Computer Vision – Project1 Report

2016112083 김연웅

<Task 1 – Noise Estimation & Removal >

1. apply_xxx_filters

■ 코드 설명 - apply_average_filter(img, kernel_size)

```
def apply_average_filter(img, kernel_size):
    """
    You should implement average filter convolution algorithm in this function.
    It takes 2 arguments,
    'img' is source image, and you should perform convolution with average filter.
    'kernel_size' is a int value, which determines kernel size of average filter.

    You should return result image.
    """

    # variables initialization
    n_row, n_col, n_chan = img.shape
    padding_size = (kernel_size // 2)

    # create a kernel for average filter
    kernel = np.ones((kernel_size, kernel_size))
    kernel = kernel / (kernel_size * kernel_size)

    # create container for result image
    img_new = np.zeros([n_row, n_col, n_chan])

    # for every RGB channel in image
    for chan in range(n_chan):

        # pad zeros to current channel
        current_chan = img[:, :, chan]
        padded_channel = np.zeros((n_row + 2 * padding_size, n_col + 2 * padding_size))
        padded_channel[padding_size:padding_size + n_row, padding_size:padding_size + n_col] = current_chan

        # temporary container variable for result image
        filtered_padded_channel = padded_channel.copy()

        # for every pixels in image,
        for row in range(padding_size, padding_size + n_row):
            for col in range(padding_size, padding_size + n_col):

                # average convolution computation for given pixel
                convolution_area = padded_channel[row - padding_size:row + padding_size + 1, col - padding_size:col +
padding_size + 1]
                filtered_value = int((convolution_area * kernel).sum())
                filtered_padded_channel[row, col] = filtered_value

        # save the result of filtered channel to img_new
        img_new[:, :, chan] = filtered_padded_channel[padding_size:padding_size + n_row, padding_size:padding_size +
n_col]

    # type conversion and return
    img_new = img_new.astype('uint8')

    return img_new
```

- 커널 생성: 넘파이를 이용하여 kernel_size x kernel_size shape의 1로 채워진 매트릭스를 생성하고, 이 매트릭스 kernel_size의 제곱만큼 나누어서 average filter에 해당하는 커널을 구현했다.

- Zero-padding: convolution 연산을 위해 edge부분을 zero padding을 하는 전처리 과정을 수행했다. 넘파이를 이용해 기존 이미지보다 Kernel_size // 2 값만큼 더 큰 shape의 0으로 채워진 행렬을 생성했고 이를 다시 슬라이싱을 통해 패딩 구역이 아닌 곳에 이미지 값을 입혀주어 zero-padding과정을 구현했다.

- Convolution : convolution 연산을 수행하기 위해, 이미지의 각 RGB 채널, 가로사이즈, 세로사이즈에 해당하는 루프를 통해 각 픽셀을 접근했다. 각 픽셀에 접근한 후 kernel_size에 해당하는 크기의 matrix를 슬라이싱했고 이를 미리 생성해둔 kernel의 각 픽셀by 픽셀 곱셈 연산을 해서 convolution을 수행했다.

- 미리 생성해둔 컨테이너 목적의 이미지의 사이즈와 같은 img_new 행렬에 각 채널의 연산결과를 저장했고 이를 uint8로 type casting 한 이미지를 리턴했다.

■ 코드 실행, 결과 - apply_average_filter(img, kernel_size)

- **실행 코드**

```
40  img = cv2.imread("inputs/test1.png")
41  img_clean = cv2.imread("inputs/test1_clean.png")
42  print("rms between test1 image and test1_clean: ",calculate_rms_cropped(img, img_clean))
43
44  average = apply_average_filter(img, 3)
45  cv2.imwrite("outputs/check.png", average)
46  print("rms between average filtered test1 image and test1_clean: ",calculate_rms_cropped(average, img_clean))
47
48
```

테스트 이미지로는 inputs/test1.png를 사용했다. 해당 이미지를 apply_average_filter를 사용해 average filter를 적용시켰고 그 결과 이미지를 outputs/check.png에 저장했다. 커널 사이즈는 3이다.

또한 inputs/test1_clean.jpg와의 rms_error 수치를 util.py의 calculate_rms_cropped() 함수를 사용해서 측정했다.

- **실행 결과**

```
main ×
C:\Users\sodap\AppData\Local\Programs\Python\Python37\python.exe C:/Users/sodap/Desktop/2021-1/
rms between test1 image and test1_clean:  28.646127348260023
rms between average filtered test1 image and test1_clean:  16.523674663583378

Process finished with exit code 0
```

Check.png와 test1_clean.png의 이미지 간의 rms boundary가 과제 스펙의 base line 수치에 맞는 것을 확인했다.

- **Inputs/test1.png 와 inputs/test1_clean.png**


- **Outputs/check.png (Average filter를 거친 test1의 이미지)**

■ **코드 설명 - apply_median_filter(img, kernel_size)**

```
def apply_median_filter(img, kernel_size):
    """
    You should implement median filter convolution algorithm in this function.
    It takes 2 arguments,
    'img' is source image, and you should perform convolution with median filter.
    'kernel_size' is a int value, which determines kernel size of median filter.

    You should return result image.
    """

    # variables initialization
    n_row, n_col, n_chan = img.shape
    padding_size = (kernel_size // 2)

    # create container for result image
    img_new = np.zeros([n_row, n_col, n_chan])

    # for every RGB channel in image
    for chan in range(n_chan):

        # pad zeros to current channel
        current_chan = img[:, :, chan]
        padded_channel = np.zeros((n_row + 2 * padding_size, n_col + 2 * padding_size))
        padded_channel[padding_size:padding_size + n_row, padding_size:padding_size + n_col] = current_chan

        # temporary container variable for result image
        filtered_padded_channel = padded_channel.copy()

        # for every pixels in image,
        for row in range(padding_size, padding_size + n_row):
            for col in range(padding_size, padding_size + n_col):

                # median convolution computation for given pixel
                convolution_area = padded_channel[row - padding_size:row + padding_size + 1, col - padding_size:col +
padding_size + 1]
                filtered_value = np.median(convolution_area)
                filtered_padded_channel[row, col] = filtered_value

        # save the result of filtered channel to img_new
        img_new[:, :, chan] = filtered_padded_channel[padding_size:padding_size + n_row, padding_size:padding_size +
n_col]

    # type conversion and return
    img_new.astype('uint8')

    return img_new
```

- 커널 생성: 중앙값을 구하는 numpy.median 함수를 사용하므로 따로 구현하지 않았다.
- Zero-padding: edge부분을 zero padding을 하는 전처리 과정을 수행했다. 넘파이를 이용해 기존 이미지보다 Kernel_size // 2 값만큼 더 큰 shape의 0으로 채워진 행렬을 생성했고 이를 다시 슬라이싱을 통해 패딩 구역이 아닌 곳에 이미지 값을 입혀주어 zero-padding과정을 구현했다.

- Convolution: 이미지의 각 RGB 채널, 가로사이즈, 세로사이즈에 해당하는 루프를 통해 각 픽셀을 접근했다. 각 픽셀에 접근한 후 kernel_size에 해당하는 크기의 matrix를 슬라이싱 했다. Np.median() 라이브러리를 이용해 해당 슬라이싱한 matrix의 element들 median값을 찾아냈고 이를 새로운 이미지의 해당 픽셀 값으로 저장해주었다.
- 미리 생성해둔 컨테이너 목적의 이미지의 사이즈와 같은 img_new 행렬에 각 채널의 연산결과를 저장했고 이를 uint8로 type casting 한 이미지를 리턴했다.

■ 코드 실행, 결과 - apply_median_filter(img, kernel_size)

- **실행 코드**

```
40  img = cv2.imread("inputs/test4.png")
41  img_clean = cv2.imread("inputs/test4_clean.png")
42  print("rms between test4 image and test4_clean: ",calculate_rms_cropped(img, img_clean))
43
44  median = apply_median_filter(img, 3)
45  cv2.imwrite("outputs/check.png", median)
46  print("rms between median filtered test4 image and test4_clean: ",calculate_rms_cropped(median, img_clean))
47
```

테스트 이미지로는 inputs/test4.png를 사용했다. 해당 이미지를 apply_median_filter를 사용해 median filter를 적용시켰고 그 결과 이미지를 outputs/check.png에 저장했다. 커널 사이즈는 3이다.

또한 inputs/test4_clean.jpg와의 rms_error 수치를 util.py의 calculate_rms_cropped() 함수를 사용해서 측정했다.

