University of Rajshahi
Department of Computer Science & Engineering
CSE4182 - Digital Image Processing Lab

All report merged

Prepared by: Abdur Rahim Sheikh
Student id: 1810576141
Submitted to: Dr. Md. Khademul Islam Molla
Mr. Md. Rokanujjaman
Sangeeta Biswas
Date: October 14, 2022

Contents

1 Assignment 1: Histogram Plotting **1**
1.1 Introduction 1
1.2 Description 1
1.3 Outcome 1
1.4 Conclusion 2

2 Assignment 2: Using different intensity transformation on image **3**
2.1 Introduction 3
2.2 Description 3
2.3 Outcome 3
2.4 Conclusion 4

3 Assignment 3: applying 6 different kernel to an image **5**
3.1 Introduction 5
3.2 Kernel Generation 5
3.3 Description 5
3.4 Conclusion 6

4 Assignment 4: Plotting histogram manually and built-in method **7**
4.1 Introduction 7
4.2 Description 7
4.2.1 Manual apporach 7
4.2.2 Built in approach 7
4.3 Outcome 8
4.4 Conclusion 8

5 Assignment 5: Binary masking , bit-slicing, convolution using laplacian and sobel filter **9**
5.1 Binary masking 9
5.1.1 Introduction 9
5.1.2 code and outcome 9
5.2 Bit-plane Slicing 10
5.2.1 Introduction 10
5.2.2 Code and Outcome 10
5.3 Laplacian and sobel kernel 11
5.3.1 Introduction 11
5.3.2 Code and Outcome 11
5.4 Conclusion 12

6 Assignment 6: Add 'salt / pepper noise' with an image and see the effect of average kernel, Gaussian kernel and median filter on the noisy image **13**
6.1 Introduction 13
6.2 Adding Salt and Pepper 13
6.3 Adding different filter 13
6.4 Outcome and Conclusion 13

7 Assignment 7: Intensity shifting in histogram **15**
7.1 Explanation 15
7.2 Coding and Outcome 15

8 Assignment 8: Apply morphological operation using 3 structuring element **16**
8.1 Introduction 16
8.2 Code and Outcome 16
8.3 Conclusion 17

9 Assignment 9: morphological operation manually **17**
9.1 Introduction 17
9.2 Dilation 17
9.3 Erosion 17
9.4 Outcome 18

10 Assignment 10: Applying histogram equalization **19**
10.1 Introduction 19
10.2 Description 19
10.3 Outcome 19

11 Assignment 11: Applying Furier transformation on Image **21**
11.1 Introduction 21
11.2 Methods 21
11.3 Transform to Gray-scale 21
11.4 Fourier Transform 21
11.5 Filters 22
11.5.1 Filter1 22
11.5.2 Filter2 23
11.5.3 Filter3 24
11.5.4 Filter4 25
11.6 Conclusion 26

12 Assignment 12: Checking different layer output of VGG16 model **27**
12.1 Introduction 27
12.2 Applying Layers 27
12.2.1 Layer-1 28
12.2.2 Layer-2,3,4 29
12.2.3 Layer-5,6 30
12.2.4 Layer-7,8,9 31

13 Assingment 13: Converting Jpeg to png and vice-versa **33**
13.1 description 33

14 Assignment 14: Image compression in frequency domain **35**
14.1 Introduction 35
14.2 Description 35
14.2.1 Step 1 35
14.2.2 Step 2 35

14.2.3 Step 3 35
14.3 Conclusion 35

15 Assignment 15: Deblurring image using CNN **37**
15.1 Introduction 37
15.2 Outcome of experiment 37
15.2.1 Prediction 1 37
15.2.2 Prediction 2 38

1 Assignment 1: Histogram Plotting

1.1 Introduction

What is histogram? histogram is a way to show frequency of different values or range of values. In image it's different intensity values.

1.2 Description

I was told to show histogram plotting of a RGB images for red, blue,green,gray and binary form.

So I arranged the images and titles like below.

```
img = plt.imread('assignment 1/nature.webp')
gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
_, binary = cv2.threshold(gray, 126, 255, cv2.THRESH_BINARY)
img_concated = [img,img[...,0],img[...,1],img[...,2],gray,binary]
titles = ['rgb','Reds','Greens','Blues','gray','binary']
```

Now i sent these two list in below method taking one object and name each time

```
def plot(img,title,ind):
plt.subplot(6,2,ind*2+1)
if(len(img.shape)==3):
    cmap = None
else:
    cmap = title
plt.title(title)
plt.imshow(img,cmap)
plt.subplot(6,2,ind*2+2)
plt.hist(img.ravel(),255,[0,255])
```

1.3 Outcome

The outcome of the problem is shown below.

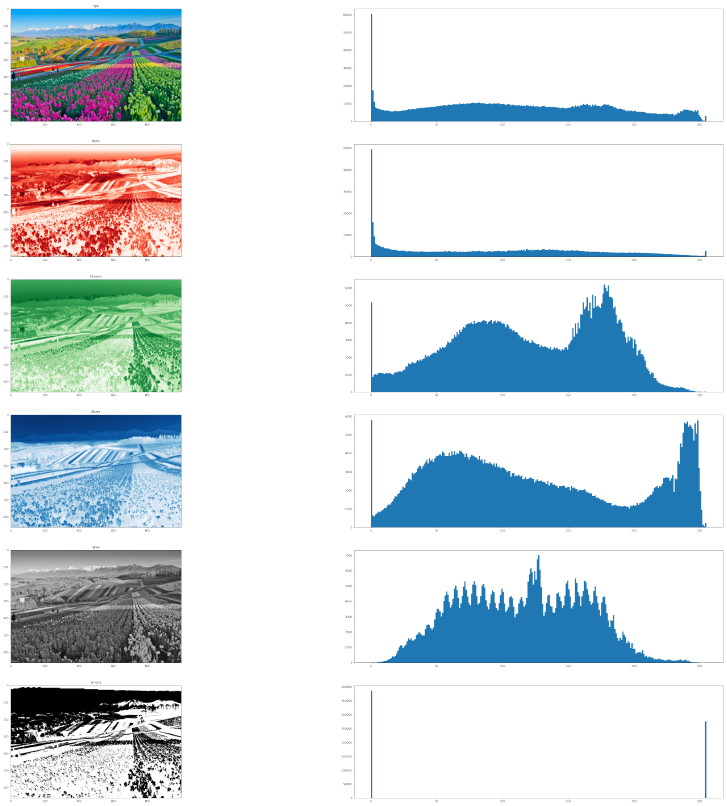

Figure 1.1: Image histogram

1.4 Conclusion

Using histogram we can know the nature of a image. Like is it high contrast, low-contrast, dark or bright then we can enhance the image using histogram-equalization method. Also can check shape of the image in frequency level.

2 Assignment 2: Using different intensity transformation on image

Abstract

Intensity transformation is a process of mapping each intensity value of an input image into the corresponding output intensity value through mathematical expression. Which will enhance the image according to our need. And sometime to make it visually appealing.

2.1 Introduction

We have been asked to perform 4 intensity transformation operation.

1) s = 100, if r ¿= T1 and r ¡= T2; otherwise s = 10. 2) s = 100, if r ¿= T1 and r ¡= T2; otherwise s = r. 3) s = c log(1 + r) . 4) s = c (r + epsilon) p̂

2.2 Description

So these four problem been solved below 1st)

```
r = np.copy(gray_home)
np.testing.assert_equal(r,gray_home)
s1 = np.where((r>=T1) & (r<=T2),100,10)

s2 = np.where((r>=T1) & (r<=T2),100,r)

rp = np.where(((r+1<=0) & (r+1)>255),255,r)
s = c * np.log(rp)
s3 = np.array(s,dtype='uint8')

s4 = c * np.power(( r + epsilon ), p)
```

Finally, we plotted these values.

2.3 Outcome

This is the outcome we got using those mathematical equations.

