East West University

Department of CSE

Lab Report 05

CSE 438

Digital Image Processing

Submitted To:

Md. Mahir Ashhab

Lecturer

Department of Computer Science and Engineering

Submitted By:

Adri Saha

ID: 2019-1-60-024

Submission Date: 28 August 2022

Report on Edge detection

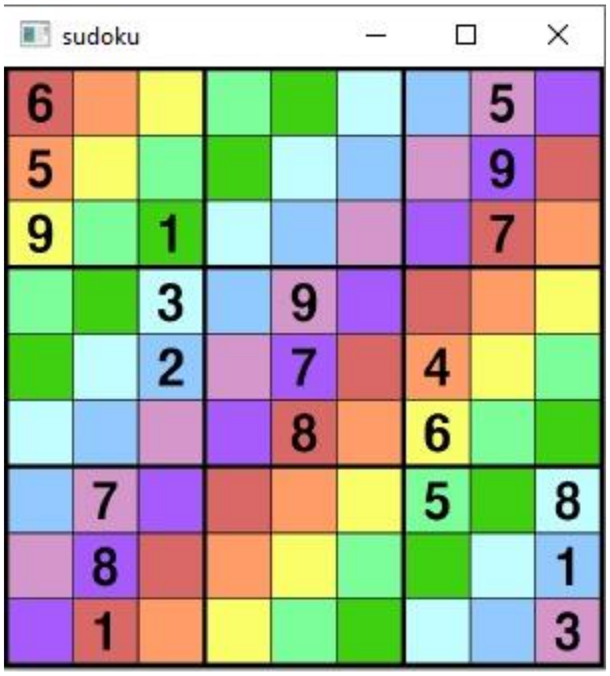

Figure 1: Sudoku image

Edge detection: One of the essential processes in image processing is edge detection which enables us to keep the structural integrity of the image while reducing the amount of data (pixels) to process.

To detect edge, we must make the image sharp. Sharpness is the stiff changes in the color plot or brightness or pixel intensity. If the gradient of a pixel is greater than its neighbor gradient, we can say specific pixel is a part of edge.

In this lab assignment, we used two edge detection techniques:

- Gradient edge detector (Sobel - first order derivatives) and
- Laplacian (2nd order derivative, therefore it is very sensitive to noise)

Both use convolutions to achieve Edge Detection, which is their common objective.

First converted the BGR image on grey image.

```
gray= cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow('sudoku in gray', gray)
```

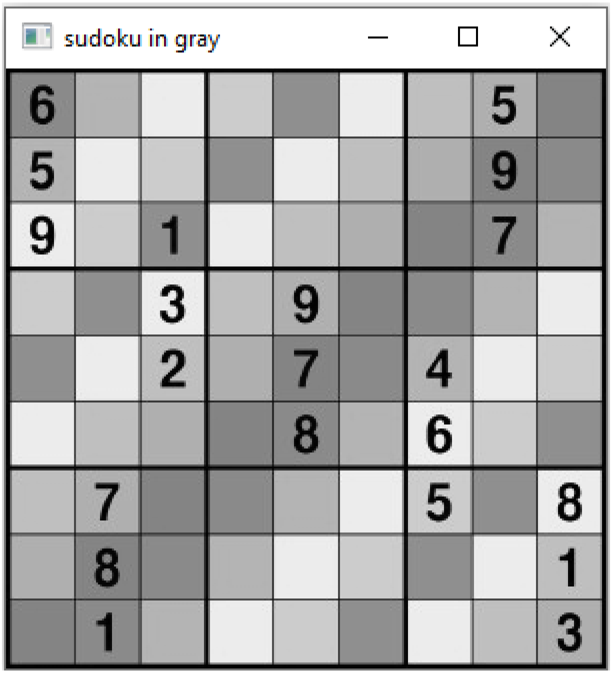

Figure 2: Gray image of sudoku

Laplacian: A 2-D isotropic measure of an image's second spatial derivative is called the Laplacian. Edge detection frequently uses the Laplacian of an image because it shows areas where there are fast changes in intensity. The Laplacian edge detector uses just one kernel, unlike the Sobel edge detector. In a single pass, it determines second order derivatives.

cv2.Laplacian(src, ddepth, other_options)

where ddepth is the desired depth of the destination image.