- **실행 결과**

```
main ×
C:\Users\sodap\AppData\Local\Programs\Python\Python37\python.exe C:/Users/sodap/Desktop/2021-1/컴전/
rms between test4 image and test4_clean:  44.73770267861831
rms between median filtered test4 image and test4_clean:  21.02211053524199

Process finished with exit code 0
```

Check.png와 test4_clean.png의 이미지 간의 rms boundary가 과제 스펙의 base line 수치에 딱 맞는 것을 확인했다.

- **Inputs/test4.png 와 inputs/test4_clean.png**


- **Outputs/check.png (Median filter를 거친 test4의 이미지)**

■ 코드 설명 - apply_bilateral_filter(img, kernel_size, sigma_s, sigma_r)

```
def apply_bilateral_filter(img, kernel_size, sigma_s, sigma_r):
    """
    You should implement convolution with additional filter.
    You can use any filters for this function, except average, median filter.
    It takes at least 2 arguments,
    'img' is source image, and you should perform convolution with median filter.
    'kernel_size' is a int value, which determines kernel size of average filter.
    'sigma_s' is a int value, which is a sigma value for G_s
    'sigma_r' is a int value, which is a sigma value for G_r
    You can add more arguments for this function if you need.
    You should return result image.
    """

    # Gaussian Function
    def gaussian(x, sigma):
        return (1.0 / (2 * np.pi * (sigma ** 2))) * np.exp(- (x ** 2) / (2 * sigma ** 2))

    # Distance Function
    def distance(x1, y1, x2, y2):
        return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

    # variables initialization
    n_row, n_col, n_chan = img.shape
    img_new = np.zeros([n_row, n_col, n_chan])

    # for every RGB channel in image
    for chan in range(n_chan):

        # variables initialization
        current_chan = img[:, :, chan]
        filtered_chan = np.zeros([n_row, n_col])

        # for every pixel in an image
        for col in range(n_col):
            for row in range(n_row):
                wp = 0                      # normalization wp
                filtered_value_sum = 0      # sum of Gs x Gr x Iq in convolution area

                # for every pixels in convolution kernel
                for col_kernel in range(-(kernel_size // 2), (kernel_size // 2) + 1):
                    for row_kernel in range(-(kernel_size // 2), (kernel_size // 2) + 1):

                        # calculate current pixel in convolution area
                        col_fPoint = col + col_kernel
                        row_fPoint = row + row_kernel

                        # Only when current pixel coordinates is not out of index, do the convolution calculation
                        if (col_fPoint >= 0) and (col_fPoint < n_col) and (row_fPoint >= 0) and (row_fPoint < n_row):

                            # compute gaussian spatial/range values in this point
                            gaussian_spatial = gaussian(distance(row_fPoint, col_fPoint, row, col), sigma_s)
                            gaussian_range = gaussian(abs(int(current_chan[row_fPoint][col_fPoint]) -
int(current_chan[row][col])), sigma_r)

                            w = gaussian_spatial * gaussian_range                                   # w = Gs x Gr
                            filtered_value_sum += current_chan[row_fPoint][col_fPoint] * w          # w * Iq
                            wp += w                                                                 # wp (variable for
normalization)
```

```
            filtered_value_norm = filtered_value_sum / wp                    # normalization

            # save result of computation for corresponding pixel
            filtered_chan[row][col] = int(np.round(filtered_value_norm))

    # save computation result of channel
    img_new[:, :, chan] = filtered_chan

# type conversion and return
img_new = img_new.astype('uint8')

return img_new
```

- apply_bilateral_filter 은 앞선 두 필터 구현(average, median)과 달리 인풋 이미지의 모든 픽셀과 그 픽셀에 해당하는 convolution area의 모든 픽셀에 접근하여 픽셀의 모든 값을 각각 연산하는 bruteforce의 형태 구현했다. Convolution을 행렬간의 곱으로 연산하는 위의 두 함수에 비해 연산 량이 많고, 수행에 오랜 시간이 소요된다.

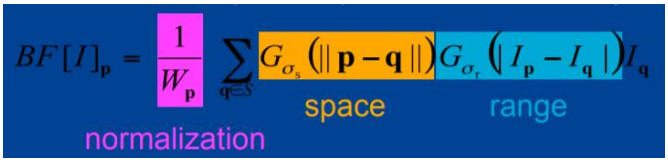
- sub함수 gaussian(x, sigma): x를 인풋으로 받고, 해당하는 gaussian value를 리턴한다. Gaussian Spatial과 Gaussian Range를 구할 때 쓰인다. Gaussian Spatial의 경우, 아래의 distance함수를 이용하여 해당하는 현재 픽셀과 neighboring 좌표의 거리를 파라미터로 넘긴다. Gaussian Range의 경우 해당하는 현재 이미지의 픽셀 좌표와 neighboring 좌표간의 value 차의 절댓값을 x 파라미터로 넘긴다.

- sub함수 distance(x1, y1, x2, y2): Gs(Gaussian Spatial)을 계산할 때 쓰이는 distance 함수이다. 두 픽셀의 좌표를 인풋으로 받고, 두 좌표의 거리를 리턴한다.

- 앞서 설명한 두 필터구현과 달리 bilateral filter에서는 이미지에 대한 zero-padding 전처리를 수행하지 않았다.(rms_boundary의 오차를 줄이려는 목적에서 하지 않았으나, 이후, calculate_rms_crop 평가함수가 공개되었고 rms 값에 큰 차이가 없어졌다.)

- 마찬가지로, 모든 RGB채널, 가로사이즈, 세로사이즈 만큼의 루프를 돌려 이미지의 각 픽셀에 접근했다. 해당 픽셀에서는 다시 이중 포문을 통해 kernel_size에 해당하는 neighboring 픽셀들에 접근했다.

- Bilateral filtering 의 convolution 연산 수행은 다음과 같다. 모든 이미지의 픽셀과, 각 이미지 픽셀의 kernel size 에 해당하는 neighboring pixel 들에 대해 다음과 같은 연산을 수행했다.



(Bilateral Filter)

(Where Wp is)

1) Gaussian Spatial (Gs): neighboring 픽셀과 현재 픽셀 간의 거리(||p-q||)를 distance함수를 통해 구하고 그 값을 gaussian함수에 파라미터로 주어 해당 픽셀에 대한 Gaussian spatial값을 구했다.

2) Gaussian range(Gr): 마찬가지로 neighboring 픽셀과 현재 픽셀의 value값 차의 절대값(| Ip – Iq |)을 gaussian 함수에 파라미터로 주어 해당 픽셀에 대한 Gaussian Range 값을 구했다.

3) W: 이후 현재 픽셀의 value(Iq)를 Gs, Gr과 곱해준 값이 w이고 이 w를 kernel size에 해당하는 모든 neighboring pixel들에 대하여 구해주고, 그 값의 합을 filtered_value_sum에 넣어주었다.

4) WP: 모든 neighboring 픽셀과 현재 픽셀과의 Gs, Gr값의 곱이다. Normalization factor로 쓰인다.

5) 모든 neighboring 픽셀들에 대해 연산이 끝난 후 filtered_value_sum을 WP로 나누는 nomalization을 거친 값을 새로운 이미지의 해당 픽셀 값으로 저장했다.

- 마지막으로, 앞선 두 필터의 구현과 마찬가지로, 채널들을 합쳐 새로운 이미지를 저장하고, type casting 이후 리턴했다.

■ 코드 실행, 결과 - apply_bilateral_filter(img, kernel_size, sigma_s, sigma_r)

- **실행 코드**

```
40  img = cv2.imread("inputs/test2.png")
41  img_clean = cv2.imread("inputs/test2_clean.png")
42  print("rms between test2 image and test2_clean: ",calculate_rms_cropped(img, img_clean))
43
44  bilateral = apply_bilateral_filter(img, 3, 100, 30)
45  cv2.imwrite("outputs/check.png", bilateral)
46  print("rms between bilateral filtered test2 image and test2_clean: ",calculate_rms_cropped(bilateral, img_clean))
```

테스트 이미지로는 inputs/test42png를 사용했다. 해당 이미지를 apply_bilateral_filter를 사용해 bilateral filter를 적용시켰고 그 결과 이미지를 outputs/check.png에 저장했다. 커널 사이즈는 3이다. Sigma_s는 100, sigma_r 는 30 이다.

또한 inputs/test2_clean.jpg와의 rms_error 수치를 util.py의 calculate_rms_cropped() 함수를 사용해서 측정했다.

- **실행 결과**

```
main ×
C:\Users\sodap\AppData\Local\Programs\Python\Python37\python.exe C:/Users/sodap/Desktop/2021-1/컴젼/Project1(
rms between test2 image and test2_clean:  19.32460308186706
rms between bilateral filtered test2 image and test2_clean:  10.361853544634124

Process finished with exit code 0
```

Check.png와 test2_clean.png의 이미지 간의 rms boundary가 과제 스펙의 base line 수치에 맞는 것을 확인했다.