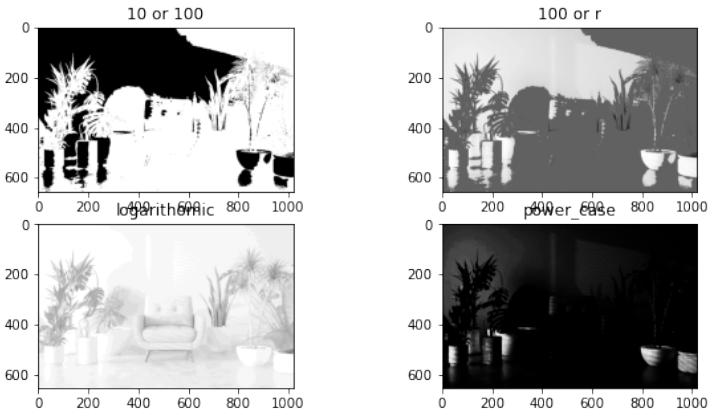

Figure 2.1: intensity transformation

2.4 Conclusion

Using intensity transformation we can enhance image in various notion. And get our desired change in our image which fulfills our need from the image.

3 Assignment 3: applying 6 different kernel to an image

3.1 Introduction

In this assignment we have been asked to generate 6 kernel and apply it to our image to see how it effects on our image.

3.2 Kernel Generation

I randomly generated 6 kernel from size 1 to 6 using below code

```
kernels = []
titles = []
for i in range(1,7):
    kernels.append(random.randint(-i,i,(i+1,i+1)))
    titles.append(f'kernel {i+1},{i+1} shape')
```

3.3 Description

Now as I have generated all kernel it's time to apply them on our image. I selected below image to apply kernel upon.

Figure 3.1: Sweet house

Then I used openCV package to apply filter called filter2D, passed image and kernel as parameter. Also I grayed out our image before applying filter.

```
grayscale = cv2.cvtColor(house,cv2.COLOR_RGB2GRAY)
plt.figure(figsize=(30,30))
for i in range(6):
    plot(cv2.filter2D(grayscale,-1,kernels[i]),titles[i],i+1)
```

Finally I plotted them.

3.4 Conclusion

we can see that till 2*2 filter it was nothing but black image. Then from 3*3 to 6*6 we could see edges of various kind. So we can say that filters are used for various reason and edge detection is most important among them.

4 Assignment 4: Plotting histogram manually and built-in method

4.1 Introduction

Histogram plotting is an idea to plot frequency of distinct intensity value of image. Which helps us go gain information about image.

4.2 Description

I completed it in two way manually and using built-in methods. I applied these idea for these below image.

Figure 4.1: Hen

4.2.1 Manual apporach

Below code calculates frequency values

```
array = []
total = 0
for i in range (256):
    total += (gray_hen==i).sum()
    array.append((gray_hen==i).sum())
print(gray_hen.shape[0]*gray_hen.shape[1],total)
```

4.2.2 Built in approach

bilt in approach used a method of matplotlib package matplotlib.pyplot.hist()

```
plt.hist(gray_hen.ravel(),255,[0,255])
```

4.3 Outcome

Outcome of manual and built-in methods are shown respectively.

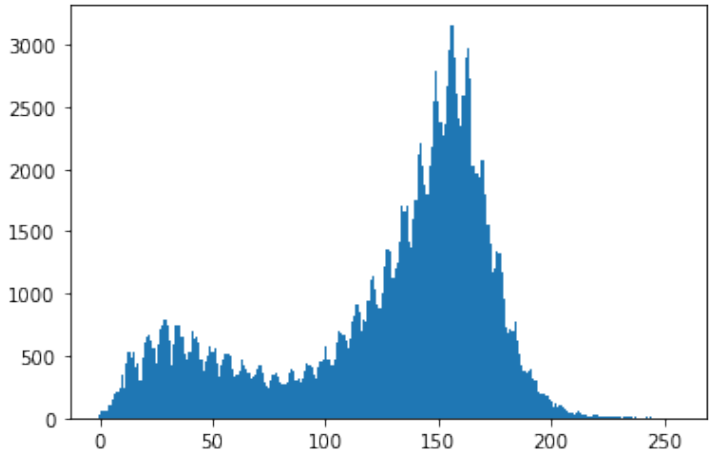

Figure 4.2: Manual Histogram

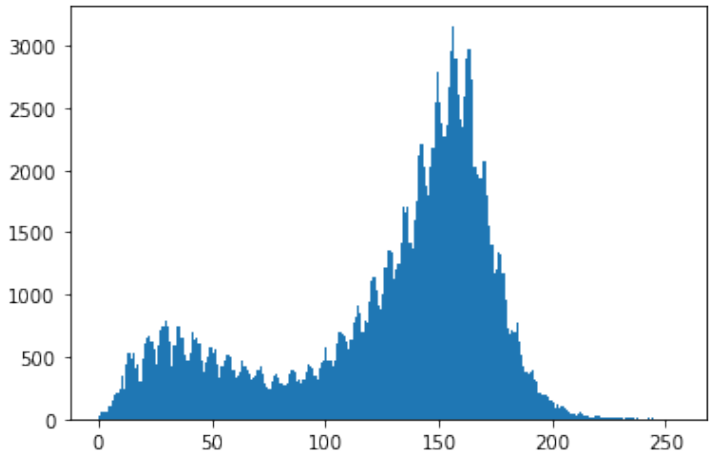

Figure 4.3: built-in histogram

4.4 Conclusion

So we come to see that both output are identical so we have successfully applied histogram calculation idea. And histogram helps us in many way determining the nature of image.Like dark,bright,low-contrast,hight-contrast has different histogram distribution.

5 Assignment 5: Binary masking , bit-slicing, convolution using laplacian and sobel filter

5.1 Binary masking

5.1.1 Introduction

Binary masking is an idea of cropping image in a specific region. So the mask contains high value (255) in area we desire to keep and other area 0. so after using bitwise and on kernel and image desired area will be cropped.

5.1.2 code and outcome

```
dear = plt.imread('assignment 5/dear.jpeg')
print(dear.shape[:2])
mask = np.zeros(dear.shape[:2],dtype="uint8")
mask[50:220,100:330] = 255
masked_dear = cv2.bitwise_and(dear,dear,mask=mask)
```

After this we get

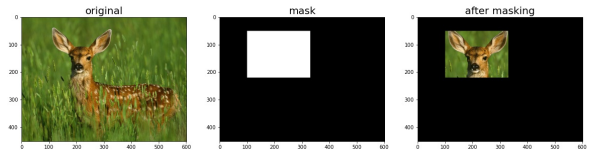

Figure 5.1: Binary masking

5.2 Bit-plane Slicing

5.2.1 Introduction

We know that an image is a representation of 2D intensity values distributed in 0 to 255. So every cell contains a value. Bit-plane slicing means taking n'th bit from every cell. And represent the image in view of that bit plane.

5.2.2 Code and Outcome

Code is

```
def bit_slice(img,bit):
    img = img & (1<<bit)
    return img
```

Outcome,

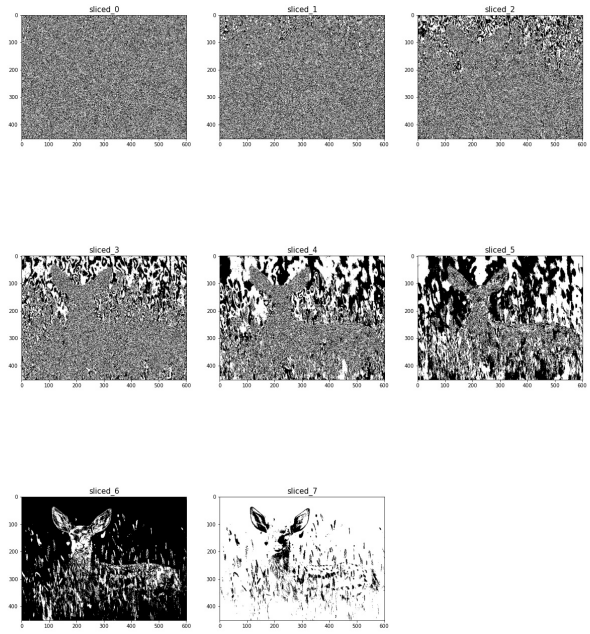

Figure 5.2: bit plane sliced

5.3 Laplacian and sobel kernel

5.3.1 Introduction

Both kernel are edge detector sobel uses first order derivative and response to change in intensity values. Where Laplacian is 2nd order derivative and response to sudden change to intensity values.

