University of Rajshahi

Department of Computer Science & Engineering

CSE4182 - Digital Image Processing Lab

All report merged

Prepared by: Abdur Rahim Sheikh
student id: 1810576141
Submited to: Dr. Md. Khademul Islam Molla
Mr. Md. Rokanujjaman
Sangeeta Biswas
Date: September 16, 2022

Contents

1 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.

1.1 Introduction

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

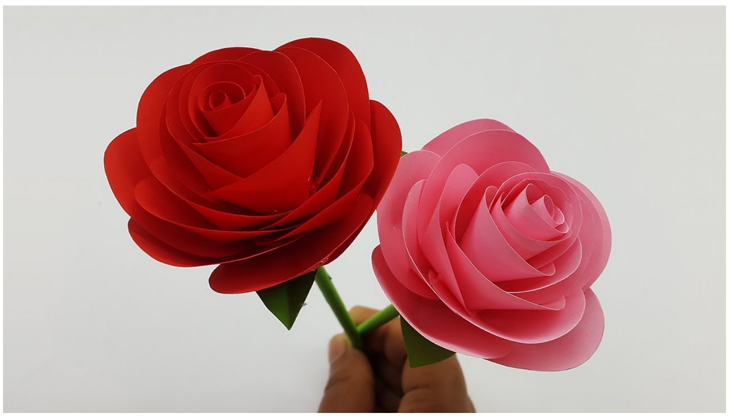

Figure 1.1: red rose

1.2 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.