- **Inputs/test2.png 와 inputs/test2_clean.png**


- **Outputs/check.png (Bilateral filter를 거친 test2의 이미지)**

2. Task1(src_img_path, clean_img_path, dst_img_path)

■ 코드 설명 – task1(src_img_path, clean_img_path, dst_img_path)

```
def task1(src_img_path, clean_img_path, dst_img_path):
    """
    This is main function for task 1.
    It takes 3 arguments,
    'src_img_path' is path for source image.
    'clean_img_path' is path for clean image.
    'dst_img_path' is path for output image, where your result image should be saved.

    You should load image in 'src_img_path', and then perform task 1 of your assignment 1,
    and then save your result image to 'dst_img_path'.
    """

    noisy_img = cv2.imread(src_img_path)
    clean_img = cv2.imread(clean_img_path)
    result_img = None
    rms_min = 100
    # do noise removal


    filtered1 = apply_average_filter(noisy_img, 3)
    if calculate_rms_cropped(filtered1, clean_img) < rms_min:
        rms_min = calculate_rms_cropped(filtered1,clean_img)
        result_img = filtered1

    filtered2 = apply_median_filter(noisy_img,3)
    if calculate_rms_cropped(filtered2, clean_img) < rms_min:
        rms_min = calculate_rms_cropped(filtered2,clean_img)
        result_img = filtered2

    filtered3 = apply_bilateral_filter(noisy_img,3,100,30)
    if calculate_rms_cropped(filtered3, clean_img) < rms_min:
        rms_min = calculate_rms_cropped(filtered3,clean_img)
        result_img = filtered3

    filtered4 = apply_bilateral_filter(noisy_img,5,100,30)
    if calculate_rms_cropped(filtered4, clean_img) < rms_min:
        rms_min = calculate_rms_cropped(filtered4,clean_img)
        result_img = filtered4


    cv2.imwrite(dst_img_path, result_img)
```

- task1 함수는 input 이미지 path, input-clean path 이미지, 그리고 destination path를 파라미터로 받고 앞서 구현한 세가지 필터를 이용해 이미지를 de-noise한다. 이후 utils.py의 calculate_rms_crop함수를 이용해 필터링을 한 이미지와 clean 이미지간의 rms를 측정하고 가장 낮은 rms 수치를 기록한 이미지를 dest_img_path에 이미지 파일로 저장한다.

- 사전에 구현한 필터들을 이용해 이미지 별로 baseline에 도달하는 필터들의 조합을 찾아냈다. Task1함수는 미리 실험한 파라미터 세팅된 필터들을 인풋이미지에 적용하고 필터링된 이미지를 dest_img_path에 파일로 저장한다.

■ Task1에서 적용하는 필터들과 필터링된 이미지와 clean 이미지간의 rms error 수치는 다음과 같다.
(해당 수치들은 모두 base라인을 통과하는 조합이다)

Inputs/test1.png ->	apply_average_filter(img, 3)	rms: 16.52367
inputs/test2.png ->	apply_bilateral_filter(img, 3, 100, 30)	rms: 10.36185
(둘다가능)	apply_bilateral_filter(img, 5, 100, 30)	rms: 11.19628
inputs/test3.png ->	apply_bilateral_filter(img, 5, 100, 30)	rms: 3.54152
inputs/test4.png ->	apply_median_filter(img, 3)	rms: 21.02211
inputs/test5.png ->	apply_median_filter(img, 3)	rms: 12.28968

- 프로젝트 스펙의 rms baseline 수치:

Test 1 : 16.5315

Test 2 : 11.3346

Test 3 : 3.8201

Test 4 : 12.0583

Test 5 : 6.6090

- **결과 분석**

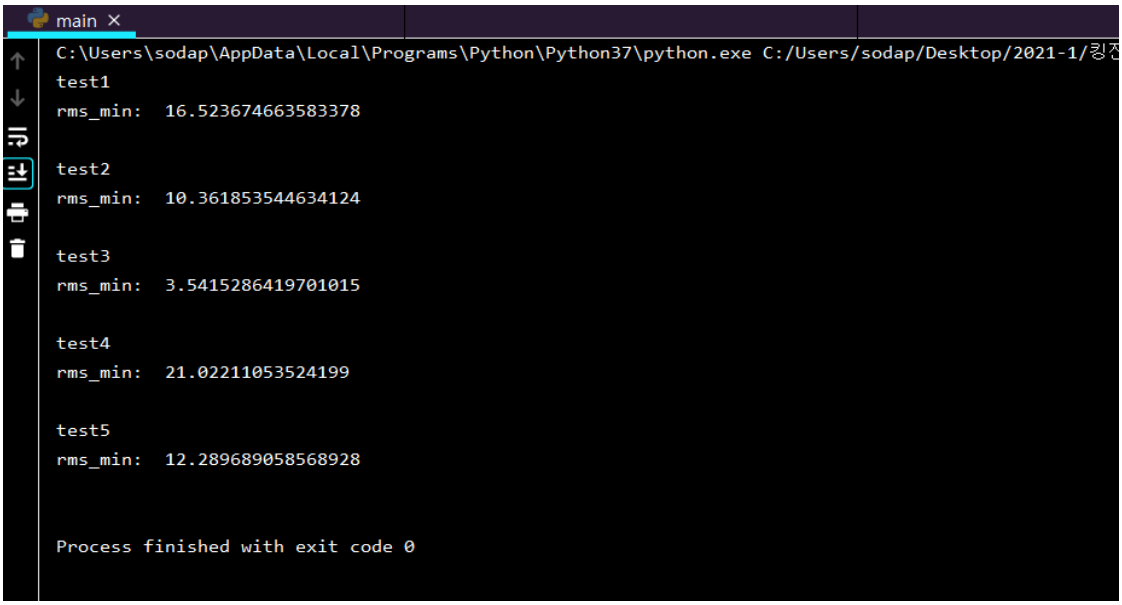
테스트 이미지는 크게 두가지 종류로 나눌 수 있다. Test1, test2는 gaussian noise가 적용된 이미지라 추측되며, test3, test4, test5는 salt and pepper noise가 적용된 이미지라 예상된다. 때문에 task1함수의 실행 결과에서도 test1, test2의경우 average filter와 gaussian filter가 최적의 필터로 선택되었고, test3, test4, test5의 경우 median filter가 최적의 필터로 선택되었다. 이는 gaussian noise를 제거하는 데에 bilateral, average filter가 우수하고, salt and pepper noise를 제거하는 데 median filter가 우수한 성능을 보인다는 수업시간의 내용과 일치한다.

■ 코드 실행, 결과 – task1(src_img_path, clean_img_path, dst_img_path)

- **실행코드**

```
from task1.noise import *
from task1.utils import *


task1("inputs/test1.png", "inputs/test1_clean.png", "outputs/test1.png")
outputs = cv2.imread("outputs/test1.png")
img_clean = cv2.imread("inputs/test1_clean.png")
print("test1")
print("rms_min: ", calculate_rms_cropped(outputs, img_clean))
print()

task1("inputs/test2.png", "inputs/test2_clean.png", "outputs/test2.png")
outputs = cv2.imread("outputs/test2.png")
img_clean = cv2.imread("inputs/test2_clean.png")
print("test2")
print("rms_min: ", calculate_rms_cropped(outputs, img_clean))
print()

task1("inputs/test3.png", "inputs/test3_clean.png", "outputs/test3.png")
outputs = cv2.imread("outputs/test3.png")
img_clean = cv2.imread("inputs/test3_clean.png")
print("test3")
print("rms_min: ", calculate_rms_cropped(outputs, img_clean))
print()

task1("inputs/test4.png", "inputs/test4_clean.png", "outputs/test4.png")
outputs = cv2.imread("outputs/test4.png")
img_clean = cv2.imread("inputs/test4_clean.png")
print("test4")
print("rms_min: ", calculate_rms_cropped(outputs, img_clean))
print()

task1("inputs/test5.png", "inputs/test5_clean.png", "outputs/test5.png")
outputs = cv2.imread("outputs/test5.png")
img_clean = cv2.imread("inputs/test5_clean.png")
print("test5")
print("rms_min: ", calculate_rms_cropped(outputs, img_clean))
print()
```

- **실행 결과**



테스트 결과, 모든 테스트 케이스의 rms error가 과제 스펙의 baseline안에 듬을 확인했다.