5.3.2 Code and Outcome

First we applied filter on below image

Figure 5.3: Deer

Kernel code is,

```
def apply_kernel(img,kernel,ind,title):
    x,y = kernel.shape
    filtered = []
    for i in range(img.shape[0]-x+1):
        row = []
        for j in range(img.shape[1]-y+1):
            calc = np.sum(np.multiply(kernel,img[i:i+x,j:j+y]))
            if calc<0:
                calc = 0
            elif calc>255:
                calc = 255
            row.append(calc)
        filtered.append(row)

    filtered_hen = np.array(filtered)
    plt.subplot(1,2,ind)
    plt.title(title,fontsize=10)
```

```
    plt.imshow(filtered_hen,cmap='gray')
    plt.savefig('problem_3- kernel_apply.jpeg')
```

After applying kernel we used below code to plot them

```
sobel = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
laplacian = np.array([[0, 1, 0],[1, -4, 1],[0, 1, 0]])
plt.figure(figsize=(20,22))
apply_kernel(gray_dear,sobel,1,'sobel filter')
apply_kernel(gray_dear,laplacian,2,'laplacian filter')
```

Outcome,

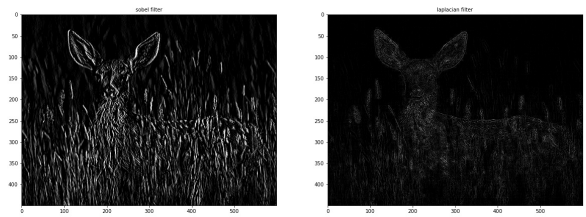

Figure 5.4: Laplacian and Sobel

5.4 Conclusion

Binary masking helps us to concentrate specific region of an image to apply various kinds of enhancement without the concern of affecting other area. **Bit plane slicing** helps to compress the image size. We can see in figure 5.2 that 6th,7'th and 8'th bit contain's almost all the information of the deer. So we just can store these three bit to save memory.

6 Assignment 6: Add 'salt / pepper noise' with an image and see the effect of average kernel, Gaussian kernel and median filter on the noisy image

6.1 Introduction

salt and pepper are two random noise in image salt noise is white or high value and pepper noise is black. Our task is to first add salt and pepper noise to an image then apply those three different kernel to reduce the noise.

6.2 Adding Salt and Pepper

From the image size we add 20% area to be noised randomly using below code.

```
w,h = grayImg.shape
total_salty = (w*h)//20
Xaxis = np.random.randint(w,size = total_salty)
Yaxis = np.random.randint(h,size = total_salty)
noisyImg = np.copy(grayImg)
for a,b in zip(Xaxis,Yaxis):
    bit = np.random.randint(0,2) * 255
    noisyImg[a,b] = bit
```

6.3 Adding different filter

Code,

```
average_noise = cv2.filter2D(noisyImg,-1,avg_kernel)
title.append('average noise')
images.append(average_noise)

gaussian = cv2.GaussianBlur(noisyImg,(3,3),0)
title.append('gaussian filter')
images.append(gaussian)

median = cv2.medianBlur(noisyImg,3)
title.append('median')
images.append(median)
```

6.4 Outcome and Conclusion

We can see in the figure that the median filter worked best for pepper and salt noise. Gaussian and average performed almost same.

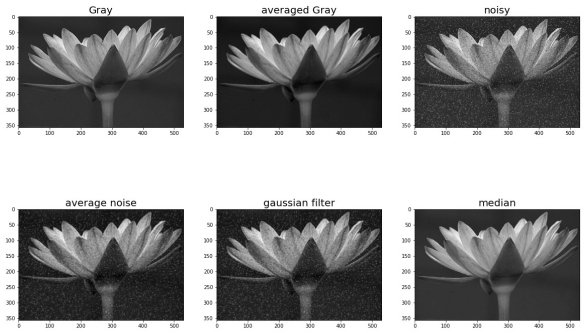

Figure 6.1: Pepper Salt Noise reduction

7 Assignment 7: Intensity shifting in histogram

7.1 Explanation

This assignment is too small just need to shift intensity which is constant value add or subtract to every cell in manner of point processing to right-shift or left-shift intensity. Challenge here is to control overflow and under flow.

7.2 Coding and Outcome

```
range_img = np.copy(gray)
range_img[gray<50] = 50
range_img[gray>200] = 200
imgs = [gray,gray+30,gray-30,range_img]
titles = ['original','right_shifted','left_shifted','range_value']
cnt = 0

plt.figure(figsize=(20,20))
for img,title in zip(imgs,titles):
    img_plot(img,title,cnt)
    cnt += 1
```

Outcome is,

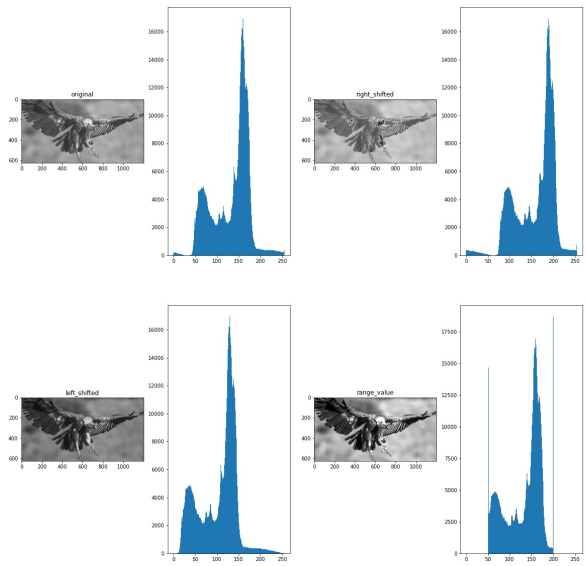

Figure 7.1: Histogram shifting

8 Assignment 8: Apply morphological operation using 3 structuring element

Abstract

Morphology is a comprehensive set of image processing operations that process images based on shapes. All morphological processing operations are based on mentioned terms.

Structuring Element: It is a matrix or a small-sized template that is used to traverse an image. The structuring element is positioned at all possible locations in the image, and it is compared with the connected pixels. It can be of any shape. Fit: When all the pixels in the structuring element cover the pixels of the object, we call it Fit. Hit: When at least one of the pixels in the structuring element cover the pixels of the object, we call it Hit. Miss: When no pixel in the structuring element cover the pixels of the object, we call it miss. [1]

8.1 Introduction

So we have calculated morphological operations using cv2 method cv2.erode for erosion and cv2.dilate for dilation.

8.2 Code and Outcome

Code,

```
erosions = []
dialations = []
opening = []
closing = []
for kernel in [kernel1,kernel2,kernel3]:
#     print(kernel)
    erosions.append(cv2.erode(image, kernel))
    dialations.append(cv2.dilate(image, kernel))
#     opening
    opening.append(cv2.dilate(cv2.erode(image,kernel),kernel))
#     closing
    closing.append(cv2.erode(cv2.dilate(image,kernel),kernel))
```

[1] https://towardsdatascience.com/understanding-morphological-image-processing-and-its-operations-

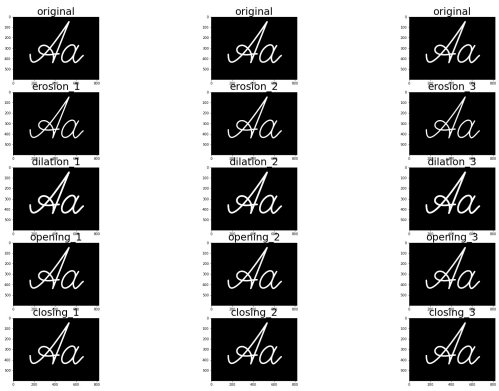

Figure 8.1: morphology

8.3 Conclusion

Compound version of morphology like opening, helps us various real world problem like, 1) Erosion can split apart joined objects or strip away extrusions 2) Dilation can repair breaks, intrusions 3) Combination of them are used in boundary extraction 4) Region Filling

9 Assignment 9: morphological operation manually

9.1 Introduction

Morphology already been discussed in assignment 8 in this assignment we have to manually implement erosion,dilation,opening,closing

9.2 Dilation

For dilation I have counted if the kernel area hits if hits 1 otherwise 0

```
def dilate(img,kernel):
    h,w = kernel.shape
    new_img = np.zeros((img.shape[0]-h+1,img.shape[1]-w+1))

    for i in range(new_img.shape[0]):
        for j in range(new_img.shape[1]):
            hit = np.matmul(img[i:i+h,j:j+w],kernel).sum() > 0
            new_img[i,j] = hit

    return new_img
```

9.3 Erosion

For erosion we checked if the kernel area fits or not. If fits 1 otherwise 0.

```python
def erode(img,kernel):
    h,w = kernel.shape
    new_img = np.zeros((img.shape[0]-h+1,img.shape[1]-w+1))

    for i in range(new_img.shape[0]):
        for j in range(new_img.shape[1]):
            fit = np.matmul(img[i:i+h,j:j+w],kernel).sum() >= kernel.sum()
            new_img[i,j] = fit

    return new_img
```

9.4 Outcome

Outcome of this functions is same as built-in method used in previous assignment.