1.3 Transform to Gray-scale

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

1.4 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))
```

1.5 Filters

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

1.5.1 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:

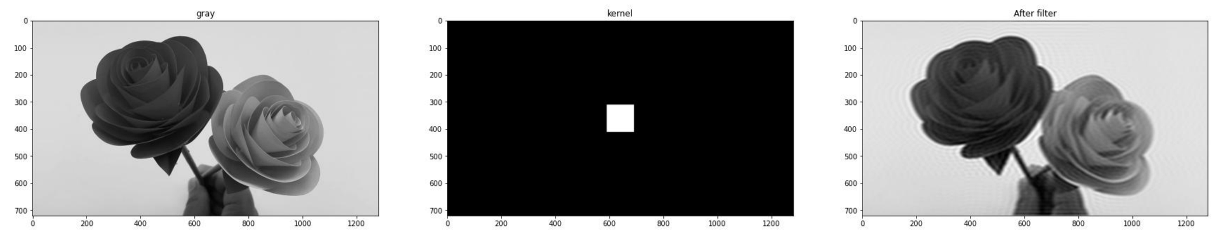

Figure 1.2: Filter 1 applied in frequency domain(Fourier Transformed)

Analysis: there is some circular glow around the image.

1.5.2 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:

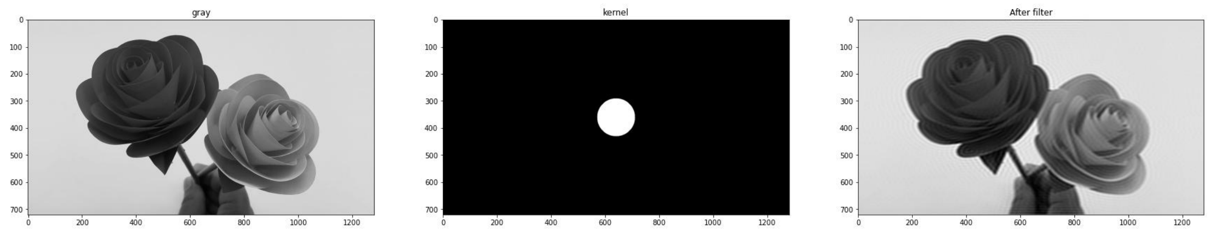

Figure 1.3: Filter 2 applied in frequency domain(Fourier Transformed)

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

1.5.3 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:

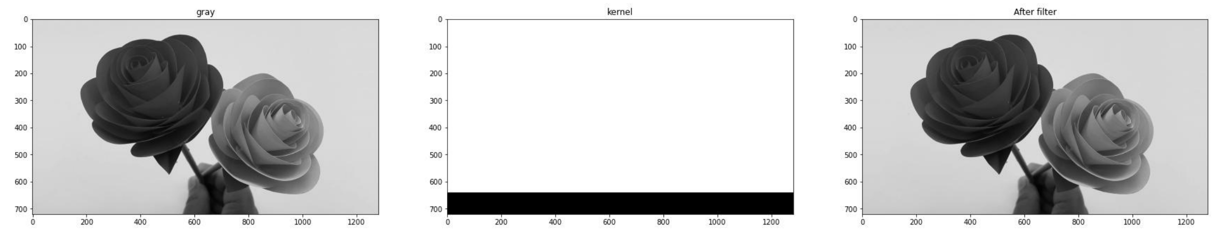