< Task2 – Fourier Transform >

1. fm_spectrum(img)

■ **코드 설명 – fm_spectrum(img)**

```python
def fm_spectrum(img):
    """
    Get frequency magnitude spectrum image of input image.
    Returning spectrum image is shifted to center
    Input image should be gray scaled

    :param img: input image
    :return: frequency magnitude spectrum
    """

    # fourier transform 2d using numpy library
    f = np.fft.fft2(img)

    # variables initialization
    n_row, n_col = f.shape
    temp = np.zeros(f.shape, dtype='complex')

    # for every pixel in a frequency domain of input image,
    #   [       |       ]
    #   [   (1) |  (2)   ]
    #   [-----------------]
    #   [   (3) |  (4)   ]
    #   [       |       ]
    # swaps the first quadrant (1) with the fourth(4) and the second quadrant (2) with the third(3).
    for row in range(n_row):
        for col in range(n_col):

            # swap first quadrant with the fourth
            if row < n_row // 2 and col < n_col // 2:
                temp[row + n_row // 2][col + n_col // 2] = f[row][col]
            # swap second quadrant with the third
            elif row < n_row // 2 and col > n_col // 2:
                temp[row + n_row // 2][col - n_col // 2] = f[row][col]
            # swap third quadrant with the second
            elif row > n_row // 2 and col < n_col // 2:
                temp[row - n_row // 2][col + n_col // 2] = f[row][col]
            # swap fourth quadrant with the first
```

```
            else:
                temp[row - n_row // 2][col - n_col // 2] = f[row][col]

    # get the spectrum and return
    spectrum = np.log(1 + np.abs(temp.real))
    return spectrum
```

- 이미지를 인풋으로 받고 fourier transform을 거친 후, 저주파 영역을 중앙으로 옮긴 후 magnitude화 된 spectrum을 아웃풋으로 리턴한다.
- np.fft.fft2 넘파이 라이브러리 함수를 이용해 먼저 fourier transfrom을 수행했다.
- 이후 np.fft.fftshift 기능을 직접 구현했다.

```
[       |       ]
[  (1)  |  (2)  ]
[-------------------]
[  (3)  |  (4)  ]
[       |       ]
```

fourier transform을 거친 spectrum의 1사분면과 4사분면, 2사분면과 3사분면을 서로 swap해서 center shift를 구현했다. 이후 형변환과 magnitude 과정을 거친 magnitude spectrum을 결과로 반환한다.

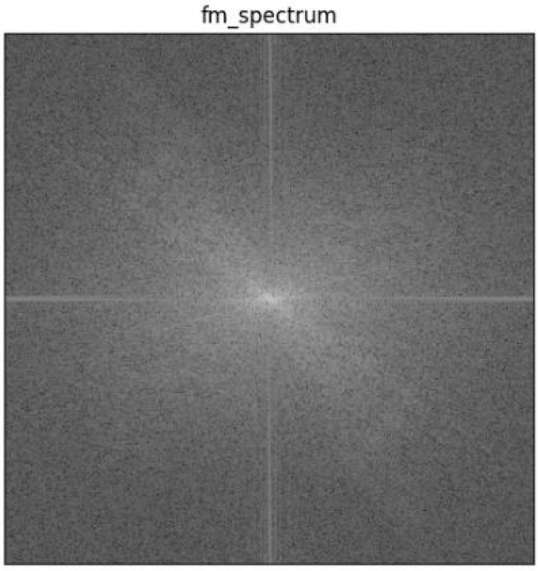
■ 코드 실행, 결과 - fm_spectrum(img)

- **실행 코드**

```
5   from task2.fourier import *
6
7   img = cv2.imread('task2_sample.png',0)
8
9
0   plt.subplot(111),plt.imshow(img, cmap='gray')
1   plt.title('original Image'), plt.xticks([]), plt.yticks([])
2   plt.show()
3   plt.subplot(111),plt.imshow(fm_spectrum(img), cmap='gray')
4   plt.title('fm_spectrum'), plt.xticks([]), plt.yticks([])
5   plt.show()
6
```

Pyplot 라이브러리를 사용해 이미지를 시각화했다.

- **실행 결과**



2. High_pass_filter(img, th=30)

■ 코드 설명 – High_pass_filter(img, th=30)

```
def high_pass_filter(img, th=30):
    """
    Get filtered image that pass through with high-pass filter for an input image.
    :param img: input image
    :param th:
    :return:
    """
    # distance function
    def distance(x1, y1, x2, y2):
        return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

    # get magnitude spectrum using fftshift_diy
    spectrum = fftshift_diy(img)

    # variables initialization
    n_row, n_col = img.shape
    center_row, center_col = n_row / 2, n_col / 2

    # create high-pass filter
    filter = np.ones((n_row, n_col))

    # create the high pass filter - for every pixel in the filter,
    for row in range(n_row):
        for col in range(n_col):

            # if the distance between center point and current point is smaller than th parameter,
            # set current point value to  0, which means reject
            if distance(row, col, center_row, center_col) < th:
                filter[row, col] = 0

    # mask the filter on spectrum
    spectrum = spectrum * filter

    f_ishift = ifftshift_diy(spectrum)
    img_filtered = np.fft.ifft2(f_ishift)

    # Reform the image from high pass filtered spectrum and return the result image
    return img_filtered.real
```

- high pass filter는 저주파 영역의 정보만 남기고 고주파 영역의 정보를 non pass 시키는 필터이다. 파라미터로 input image를 받고,

threshold값을 설정할 수 있다.

- Threshold 값을 반지름으로, magnitude spectrum의 중앙지점을 원점으로 원을 경계로 안쪽의 지점들을 pass시키고 이외의 지역은 모두 0값으로 설정한다. 기본 설정값은 30이다.
- sub함수_distance: 두 좌표를 파라미터로 받고, 두 좌표간의 거리를 반환한다. 원의 영역을 계산할 때 쓰인다.
- 먼저 fftshift_diy함수를 활용해 이미지의 center shifted된 spectrum을 받아낸다. 이후 이미지의 shape에 해당하는 1으로 채워진 행렬을 생성하고, 중앙 지점으로부터 th을 경계로 하는 원 이내의 해당하는 픽셀들을 0로 설정한다. 그리고 필터를 spectrum에 곱하여 high pass filtered된 이미지를 받고 이 spectrum을 다시 역변환 과정을 거쳐 결과 이미지를 반환한다.

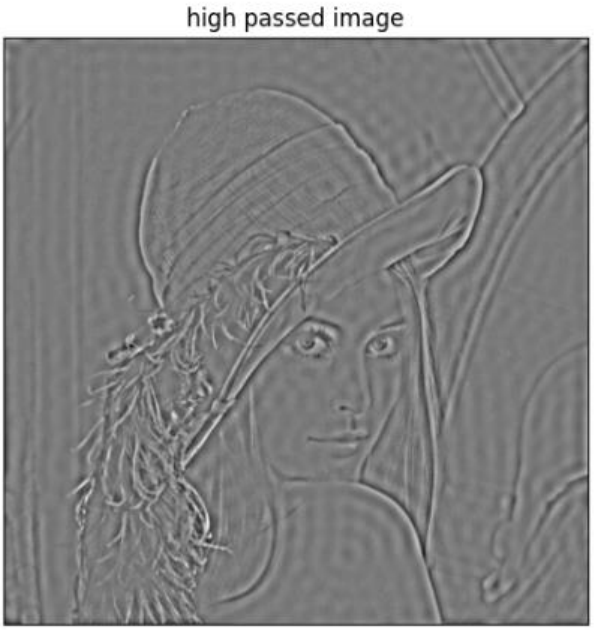
특이사항 -> return을 할 때 ifft2를 통해 역변환한 이미지의 픽셀 값들의 type은 복소수이다. 처음에는 복소수를 실수로 복원할 때, np.abs()함수를 사용해 구현했다. 그러나 np.abs()함수로 복원된 이미지는 다시 fm_spectrum을 적용하면 본래의 low passed filter가 적용된 스펙트럼을 얻을 수 없었다. 그래서 np.abs() 대신 절대값을 취하지 않고 실수를 반환하는 np.real를 사용해 최종 이미지를 반환했다. 밑의 코드 수행 결과 파트에서 더 해당 문제에 대한 이미지들을 설명과 함께 첨부했다.