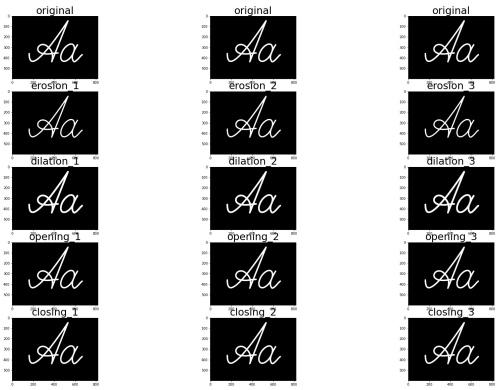

Figure 9.1: morphology

10 Assignment 10: Applying histogram equalization

Abstract

Histogram equalization is an idea to distribute frequency so that the image converts into high-contrast image. We know that from histogram perspective there is four type of image. 1) High contrast image 2) Low contrast image 3) Dark image 4) Bright image

Then after equalization all image turns to high contrast image. Which is appealing to our visualization. The idea is simple in python syntax new_img[i,j] = np.round((cdf[img[i,j]] - cdf.min())/(m*n-cdf.min()) * ((1¡¡8) -1)) here cdf is cumulative distribution function. where cdf(i) = frequency(0) + frequency(1) .... + frequency(i)

10.1 Introduction

I applied this idea manually and also applied it in histEqualize method and compared them. I have already described how would I do this.

10.2 Description

```
hist,bins = np.histogram(img,256,[0,256])
m,n = img.shape
cdf = hist.cumsum()
new_img = img.copy()
for i in range(m):
    for j in range(n):
        new_img[i,j] = np.round((cdf[img[i,j]] - cdf.min())/(m*n-cdf.min()) * ((1<<8)
```

Above code creates new image with equalized frequency distribution.

Finally I plotted it using below code.

```
images = [img,new_img,cv2.equalizeHist(img)]
titles = ['original','manual equalization','cv2.equalizeHist']
plt.figure(figsize=(30,20))
for i in range(3):
    image_plotting(images[i],i*2+1,titles[i])
```

10.3 Outcome

Here is my outcome,

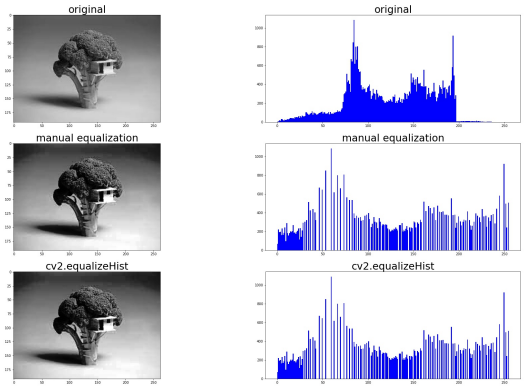

Figure 10.1: Histogram Equalization

11 Assignment 11: Applying Furier transformation on Image

Abstract

The Fourier Transform is used to work if signal as well as image in frequency domain. And a fastest version of Fourier Transform which is fft (Fast Fourier Transform) can be used to work with any signal in frequency domain with minimal time complexity and more flexibility.

11.1 Introduction

To understand the effect of Fourier Transform we need an image and here is the image I chose to apply fast Fourier transform.

Figure 11.1: red rose

11.2 Methods

This is an RGB image. We will apply Fourier Transform in a gray-scale image. To convert gray-scale we will use the python module OpenCV called cv2.

11.3 Transform to Gray-scale

```
img = plt.imread('rose.jpg')
src_img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
plt.imshow(src_img)
```

11.4 Fourier Transform

In this report, we used Numpy's built-in function FFT2 (2 because our image is a 2D signal). Then centered the Fourier transformed image by using "fftshift" function. Then, we will calculate the magnitude spectrum.

```
fftImg = np.fft.fft2(src_img)
centerImg = np.fft.fftshift(fftImg)
```

```
centerImgLog = 100 * np.log(abs(centerImg))
```

11.5 Filters

I created 4 filter kind of arbitary but those are 3 rectengle of different size and one circle.

11.5.1 Filter1

It's a 100 by 100 rectangle in the center.

```
filter1 = np.zeros((r,c),np.uint8)
filter1= cv2.rectangle(filter1,(oy-50,ox-50),(oy+50,ox+50),(255,255,255),-1)
imageBack1 = centerImg * filter1
filterImg1 = np.abs(np.fft.ifft2(imageBack1))
```

Output:

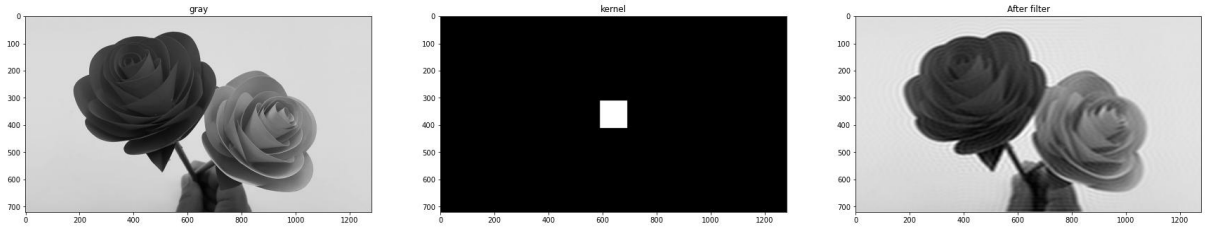

Figure 11.2: Filter 1 applied in frequency domain(Fourier Transformed)

Analysis: there is some circular glow around the image.

11.5.2 Filter2

This filter is a circle in the center.

```
filter2 = np.zeros((r,c),np.uint8)
filter2 = cv2.circle(filter2,(oy,ox),70,(255,255,255),-1)
imageBack2 = centerImg * filter2
filterImg2 = np.abs(np.fft.ifft2(imageBack2))
```

Output:

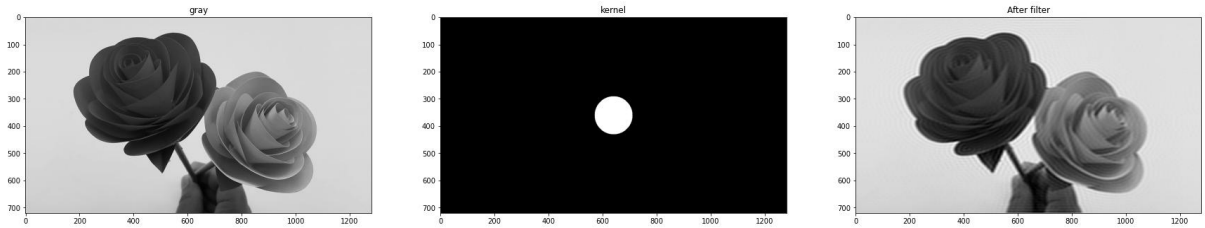

Figure 11.3: Filter 2 applied in frequency domain(Fourier Transformed)

Analysis: This also have circular glow but it's more sharp then the rectangle.