Figure 1.4: Filter 3 applied in frequency domain(Fourier Transformed)

Analysis: No effect seen.

1.5.4 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:

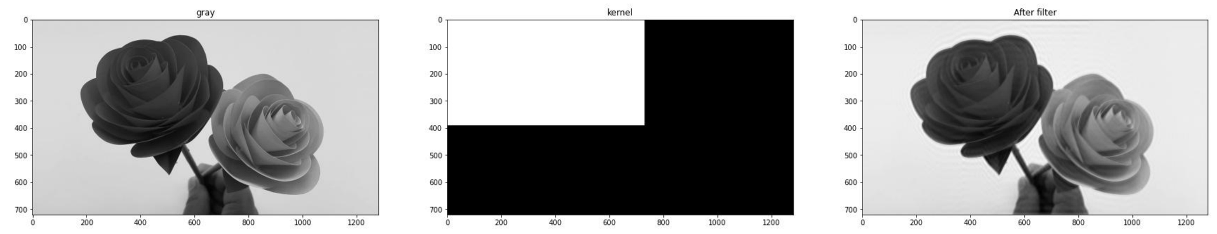

Figure 1.5: Filter 4 applied in the frequency domain(Fourier Transformed)

1.6 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.

2 Assignment 12: Checking different layer output of VGG16 model