For high pass filter, first we used Laplacian filter.

```
laplacian= cv.Laplacian(gray, cv.CV_64F,)
cv.imshow('sudoku in laplacian', laplacian)
```

64F: It is (ddepth) depth on color channel which represents number bits for calculation or type of data. Using 64F we can show an image on 64 digits binary number (8*8 bit). Because changes can be positive, negative or fraction. We used double data type by using 64F.

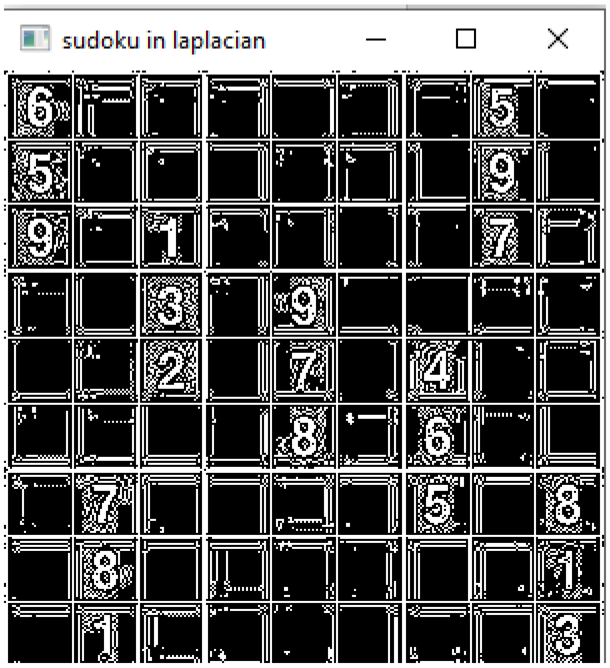

Figure 3: Sudoku image using Laplacian operator

Laplacian works with 2nd order derivatives and use 3 channel changes. The output looks like sketch on paper. Edges can be nicely identified.

```
lap= np.uint8(np.absolute(laplacian))
cv.imshow('sudoku in lap',lap)
```

Then it converted on uint8 as we converted Laplacian per bit value on absolute value.

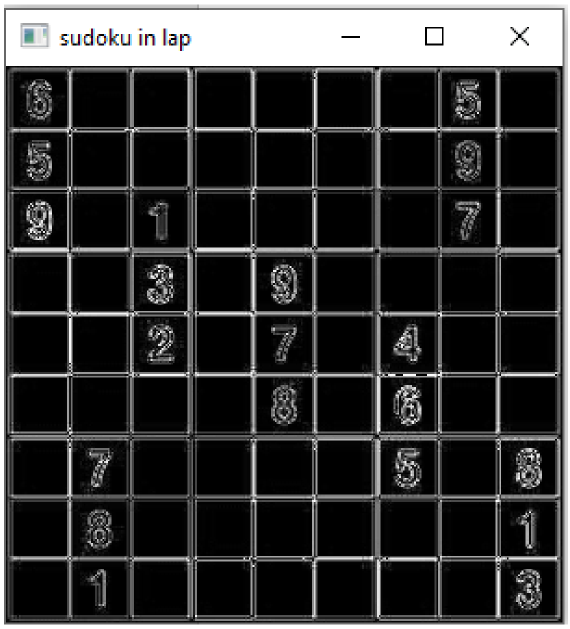

Figure 4: Sudoku image using absolute Laplacian operator

After using absolute value, we find a fine image.

Sobel Edge detection

A gradient-based technique based on the first order derivatives is the Sobel edge detector. It calculates the image's first derivatives for the X and Y axes individually.

As sobel operator works with first order derivatives, so we need to find edge on specific part like Sobel X and Sobel Y. It works for 1-bit changes. Gx moves on X-axis and pick the edges of X-axis and on a same way Gy moves on Y-axis and pick the edges of Y-axis.

Horizontal change when edge is vertical and vertical change means change in vertical axes when edge is horizontal. If we want to detect horizontal edge, we need to use gradient of vertical change which finds a horizontal edge.

```
sobelx=cv.Sobel(gray,cv.CV_64F,1,0)
```

1,0 for X axis.