11.5.3 Filter3

The idea was to create a filter that contained half of all the values 1 and the others zero (axis=column). It is the same as above but the difference is we applied it with a column axis.

```
filter3 = np.zeros((r,c),np.uint8)
filter3[:oy,:] = 1
imageBack3 = centerImg * filter3
filterImg3 = np.abs(np.fft.ifft2(imageBack3))
```

Output:

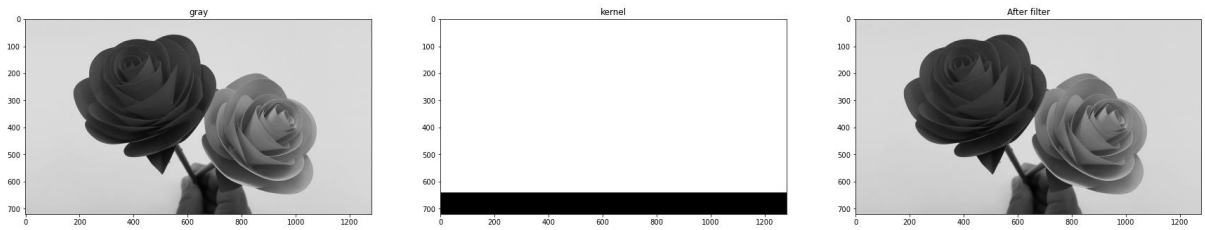

Figure 11.4: Filter 3 applied in frequency domain(Fourier Transformed)

Analysis: No effect seen.

11.5.4 Filter4

It's another rectangle.

```
filter4 = np.zeros((r,c),np.uint8)
filter4[:ox+30,:oy+90] = 1
imageBack4 = centerImg * filter4
filterImg4 = np.abs(np.fft.ifft2(imageBack4))
```

Output:

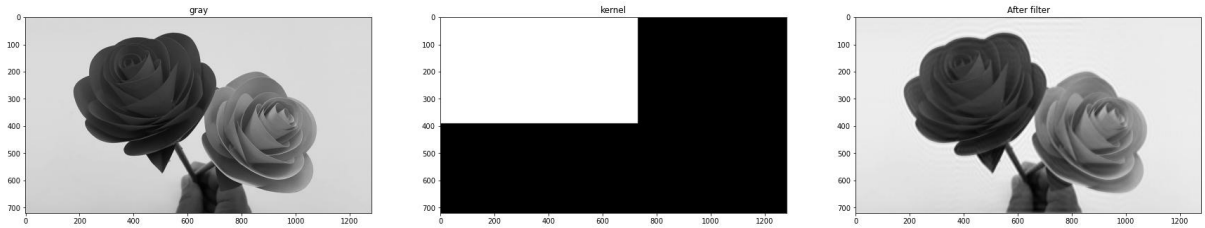

Figure 11.5: Filter 4 applied in the frequency domain(Fourier Transformed)

11.6 Conclusion

The main advantage of Fourier transform is that we can apply filters in the frequency domain. And it's time complexity is less. So it's a great tool to filter images and do other application with it.

12 Assignment 12: Checking different layer output of VGG16 model

12.1 Introduction

VGG16 is a model developed in Oxford University in 2014 for object detection and classification. We are told to feed an image and show which layer most specifically n'th kernel does what with the image.

12.2 Applying Layers

There is total of 23 in VGG16 where layer **0** is just gray image then **2 to 9** gives visually understandable output **10 to 13** gives some binary related value on helps the computer and left **14 to 22** layer is not interpret able as image. So I am just going to explain we I can interpret visually from this image

Figure 12.1: Original Image

12.2.1 Layer-1

It is detected every differential value of the image it's showing all the edges similar to laplacian filter.

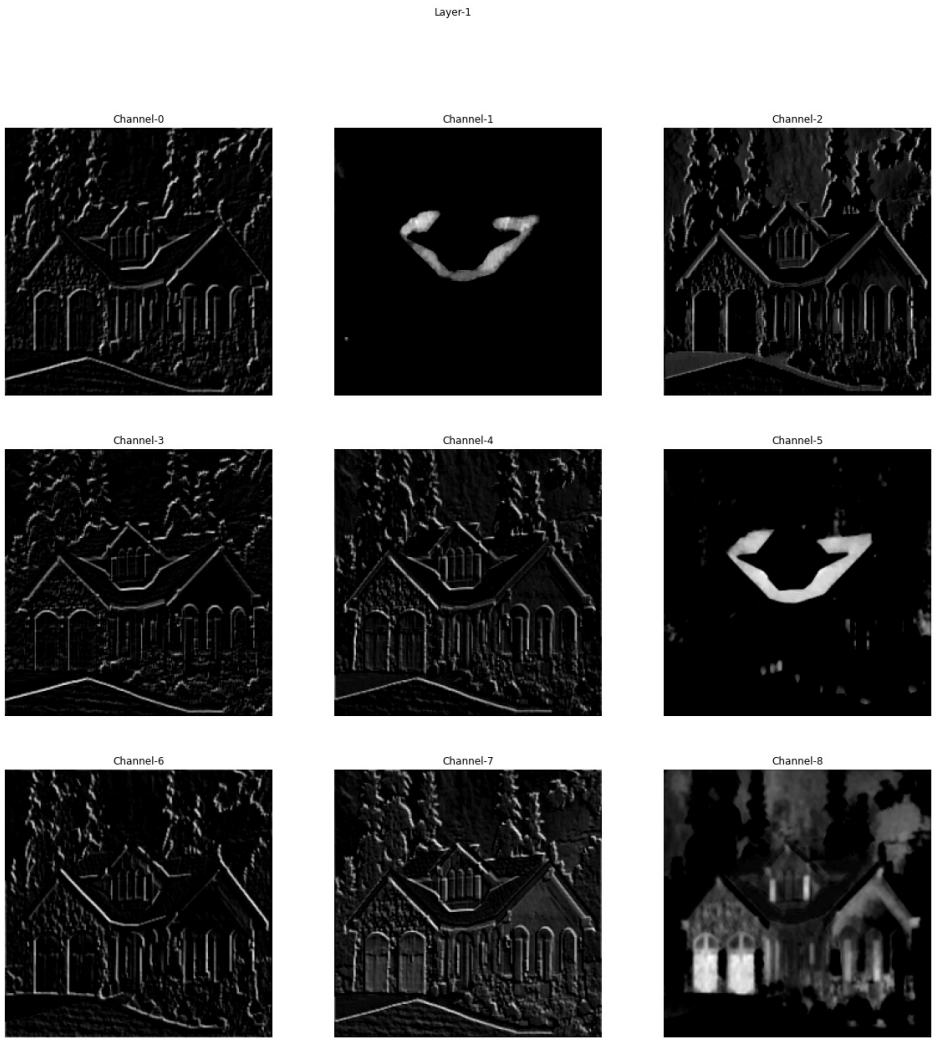

Figure 12.2: Output of Layer 1

12.2.2 Layer-2,3,4

It looks like pictures has been smoothen in this layer. 3 and 4 looks almost same.

Figure 12.3: Output of Layer 2

12.2.3 Layer-5,6

In these layer picture has been distorted from the smoothing.