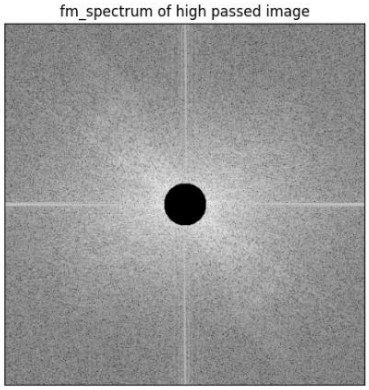
■ 코드 실행, 결과 - High_pass_filter(img, th=30)

- **실행 코드**

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

from task2.fourier import *

img = cv2.imread('task2_sample.png',0)
high_passed = high_pass_filter(img)


plt.subplot(111),plt.imshow(img, cmap='gray')
plt.title('original Image'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(high_passed, cmap='gray')
plt.title('high passed image'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(fm_spectrum(high_passed), cmap='gray')
plt.title('fm_spectrum of high passed image'), plt.xticks([]), plt.yticks([])
plt.show()
```

- **실행 결과**





위의 코드는 최종 역변환된 image의 결과물을 np.real을 통해 복원한 이미지이다. 결과 이미지가, 스펙의 예시처럼 회색 바탕의 색감을 가지고 있고, high passed image를 다시 fm_spectrum으로 fourier transform을 하면 스펙트럼의 가운데 저주파 부분이 검정색으로 reject되어있음을 확인할 수 있다.

하지만 np.abs()함수를 사용해 최종 결과 이미지를 복원하면 결과는 다르다.

■ Np. Abs()함수를 사용해 반환한 결과

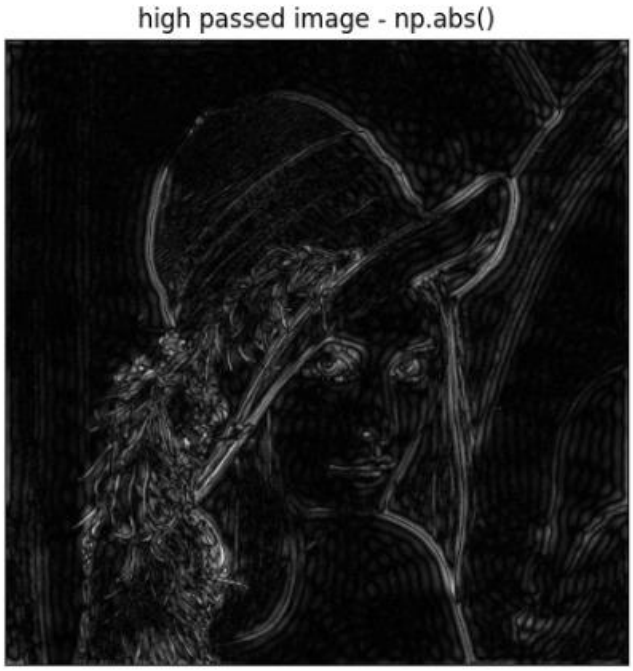
- **코드 high_pass_filter(img, th=30)**

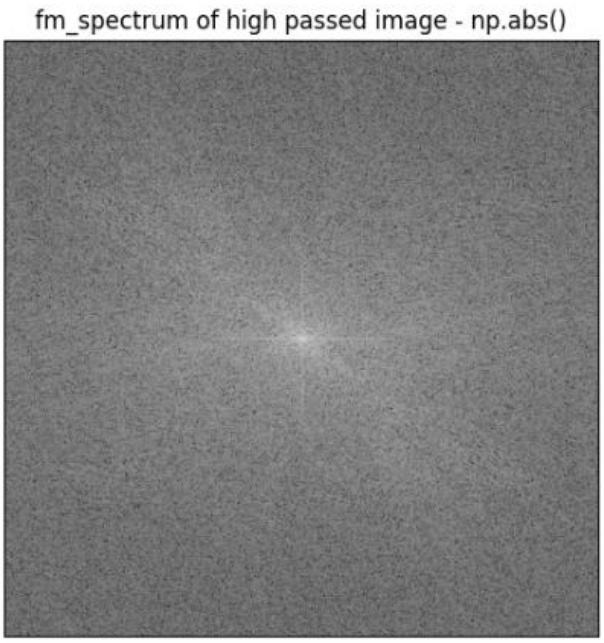
```
def high_pass_filter(img, th=30):
    """
    Get filtered image that pass through with high-pass filter for an input image.
    :param img: input image
    :param th:
    :return:
    """
    # distance function
    def distance(x1, y1, x2, y2):
        return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

    # get magnitude spectrum using fftshift_diy
    spectrum = fftshift_diy(img)

    # variables initialization
    n_row, n_col = img.shape
    center_row, center_col = n_row / 2, n_col / 2

    # create high-pass filter
    filter = np.ones((n_row, n_col))

    # create the high pass filter - for every pixel in the filter,
    for row in range(n_row):
        for col in range(n_col):

            # if the distance between center point and current point is smaller than th parameter,
            # set current point value to  0, which means reject
            if distance(row, col, center_row, center_col) < th:
                filter[row, col] = 0

    # mask the filter on spectrum
    spectrum = spectrum * filter

    f_ishift = ifftshift_diy(spectrum)
    img_filtered = np.fft.ifft2(f_ishift)

    # Reform the image from high pass filtered spectrum and return the result image
    return np.abs(img_filtered)
```

- **실행 코드**

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

from task2.fourier import *

img = cv2.imread('task2_sample.png',0)
high_passed = high_pass_filter(img)


plt.subplot(111),plt.imshow(img, cmap='gray')
plt.title('original Image'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(high_passed, cmap='gray')
plt.title('high passed image - np.abs()'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(fm_spectrum(high_passed), cmap='gray')
plt.title('fm_spectrum of high passed image - np.abs()'), plt.xticks([]), plt.yticks([])
plt.show()
```

- **실행 결과**





실행 결과, 과제 스펙에서 나오는 것과 같은 회색 바탕의 이미지가 아니라 검정색 바탕의 이미지가 산출되었다. 또한 결과 이미지를 fm_spectrum을 통해 다시 스펙트럼화하면 np.real을 사용했을때처럼 원래 이미지의 스펙트럼에서 low frequency가 제거된(필터가 적용된) 스펙트럼이 아닌 임의의 스펙트럼이 복원되었다.

→ Np.abs() VS np.real

과제를 하며, 최종 이미지 복원에서 np.abs()를 사용할지, np.real를 사용할지 많은 고민을 했다. 결국 제출할 때 선택한 함수는 np.real이고 그 이유는 다음과 같다.

1. Np.real 적용시 반환되는 이미지가 과제 스펙의 예시 이미지와 거의 유사하다.
2. 이론상 반환된 이미지를 다시 스펙트럼화 하면 본래 이미지의 스펙트럼에, low frequency part가 제거된 형태의 스펙트럼이 형성되는게 맞다.
3. 테스트 결과, np.abs()로 리턴한 이미지는 cv2.imwrite함수로 png파일 생성 및 저장이 가능했으나, np.real로 리턴한 이미지는 cv2.imwrite 함수로 png파일 생성 및 저장하는 기능이 작동하지 않았다. 본 과제에서 요구하는 denoised1.png, denoised2.png 파일들을 생성하기 위해 np.abs()로 수정해 파일을 생성했고 제출은 다시 np.real로 바꾸어 제출했다. Fourier.py의 main함수에는 pyplot을 통한 이미지 show 만 있으므로 np.real이 더 적합하다.

3. Low_pass_filter(img, th=20)

■ 코드 설명 – Low_pass_filter(img, th=20)

```
def low_pass_filter(img, th=20):
    """
    Get filtered image that pass through with low-pass filter for an input image.
    :param img: input image
    :param th:  threshold unit in piexel
    :return:  filtered image that pass through with low pass filter
    """
    # Write low pass filter here

    # distance function
    def distance(x1, y1, x2, y2):
        return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

    # get magnitude spectrum using fftshift_diy
    spectrum = fftshift_diy(img)

    # variables initialization
    n_row, n_col = img.shape
    center_row, center_col = n_row / 2, n_col / 2

    # create low pass filter
    filter = np.zeros((n_row, n_col))

    # create the low pass filter - for every pixel in the filter,
    for row in range(n_row):
        for col in range(n_col):

            # if the distance between center point and current point is smaller than th parameter,
            # set current point value to 1, which means pass
            if distance(row, col, center_row, center_col) < th:
                filter[row, col] = 1

    # mask the filter on spectrum
    spectrum = spectrum * filter

    # Reform the image from low pass filtered spectrum and return the result image
    f_ishift = ifftshift_diy(spectrum)
    img_filtered= np.fft.ifft2(f_ishift)

    return img_filtered.real
```

- 앞서 설명한 high pass filter와 달리 low pass filter는 이미지의 magnitude spectrum에서 저주파 부분만 통과시키는 함수이다
- High pass filter와 구현 방법은 똑같으나, filter의 0과 1을 반대로 주어 low pass filter를 구현했다.
- 반환시 np.abs가 아닌 마찬가지로 np.real을 사용했다.