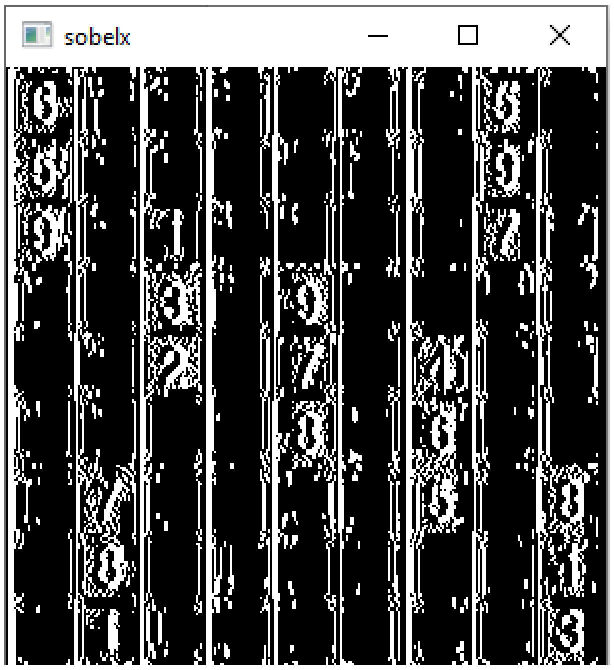

Figure 5: Sudoku image using Sobel X operator

Here we see the changes on X-axis. 0,1 for Y axis.

```
sobely=cv.Sobel(gray,cv.CV_64F,0,1)
```

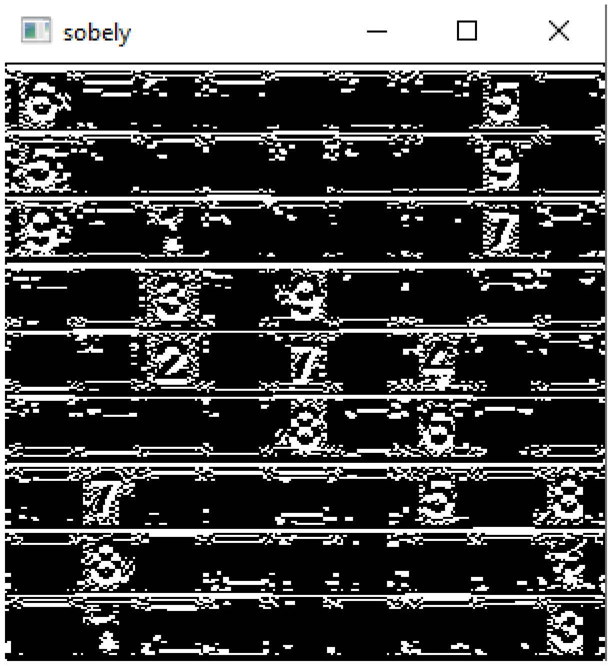

Figure 6: Sudoku image using Sobel Y operator

Here we see the changes on Y-axis.

```
combined= cv.bitwise_or(sobelx,sobely)
```

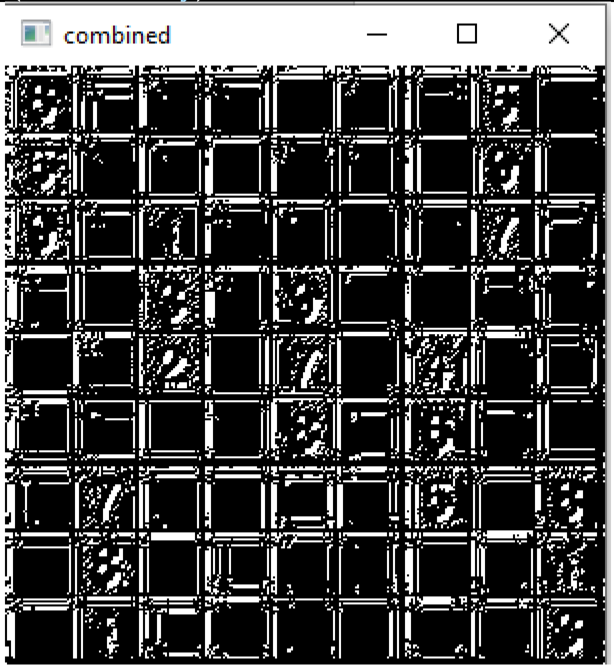

Figure 7: Sudoku image using Sobel combined

Combined the image using bitwise OR. We can’t find the actual edges here. For the negative changes, specific white parts have been replaced with black.