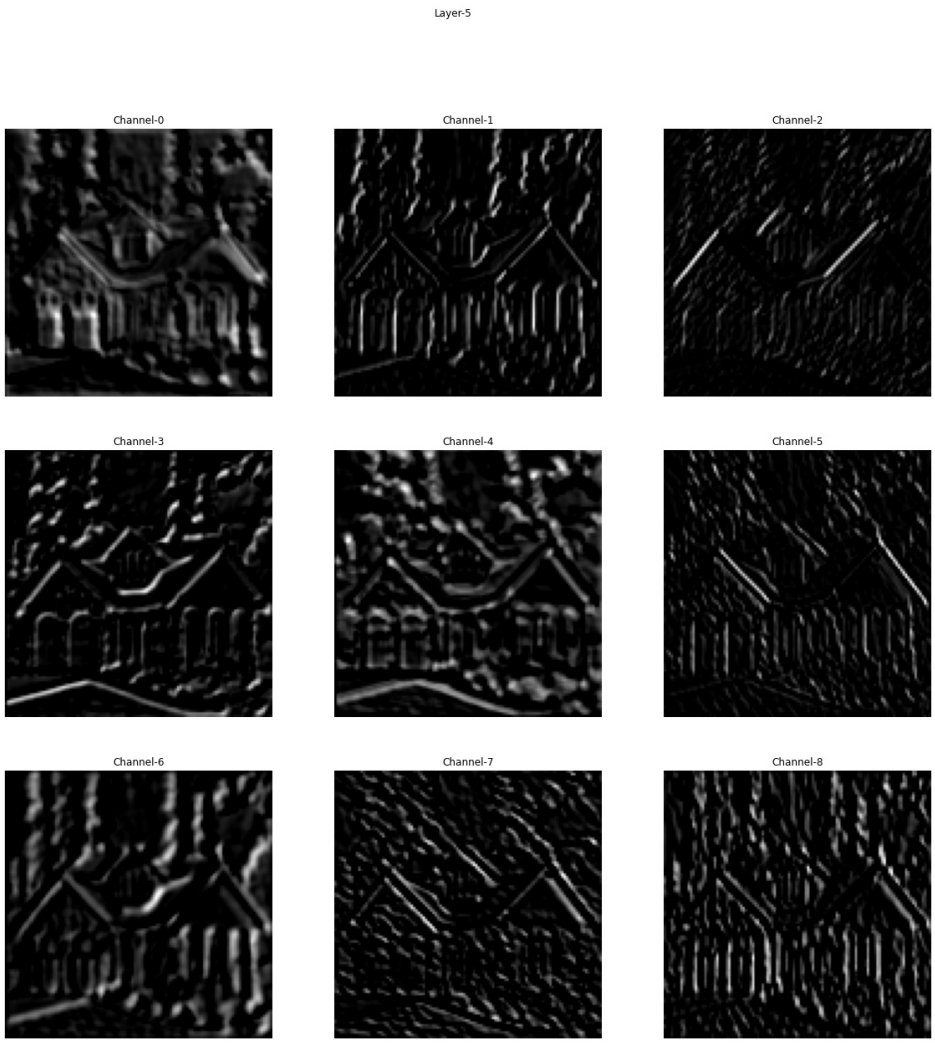

Figure 12.4: Output of Layer 5

12.2.4 Layer-7,8,9

In these layers each each part is separated.

