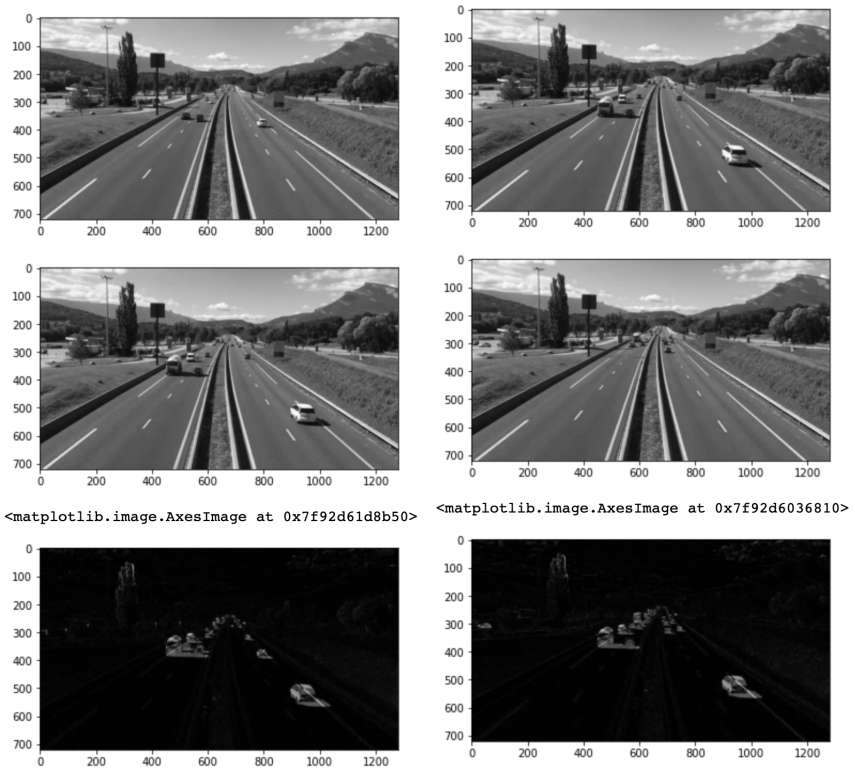

Figure 20: **Upper left : First frame of a video, Upper Right : Next frame of the video , Bottom Centre : Difference between two frame**

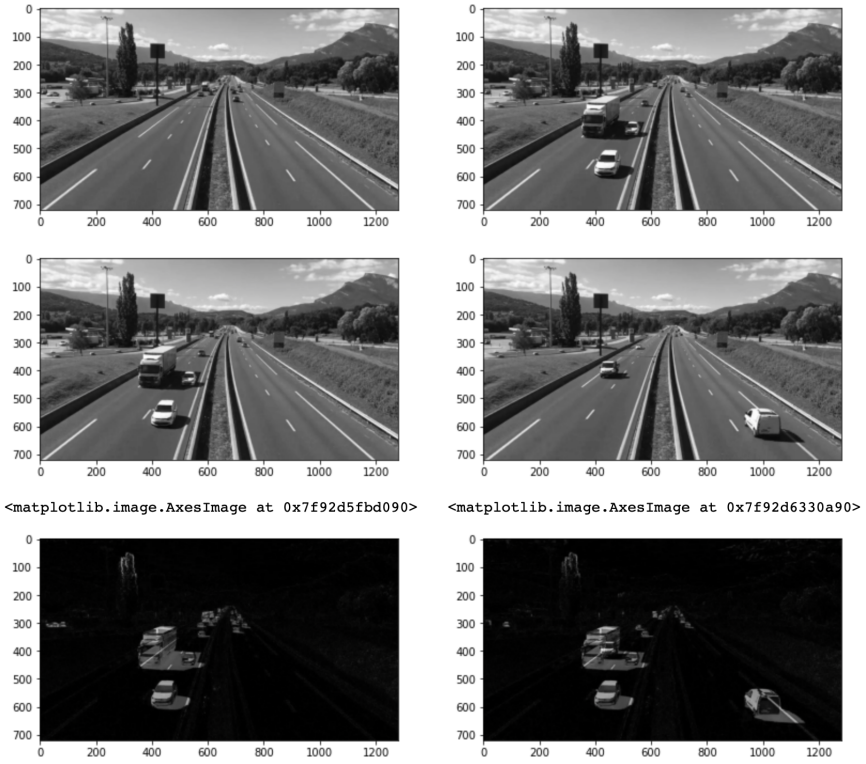

Figure 21: **Upper left : First frame of a video, Upper Right : Next frame of the video , Bottom Centre : Difference between two frame**

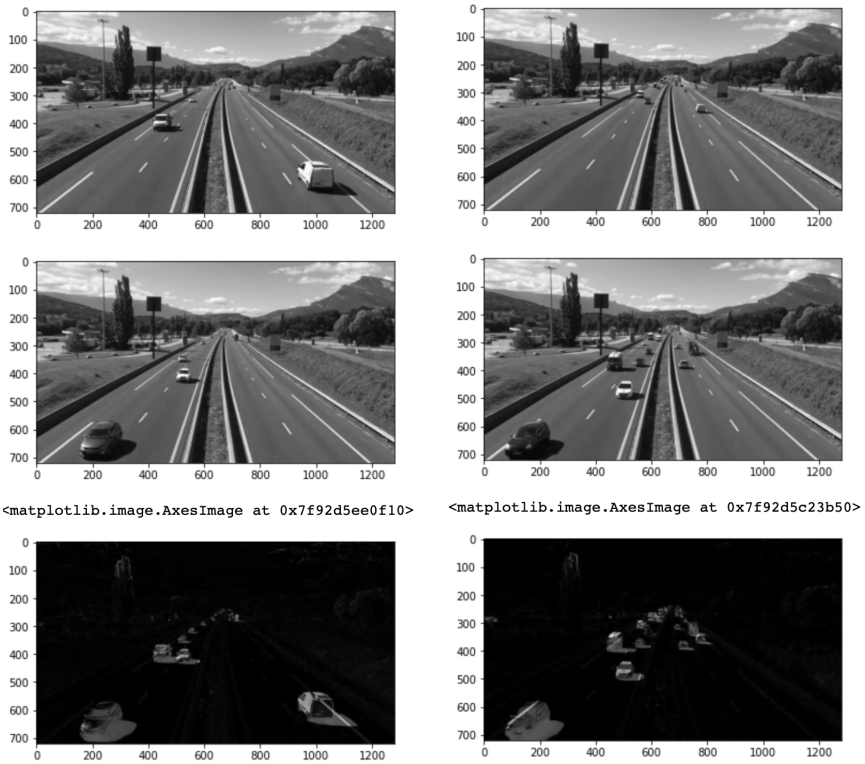

Figure 22: **Upper left : First frame of a video, Upper Right : Next frame of the video , Bottom Centre : Difference between two frame**

3.3 Observation

We have taken the video and extracted the frames at 10 frames per sec. Then the two consecutive frames are taken and difference is being captured which is shown in Figure 20, 21, 22. We have the video of less than 0.7 sec. We have extracted the 7 frames and taken the consecutive difference between first and next frame as shown ion the figure. We can see the edges near the moving objects clearly in the bottom part of the image.