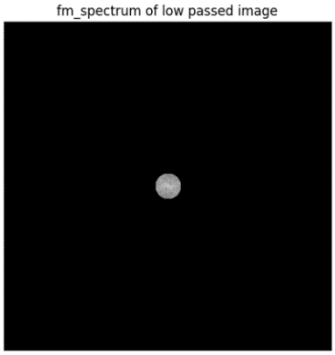
■ 코드 실행, 결과 - Low_pass_filter(img, th=20)

- **실행 코드**

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

from task2.fourier import *

img = cv2.imread('task2_sample.png',0)
low_passed = low_pass_filter(img)


plt.subplot(111),plt.imshow(img, cmap='gray')
plt.title('original Image'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(low_passed, cmap='gray')
plt.title('high passed image '), plt.xticks([]), plt.yticks([])
plt.show()
```

```
plt.subplot(111),plt.imshow(fm_spectrum(low_passed), cmap='gray')
plt.title('fm_spectrum of high passed image'), plt.xticks([]), plt.yticks([])
plt.show()
```

- **실행 결과**



앞서 high pass filter을 거친 결과 이미지는 본래 이미지의 대부분 정보가 없이 윤곽만 남아있었다. 하지만 low pass filter을 적용한 이미지는 반대로 윤곽이 흐려진 것 이외에 대부분 정보가 남아있음을 확인할 수 있었다. 이미지의 대부분의 정보가 low-frequency에 존재함을 확인할 수 있었다.

4. denoise1(img)

■ **코드설명 denoise1(img)**

```
def denoise1(img):
    """
    Denoise checker effect with given sample image. (task2_corrupted_1.png)
    Only works for given sample image
    :param img: sample image
    :return: de-noised sample image
    """
    # get the spectrum of sample image
    spectrum = fftshift_diy(img)

    # variable initialization
    n_row, n_col = img.shape

    # Denoise checker image by changing values of points to 0, which are estimated as checker noises
    for row in range(100,n_row, 110):
        for col in range(100,n_col,110):
            for i in range(15):
                for p in range(15):
                    spectrum[row-i][col-p] = 0
    for row in range(155, n_row- 100, 220):
        for col in range(155, n_col- 100, 220):
            for i in range(15):
```

```
                for p in range(15):
                    spectrum[row - i][col - p] = 0

    # reform image from de-noised spectrum and return the de-noised image
    f_ishift = ifftshift_diy(spectrum)
    img_filtered = np.fft.ifft2(f_ishift)

    return img_filtered.real
```

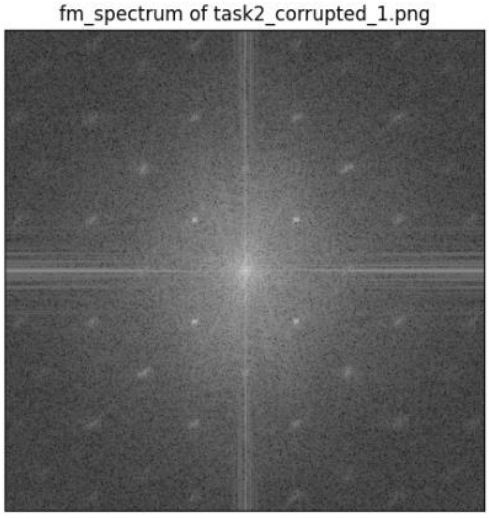
- task2_corrupted_1.png를 인풋으로 받고 denoise 작업을 한 이미지를 반환한다. 먼저, 이미지의 fm spectrum을 살펴본 결과 fm spectrum상으로 check 문양으로 noise가 되어있었다. 이를 denoise하기 위해 for문들을 사용해 해당 denoise 부분에 접근, 해당 픽셀을 0으로 바꾸어 de-noise를 구현했다. De-noise의 범위가 명확하지 않고 강하지 않은 맨 가장자리 부분은 결과의 차이가 de-noise에 도움이 되는 영향을 주지 않는 것으로 판단, 필터링 작업을 하지 않았다.

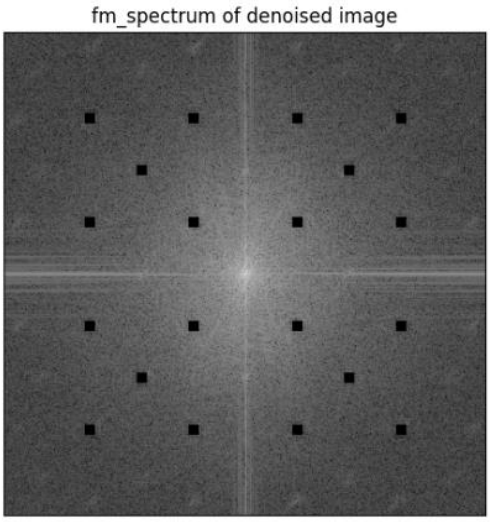
■ 코드 실행, 결과 - Low_pass_filter(img, th=20)

- **실행 코드**

```
import numpy as np
from matplotlib import pyplot as plt

from task2.fourier import *

img = cv2.imread('task2_corrupted_1.png',0)
denoised = denoise1(img)


plt.subplot(111),plt.imshow(img, cmap='gray')
plt.title('task2_corrupted_1.png '), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(fm_spectrum(img), cmap='gray')
plt.title('fm_spectrum of task2_corrupted_1.png'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(denoised, cmap='gray')
plt.title('denoised image'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(fm_spectrum(denoised), cmap='gray')
plt.title('fm_spectrum of denoised image'), plt.xticks([]), plt.yticks([])
plt.show()
```

- **실행 결과**





5. Denoise2(img)

■ **코드설명 denoise2(img)**

```
def denoise2(img):
    """
    De-noise wave effect with given sample image (task2_corrupted_2.png)
    Only works for given sample image
    :param img: sample image
    :return: de-noised sample image
    """

    # distance function
    def distance(x1, y1, x2, y2):
        return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

    # get the spectrum from sample image
    spectrum = fftshift_diy(img)

    # variables initialization
    n_row, n_col = img.shape
    center_row, center_col = n_row / 2, n_col / 2

    # create band-reject filter
    filter = np.zeros((n_row, n_col))

    for row in range(n_row):
        for col in range(n_col):
            if distance(row, col, center_row, center_col) < 40:
                filter[row, col] = 1
            elif 43 < distance(row, col, center_row, center_col):
                filter[row, col] = 1

    # mask the band-reject filter to spectrum
    spectrum = spectrum * filter

    # reform image from de-noised spectrum and return de-noised image
    f_ishift = ifftshift_diy(spectrum)
    img_filtered = np.fft.ifft2(f_ishift)