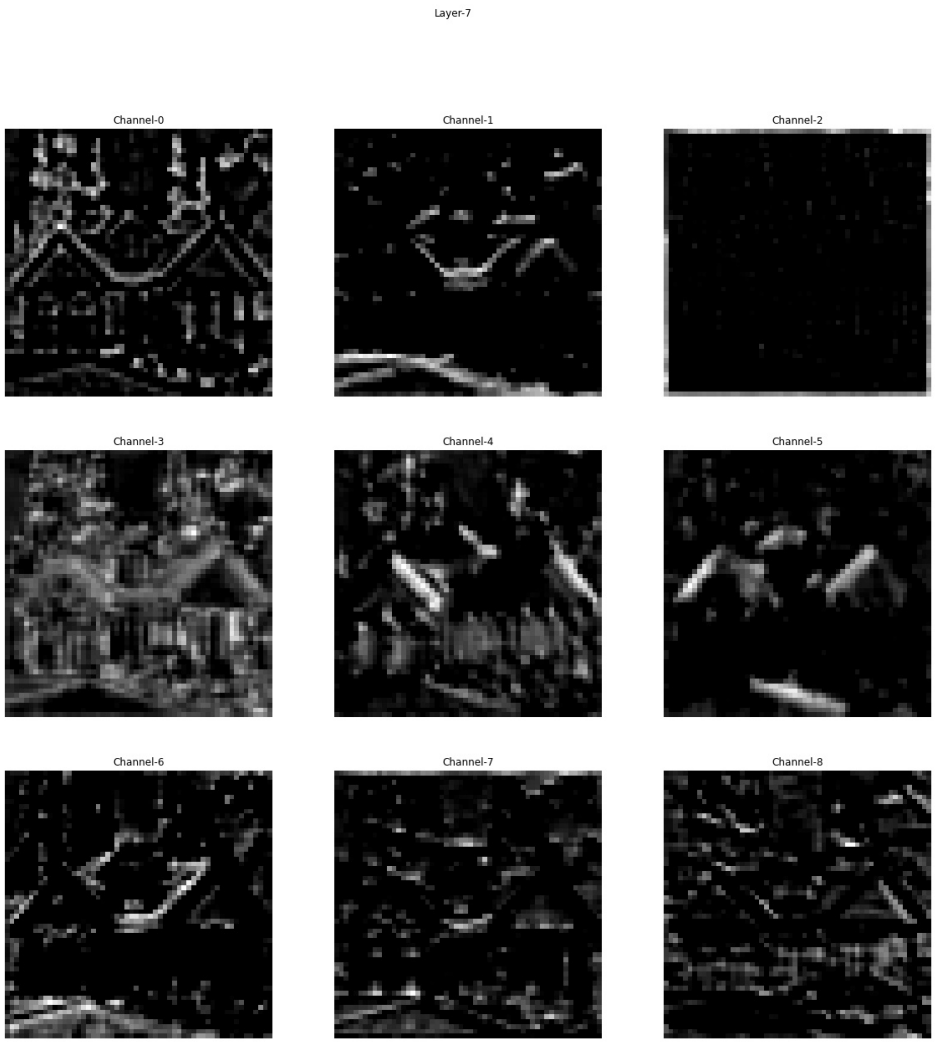

Figure 12.5: Output of Layer 7

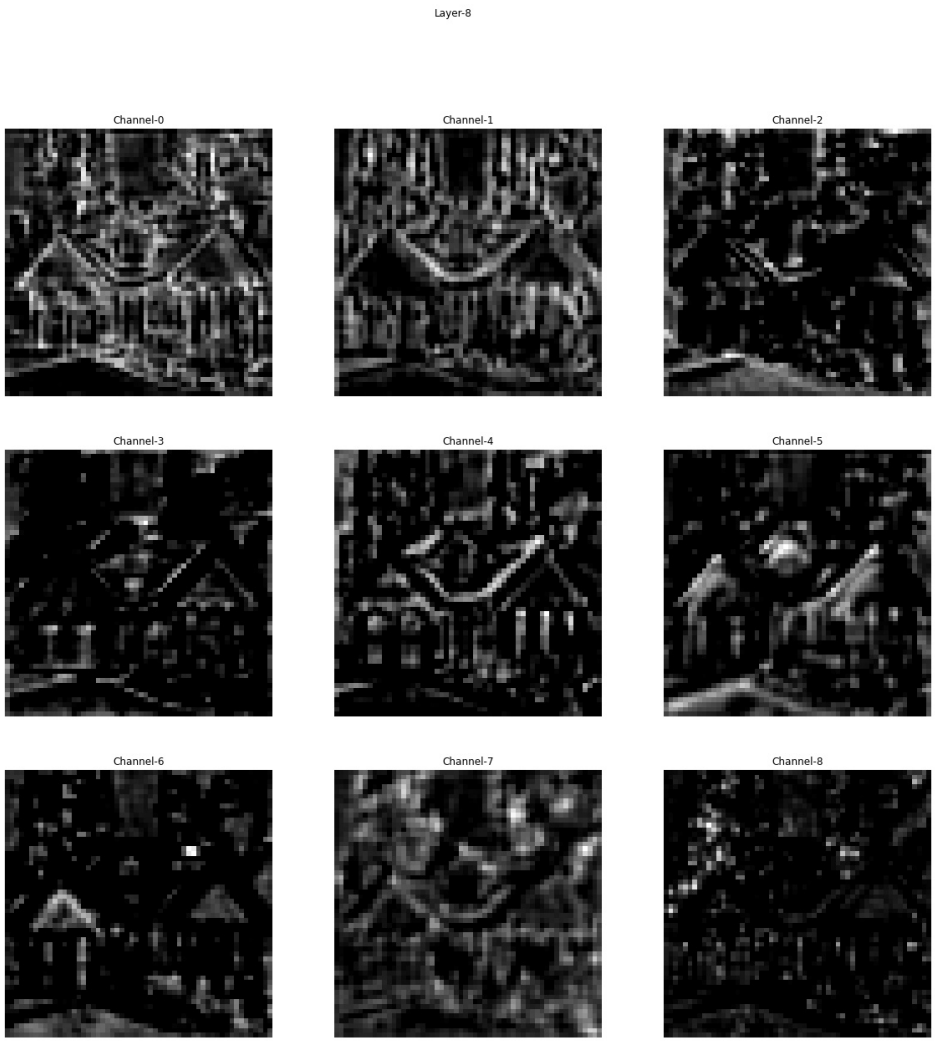

Figure 12.6: Output of Layer 8

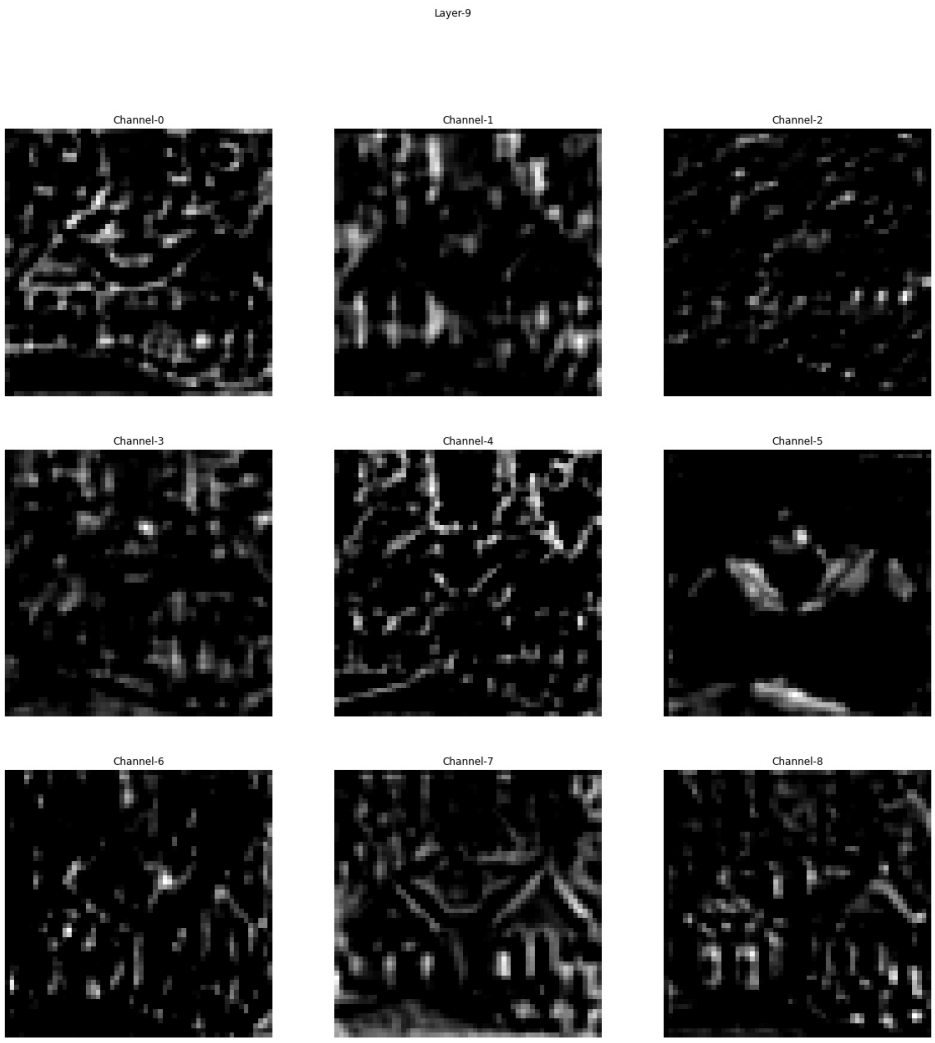

Figure 12.7: Output of Layer 9

13 Assingment 13: Converting Jpeg to png and vice-versa

13.1 description

I was mainly asked to see the format of any image and change there format so the image change internally. used PIL packge of python to do the job.

```
from PIL import Image
img = Image.open('world.jpeg')
img.save("changed_world.png",format='png')

img2 = Image.open('changed_world.png')
```

```
img2.save('changed_world_again.jpeg',format='jpeg')
img3 = Image.open('changed_world_again.jpeg')
print(img.format,img2.format,img3.format)
```

output: **JPEG PNG JPEG**

14 Assignment 14: Image compression in frequency domain

14.1 Introduction

If we can loss some low level data which we don't think necessary for what we storing the data. Then we can discard them from the image and that's how we can compress it's size and save space in disk and easy to transfer data.

14.2 Description

14.2.1 Step 1

So what we can do is first transform the image in frequency domain using fast furier transform.

```
src_img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
fftImg = np.fft.fft2(src_img)
```

14.2.2 Step 2

Then sort their magnitude and find the value at which extend of frequency we want to keep.

```
imgSort = np.sort(np.abs(fftImg.ravel()))
thresh = imgSort[int(np.floor(n * (1 - keep)))]
ind = np.abs(fftImg) > thresh
```

14.2.3 Step 3

Finally apply the 'ind' mask to the fft and back the original image.

```
allow_pass = fftImg * ind
ifftImg = np.fft.ifft2(allow_pass).real
```

14.3 Conclusion

Following those method i applied '(0.75 , 0.5 , 0.1 , 0.05 , 0.01, 0.001 , 0.0001)' portion keeping the image and what they left is shown below.

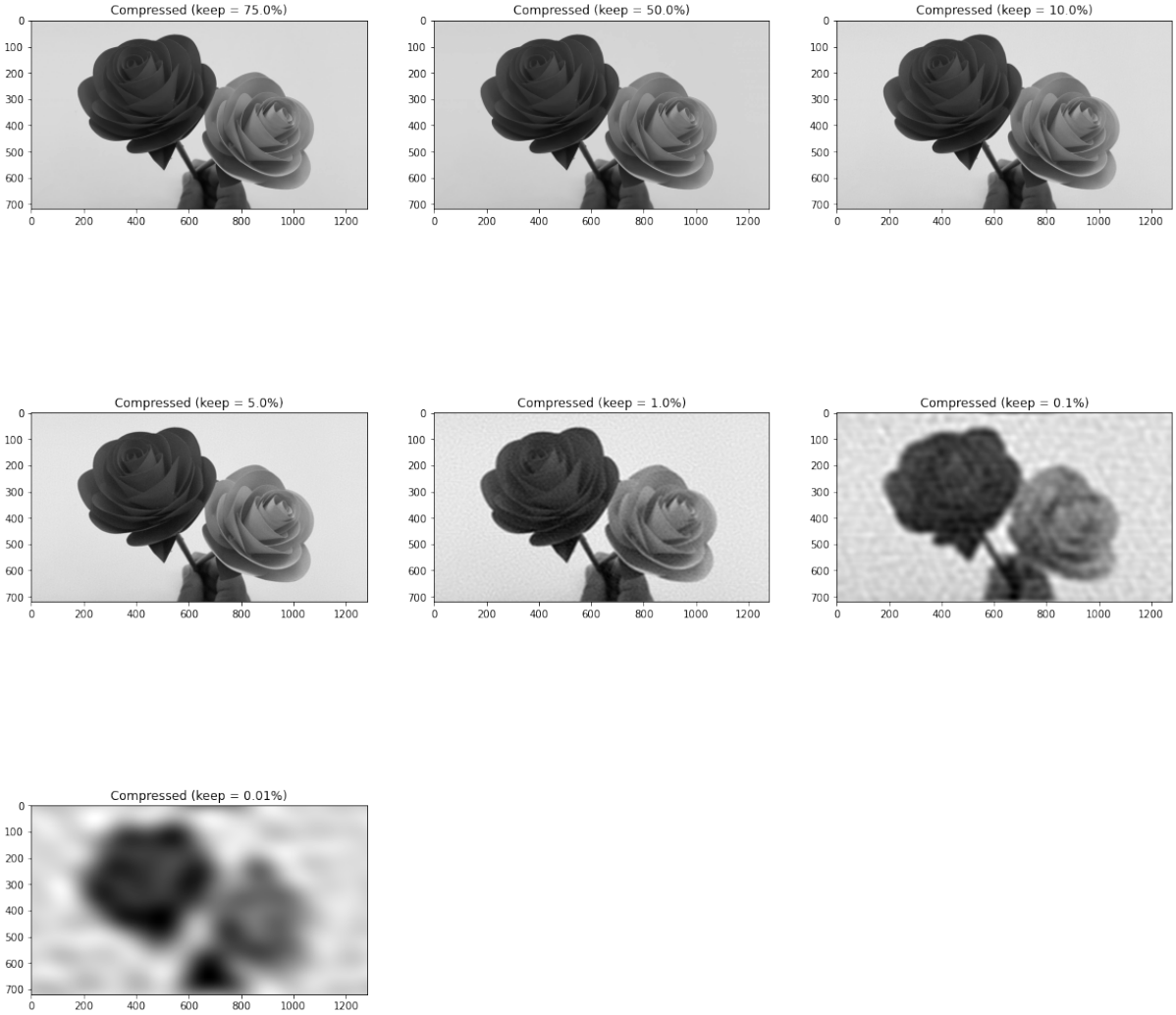

Figure 14.1: compressed image

15 Assignment 15: Deblurring image using CNN

15.1 Introduction

What is neural network in computer is. Making a system which will predict something after been trained.Here we are deblurring images. Now this assignment is a simple check for checking how model works. It's been trained on famous dataset mnist which are numbers from 0 to 9.