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.

2.1 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

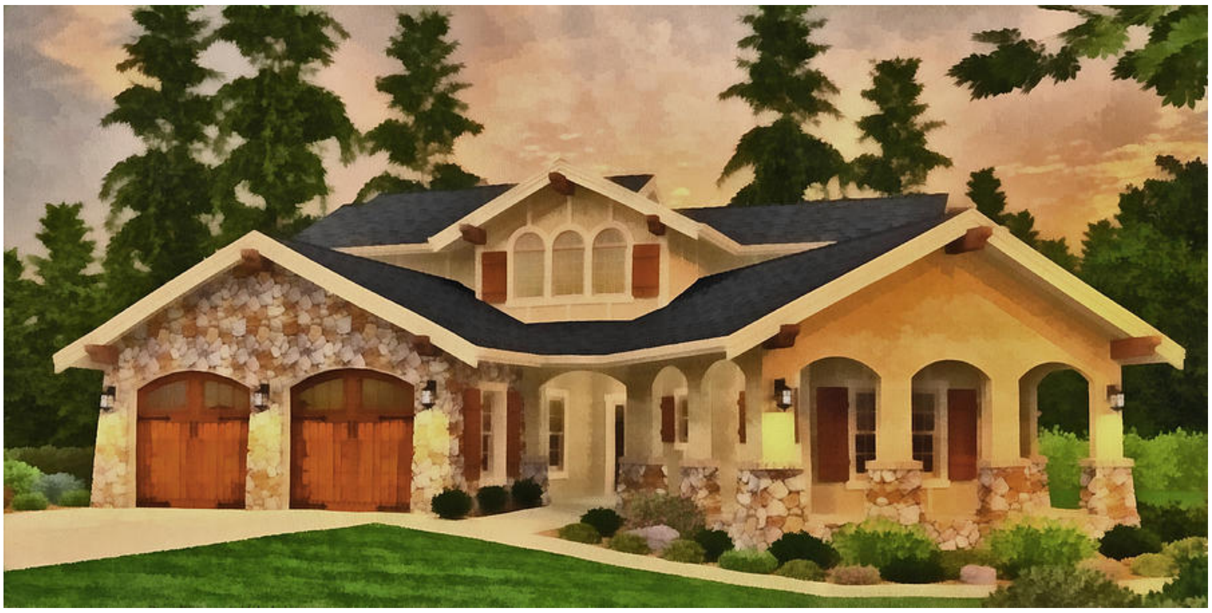

Figure 2.1: Original Image

2.1.1 Layer-1

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

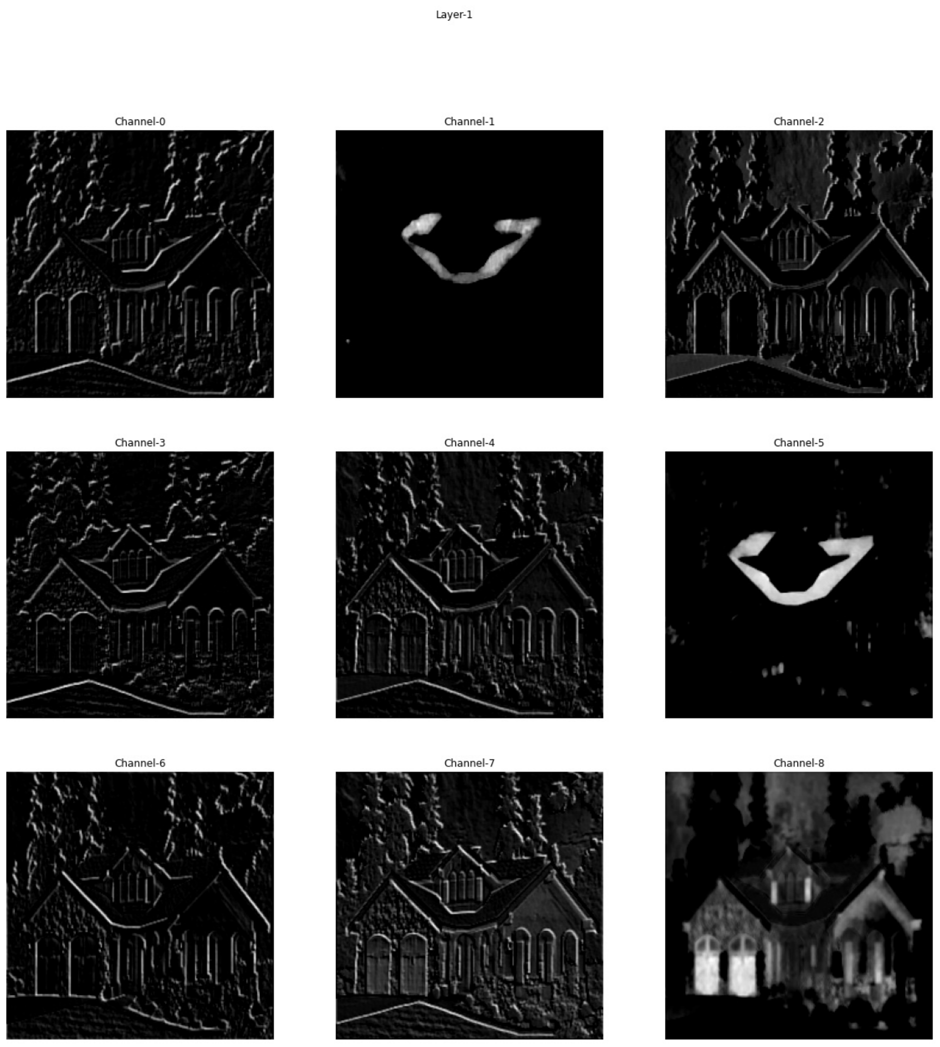

Figure 2.2: Output of Layer 1

2.1.2 Layer-2,3,4

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