    return img_filtered.real
```

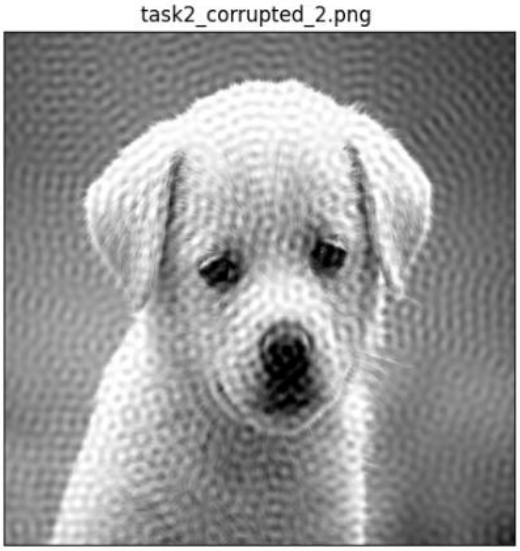
- task2_corrupted_2.png를 인풋으로 받고 denoise 작업을 한 이미지를 반환한다. 먼저, 이미지의 fm spectrum을 살펴본 결과 fm spectrum상으로 원형 모양으로 noise가 되어있었다. 이를 denoise하기 위해 for문들을 사용해 해당 denoise 부분에 접근, 해당 픽셀을 0으로 바꾸어 de-noise를 구현했다. 스펙트럼 중앙에서 반지름이 40인 원을 시작으로 반지름이 43인 원의 경계까지 0으로 바꾸었다. 즉, Band rejection filter 구현이다.

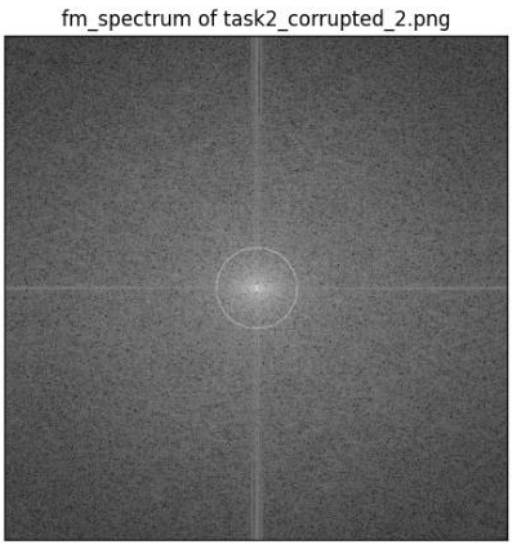
■ **코드 실행, 결과 - Low_pass_filter(img, th=20)**

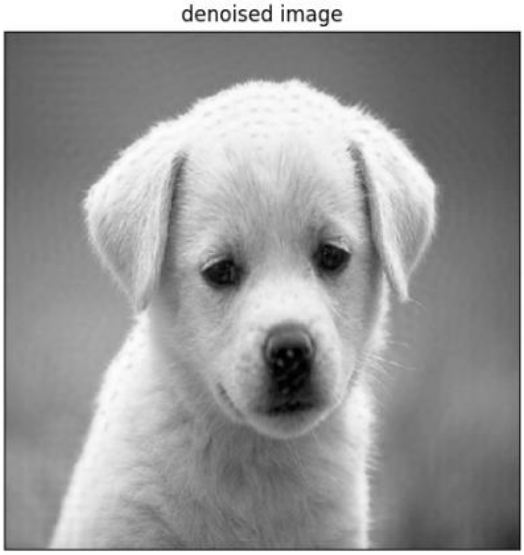
- **실행 코드**

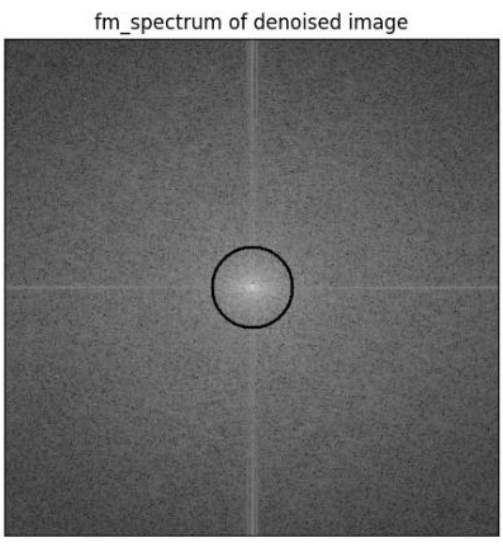
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

from task2.fourier import *

img = cv2.imread('task2_corrupted_2.png',0)
denoised = denoise2(img)


plt.subplot(111),plt.imshow(img, cmap='gray')
plt.title('task2_corrupted_2.png '), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(fm_spectrum(img), cmap='gray')
plt.title('fm_spectrum of task2_corrupted_2.png'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(denoised, cmap='gray')
plt.title('denoised image'), plt.xticks([]), plt.yticks([])
plt.show()
plt.subplot(111),plt.imshow(fm_spectrum(denoised), cmap='gray')
plt.title('fm_spectrum of denoised image'), plt.xticks([]), plt.yticks([])
plt.show()
```

- **실행 결과**









6. 추가적으로 구현한 함수들.

코드 설명

- **Fftshift_diy**

```
def fftshift_diy(img):
    """
    DIY version of numpy.fft.fftshift
    Created for this project
    This project prohibits using numpy.fft.fftshift library function
    Further explanation is written in comment lines of fm_specturm function

    :param img: input image
    :return: same as numpy.fft.fftshift(img)
    """

    # do the fourier 2d transform using numpy library
    f = np.fft.fft2(img)

    # initialization
    n_row, n_col = f.shape
    temp = np.zeros(f.shape, dtype='complex')

    # shift!
    for row in range(n_row):
        for col in range(n_col):
            if row < n_row // 2 and col < n_col // 2:
                temp[row + n_row // 2][col + n_col // 2] = f[row][col]
            elif row < n_row // 2 and col > n_col // 2:
                temp[row + n_row // 2][col - n_col // 2] = f[row][col]
            elif row > n_row // 2 and col < n_col // 2:
                temp[row - n_row // 2][col + n_col // 2] = f[row][col]
            else:
                temp[row - n_row // 2][col - n_col // 2] = f[row][col]

    return temp
```

- **Ifftshift_diy**

```
def ifftshift_diy(spectrum):
    """
       DIY version of numpy.fft.ifftshift
       Created for this project
       This project prohibits using numpy.fft.ifftshift library function
       Further explanation is written in comment lines of fm_specturm function

       :param img: input image
       :return: same as numpy.fft.ifftshift(img)
       """
    # initialization
    n_row, n_col = spectrum.shape
    temp = np.zeros(spectrum.shape, dtype='complex_')

    # shift
    for row in range(n_row):
        for col in range(n_col):
            if row < n_row // 2 and col < n_col // 2:
                temp[row + n_row // 2][col + n_col // 2] = spectrum[row][col]
            elif row < n_row // 2 and col >= n_col // 2:
                temp[row + n_row // 2][col - n_col // 2] = spectrum[row][col]
            elif row >= n_row // 2 and col < n_col // 2:
                temp[row - n_row // 2][col + n_col // 2] = spectrum[row][col]
            else:
                temp[row - n_row // 2][col - n_col // 2] = spectrum[row][col]

    return temp
```

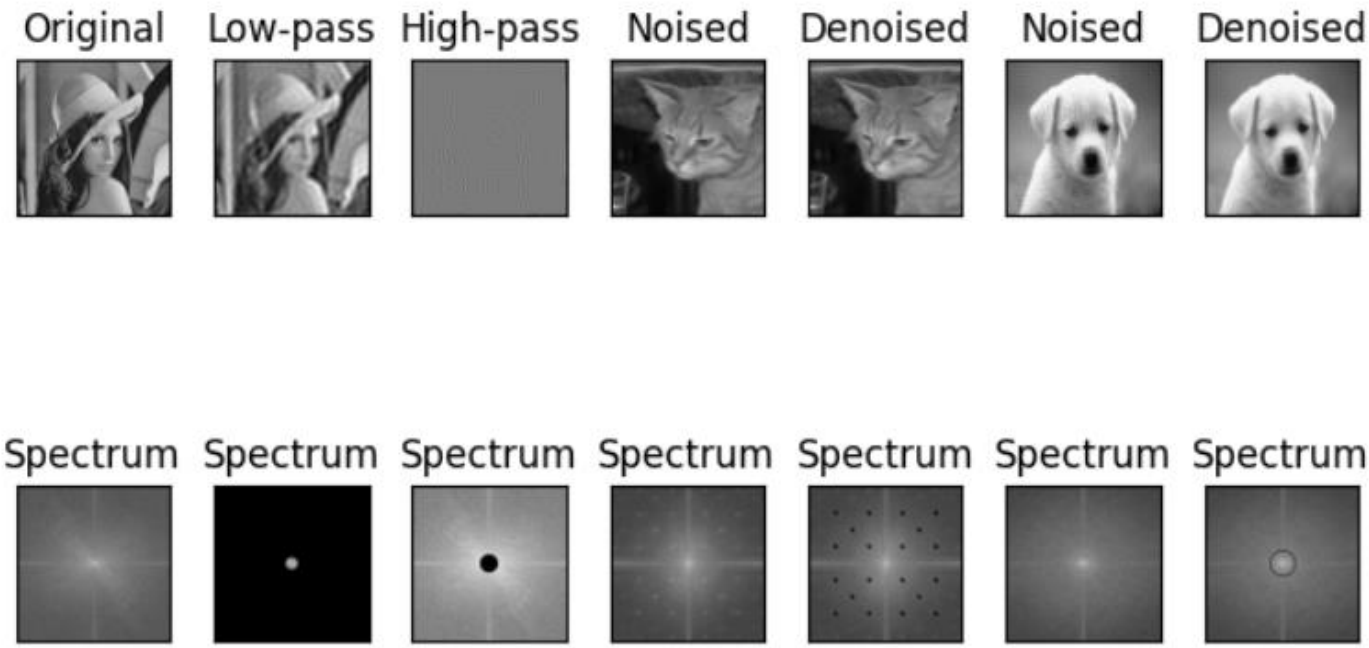
프로젝트의 제약조건은 넘파이 라이브러리의 numpy.fft.fftshift,와 numpy.fft.ifftshift을 사용하지 못하는 것이었다. 그래서 fftshift_diy,와 ifftshift_diy함수를 따로 구현해서 사용했다. 기능은 넘파이 라이브러리의 함수들과 동일하며 구현 원리에 대한 설명은 fm_spectrum 함수와 같다.