15.2 Outcome of experiment

As this was trained on mnist datset I have predicted 2 image one for number deblur and second one is my personal hand-picked blurred picture.

15.2.1 Prediction 1

The digit deblurring went smooth as you can see in below figure that model has deblurred pretty well.

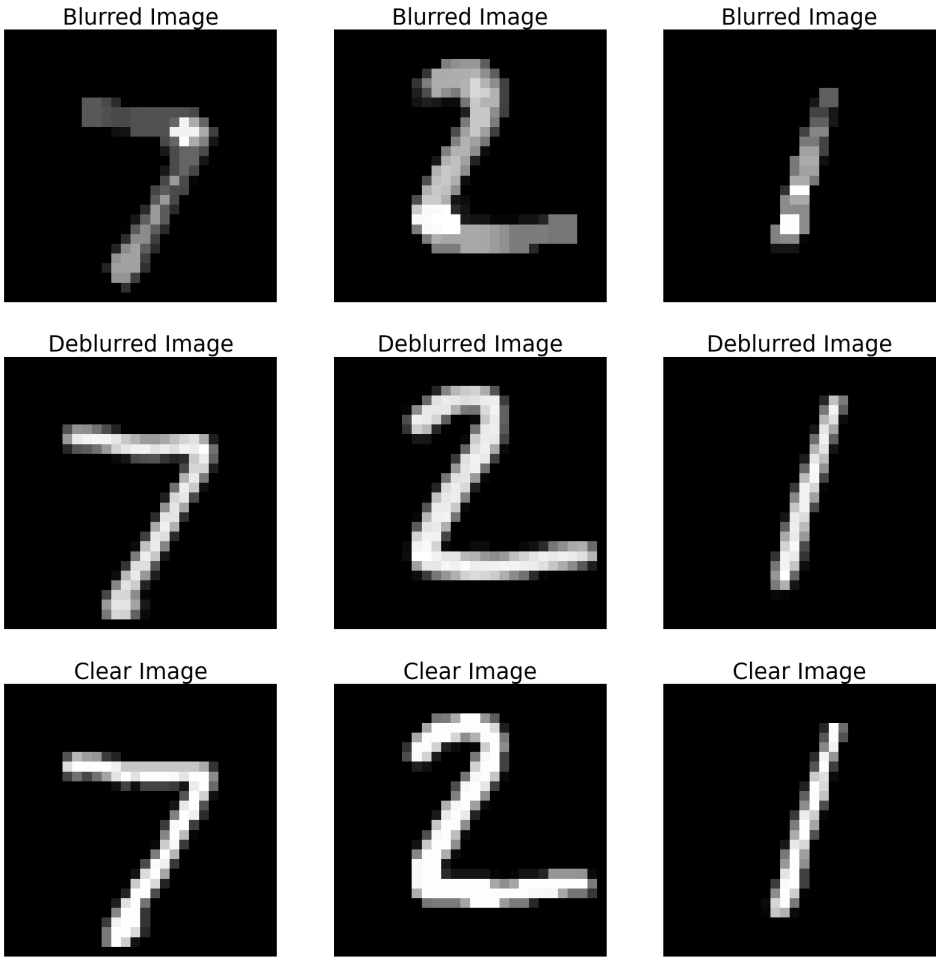

Figure 15.1: Digit Deblurred

15.2.2 Prediction 2

My local image did not performed well, as the dataset was not trainned for this and this is more complex than digit shape.

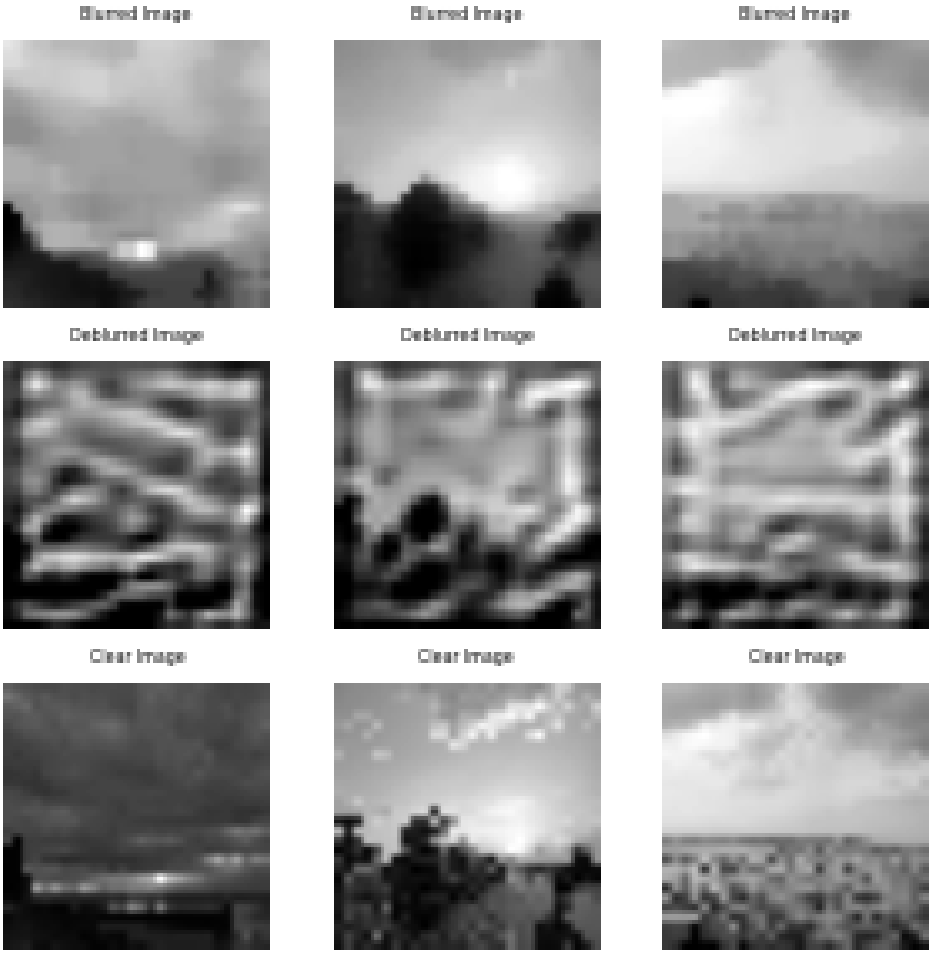

Figure 15.2: River Deblurred

List of Figures

1.1 Image histogram 2
2.1 intensity transformation 4
3.1 Sweet house 5
4.1 Hen 7
4.2 Manual Histogram 8
4.3 built-in histogram 8
5.1 Binary masking 9
5.2 bit plane sliced 10
5.3 Deer 11
5.4 Laplacian and Sobel 12
6.1 Pepper Salt Noise reduction 14
7.1 Histogram shifting 15
8.1 morphology 17
9.1 morphology 18
10.1 Histogram Equalization 20
11.1 red rose 21
11.2 Filter 1 applied in frequency domain(Fourier Transformed) 22
11.3 Filter 2 applied in frequency domain(Fourier Transformed) 23
11.4 Filter 3 applied in frequency domain(Fourier Transformed) 24
11.5 Filter 4 applied in the frequency domain(Fourier Transformed) 25
12.1 Original Image 27
12.2 Output of Layer 1 28
12.3 Output of Layer 2 29
12.4 Output of Layer 5 30
12.5 Output of Layer 7 31
12.6 Output of Layer 8 32
12.7 Output of Layer 9 33
14.1 compressed image 36
15.1 Digit Deblurred 38
15.2 River Deblurred 39