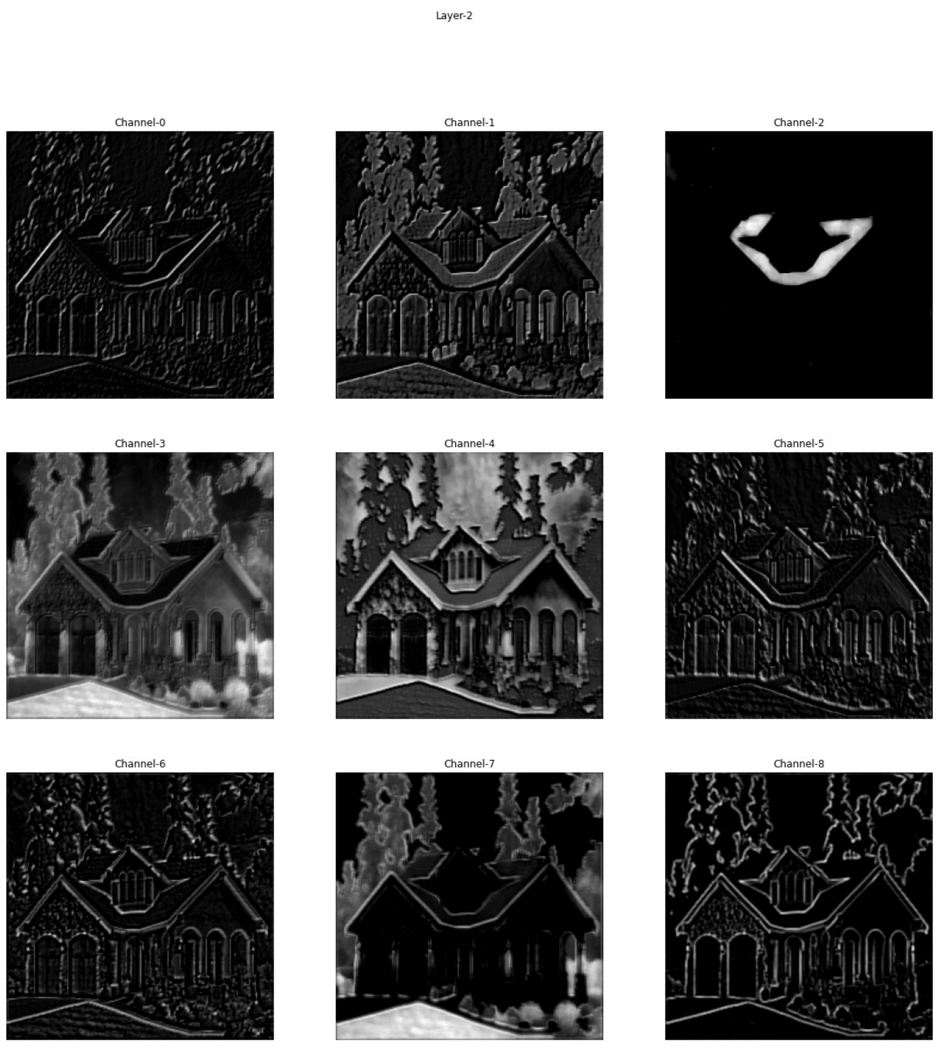

Figure 2.3: Output of Layer 2

2.1.3 Layer-5,6

In these layer picture has been distorted from the smoothing.

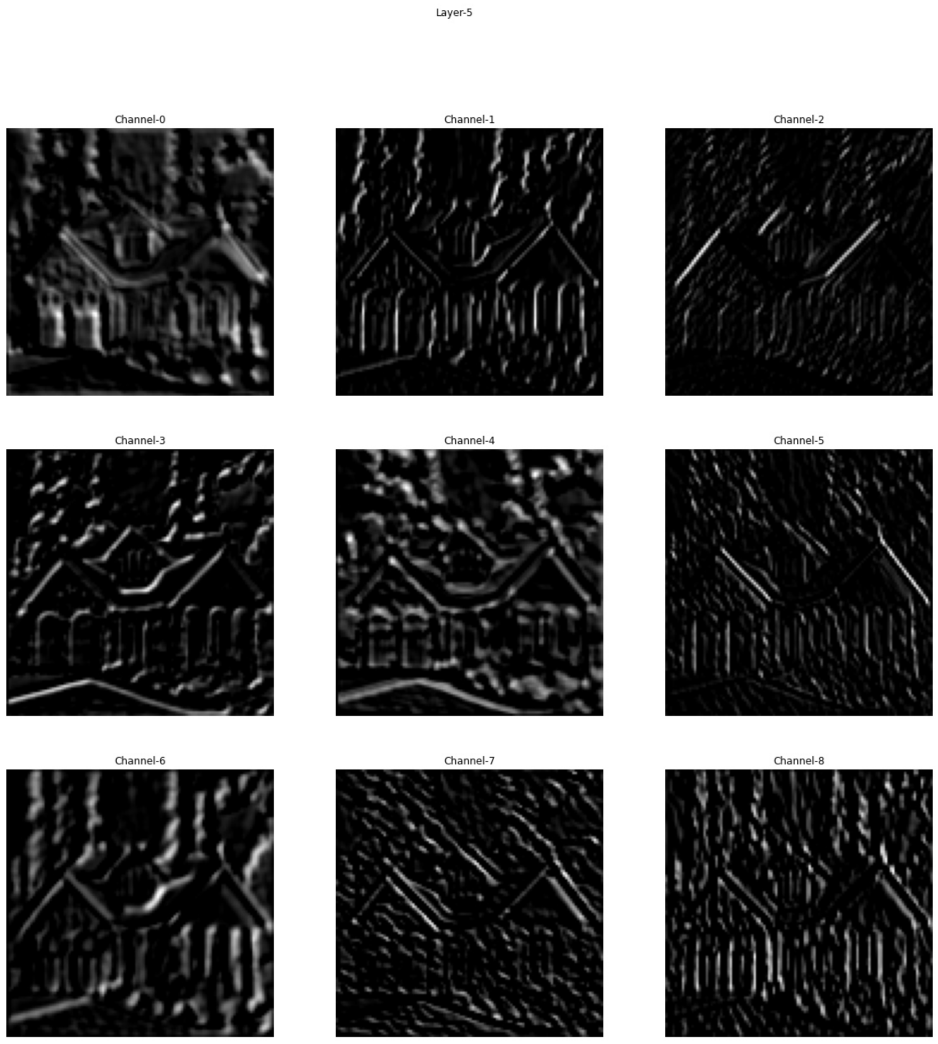

Figure 2.4: Output of Layer 5

2.1.4 Layer-7,8,9

In these layers each each part is separated.

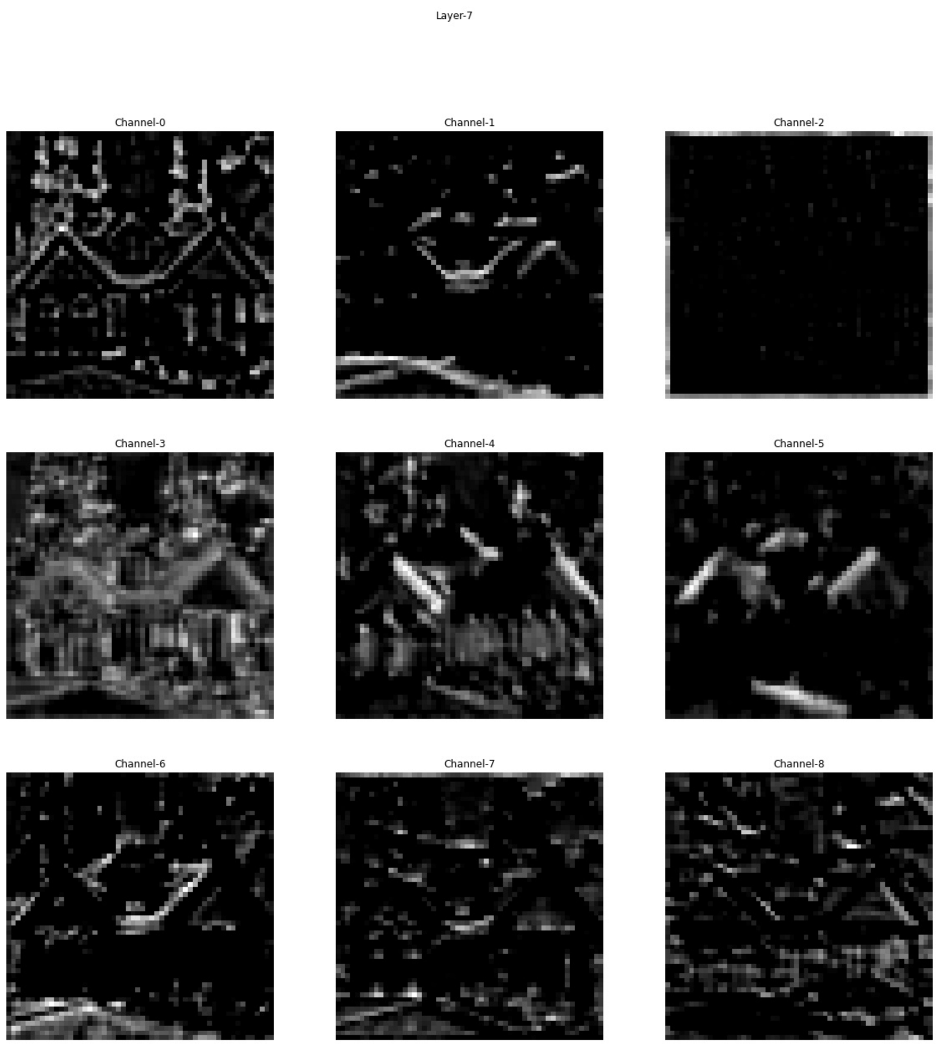

Figure 2.5: Output of Layer 7

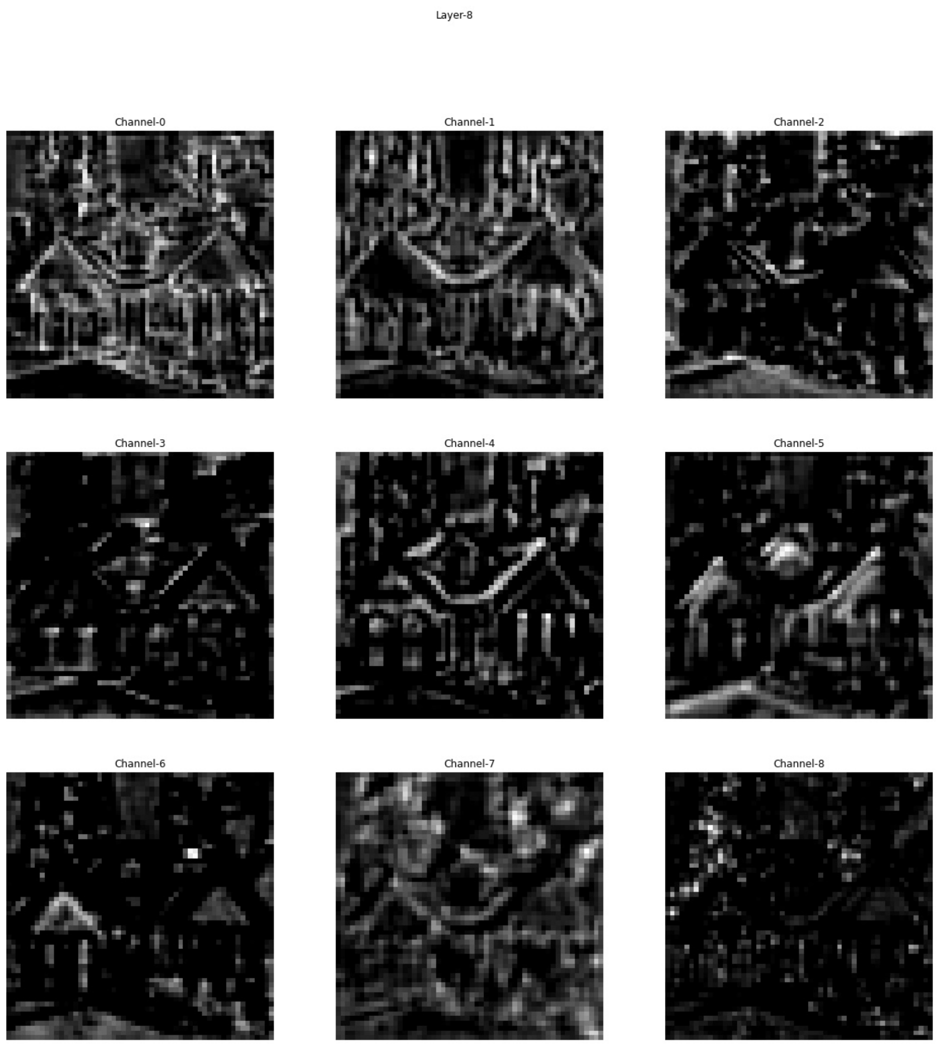

Figure 2.6: Output of Layer 8