To solve this, we used absolute sobelx.

```
sobelx=np.uint8(np.absolute(sobelx))
```

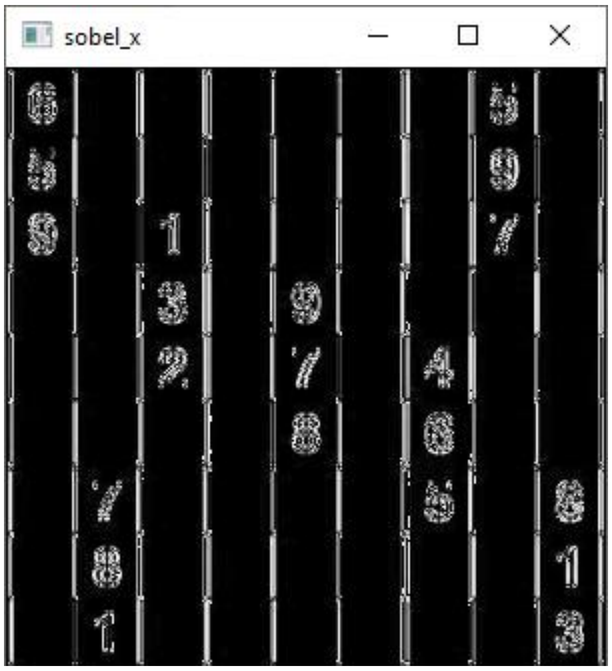

Figure 8: Sudoku image using absolute Sobel X operator

As we used absolute values of negatives, there is no more negative values and so we find a fine tune image here. So for making unsigned integer we used absolute and got a fine-tuned image.

```
sobely=np.uint8(np.absolute(sobely))
```

Again, we used absolute on sobel Y for finding this fine-tuned image.

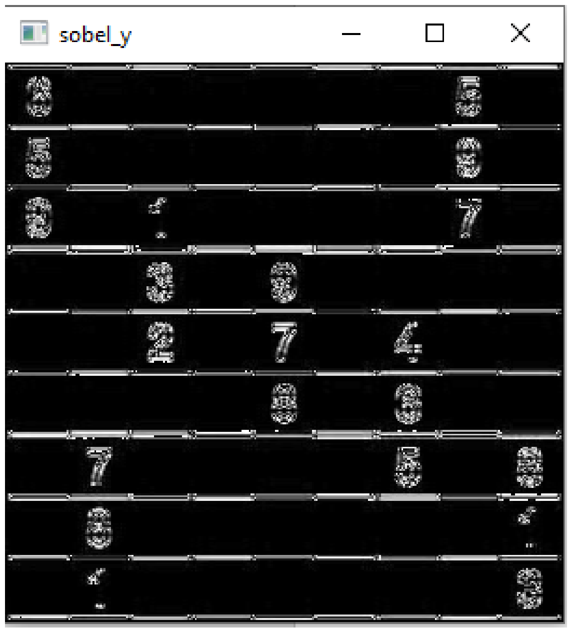

Figure 9: Sudoku image using absolute Sobel Y operator

```
combined= np.uint8(np.absolute(cv.bitwise_or(sobelx,sobely)))
```

That is how we converted the image on unsigned values and after combining this we find an overall fine-tuned image on both X & Y axis.

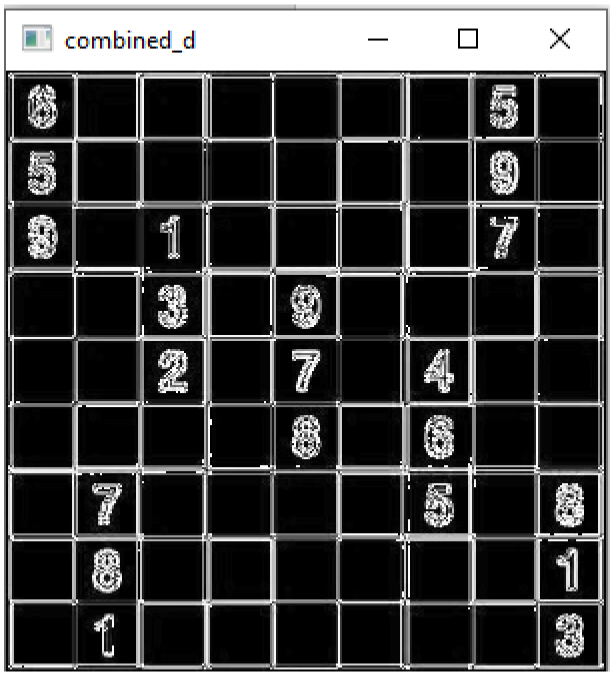

Figure 10: Sudoku image using absolute combined Sobel operator