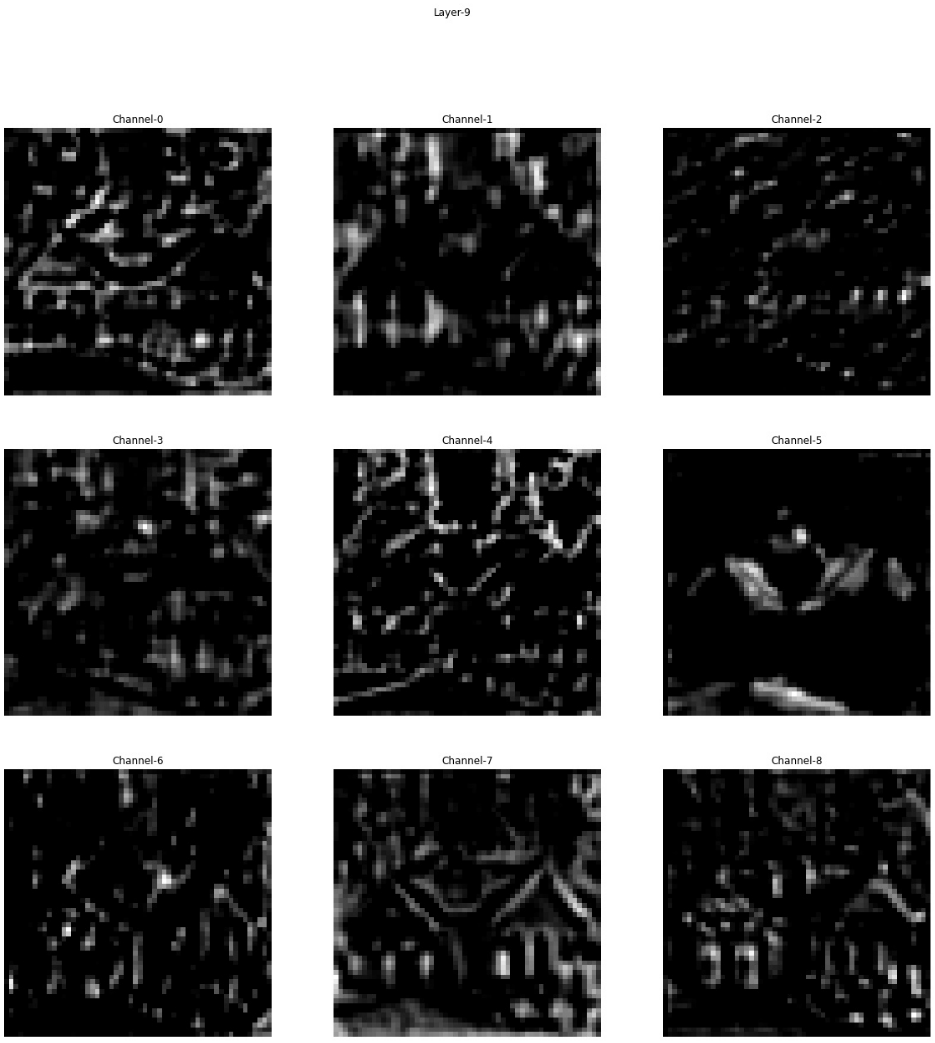

Figure 2.7: Output of Layer 9

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

3.1 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**

4 Assignment 14: Image compression in frequency domain

4.1 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.

4.2 Description

4.2.1 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)
```

4.2.2 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
```

4.2.3 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
```

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.

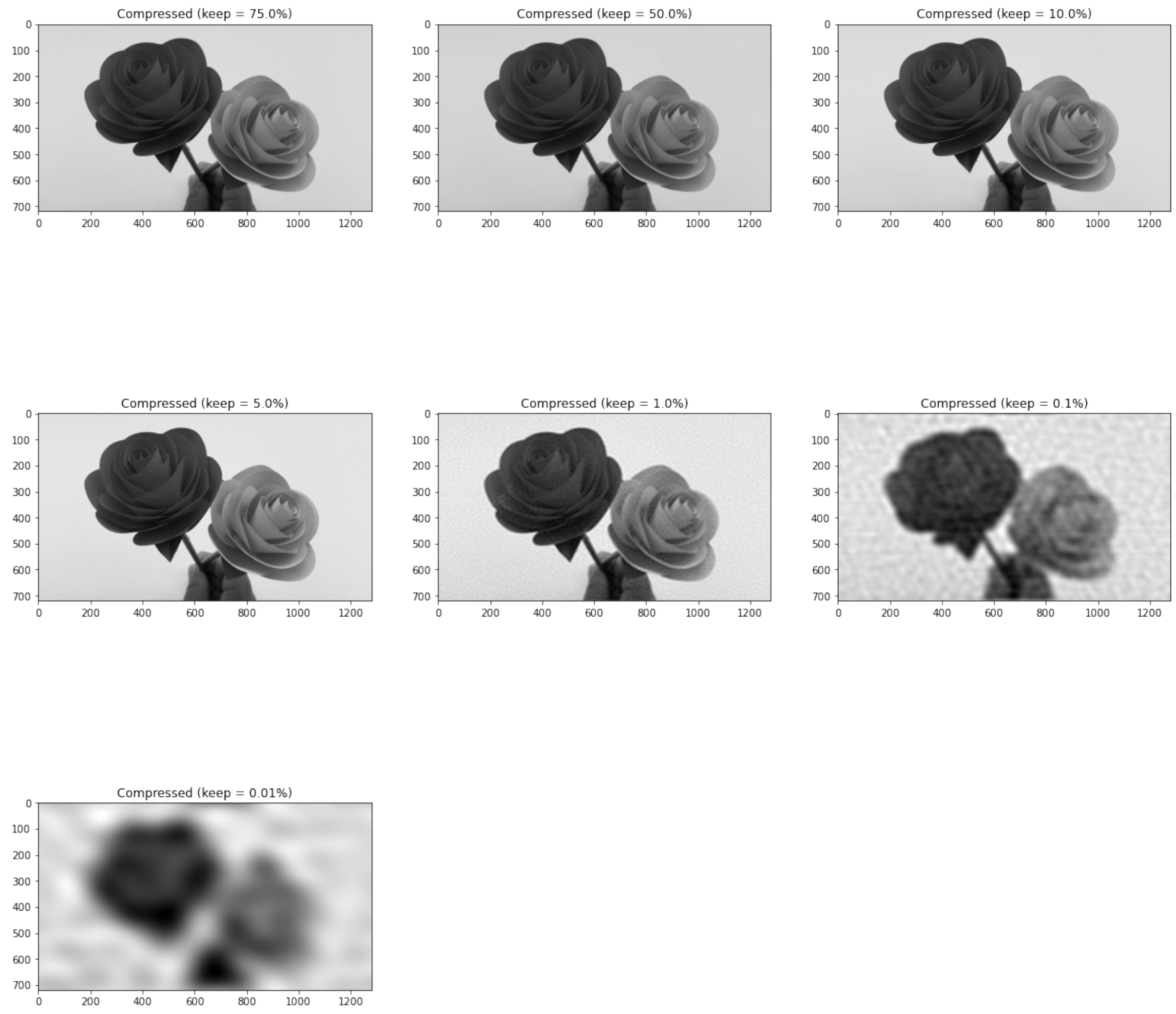
Compressed (keep = 75.0%)
Compressed (keep = 50.0%)
Compressed (keep = 10.0%)
Compressed (keep = 5.0%)
Compressed (keep = 1.0%)
Compressed (keep = 0.1%)
Compressed (keep = 0.01%)

Figure 4.1: compressed image

List of Figures