7. Fourier.py

■ **코드설명 : 프로젝트에서 제공한 메인 함수이다.**

```
if __name__ == '__main__':
    img = cv2.imread('task2_sample.png', cv2.IMREAD_GRAYSCALE)
    cor1 = cv2.imread('task2_corrupted_1.png', cv2.IMREAD_GRAYSCALE)
    cor2 = cv2.imread('task2_corrupted_2.png', cv2.IMREAD_GRAYSCALE)

    def drawFigure(loc, img, label):
        plt.subplot(*loc), plt.imshow(img, cmap='gray')
        plt.title(label), plt.xticks([]), plt.yticks([])

    drawFigure((2,7,1), img, 'Original')
    drawFigure((2,7,2), (low_pass_filter(img)), 'Low-pass')
    drawFigure((2,7,3), (high_pass_filter(img)), 'High-pass')
    drawFigure((2,7,4), cor1, 'Noised')
    drawFigure((2,7,5), denoise1(cor1), 'Denoised')
    drawFigure((2,7,6), cor2, 'Noised')
    drawFigure((2,7,7), denoise2(cor2), 'Denoised')

    drawFigure((2,7,8), fm_spectrum(img), 'Spectrum')
    drawFigure((2,7,9), fm_spectrum(low_pass_filter(img)), 'Spectrum')
    drawFigure((2,7,10), fm_spectrum(high_pass_filter(img)), 'Spectrum')
    drawFigure((2,7,11), fm_spectrum(cor1), 'Spectrum')
    drawFigure((2,7,12), fm_spectrum(denoise1(cor1)), 'Spectrum')
    drawFigure((2,7,13), fm_spectrum(cor2), 'Spectrum')
    drawFigure((2,7,14), fm_spectrum(denoise2(cor2)), 'Spectrum')

    plt.show()
```

■ **코드 실행 결과**



기본으로 제공한 main function의 실행 결과이다. 많은 수의 이미지를 한 figure에 띄우다 보니 이미지의 품질이 좋지 않다. 각각의 자세한 이미지는 위의 함수별 기능 구현 설명 부분에서 보다 큰 이미지로 첨부했다.

<개발 환경 설정과 문제점 및 해결 과정>

■ 프로젝트의 개발 환경

- OS: windows 10
- Language: python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit (AMD64)]

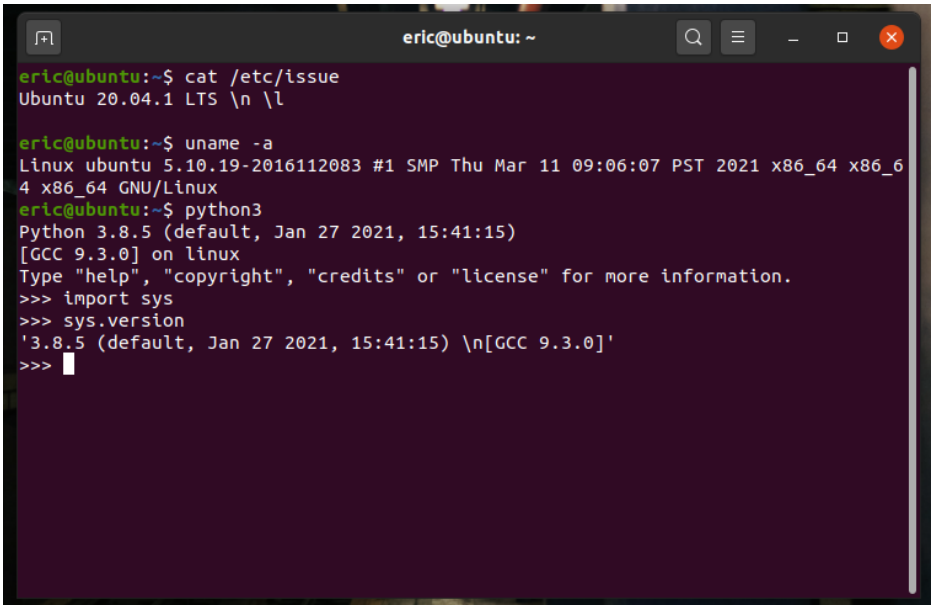
■ Grading environment

- Language: Python3 (>= 3.5.2)
- OS: Linux (Ubuntu 16.04)

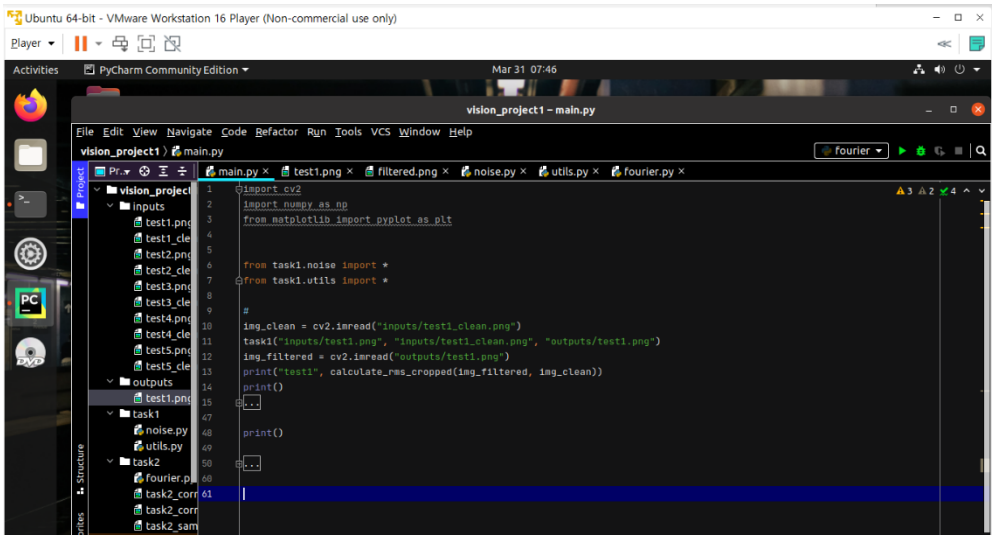
프로젝트의 개발 환경과 grading environment가 달라 프로젝트 완료 후 grading environment에 적합한 환경을 구축해 문제없이 실행이 되는지 확인하는 과정을 거쳤다. 다음은 테스트 과정 중 겪게 된 문제점들과 해결 방안이다

VM ware Workstation Player 16을 사용하여 가상환경을 통해 우분투 운영체제를 만들고 python과 pycharm ide를 설치해서 테스트를 진행했다.

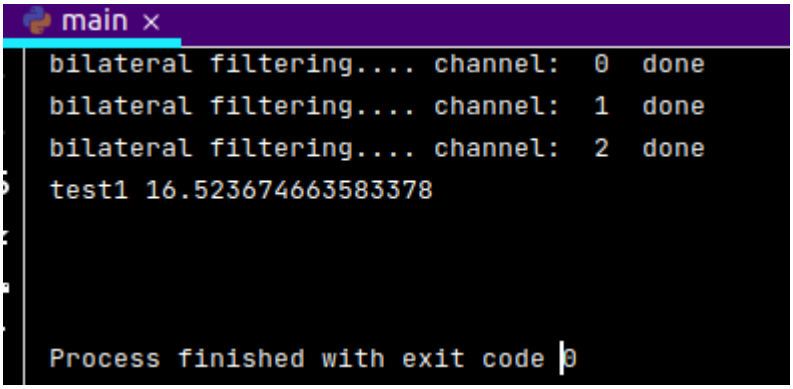
우분투 운영체제 정보 및 파이썬 버전



테스트코드 - Task1.



테스트 결과 – Task1

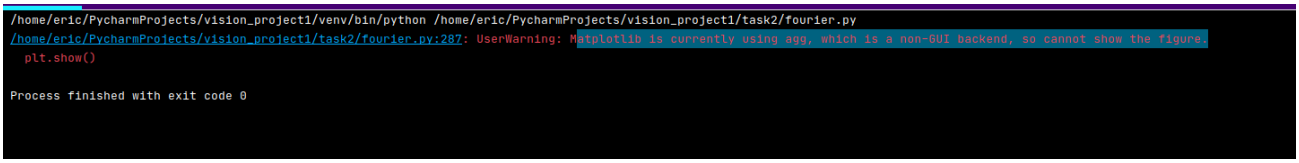


테스트 결과, task1은 에러없이 잘 실행되었고 윈도우 운영체제에서 결과도 동일했다.

테스트 코드 – task2

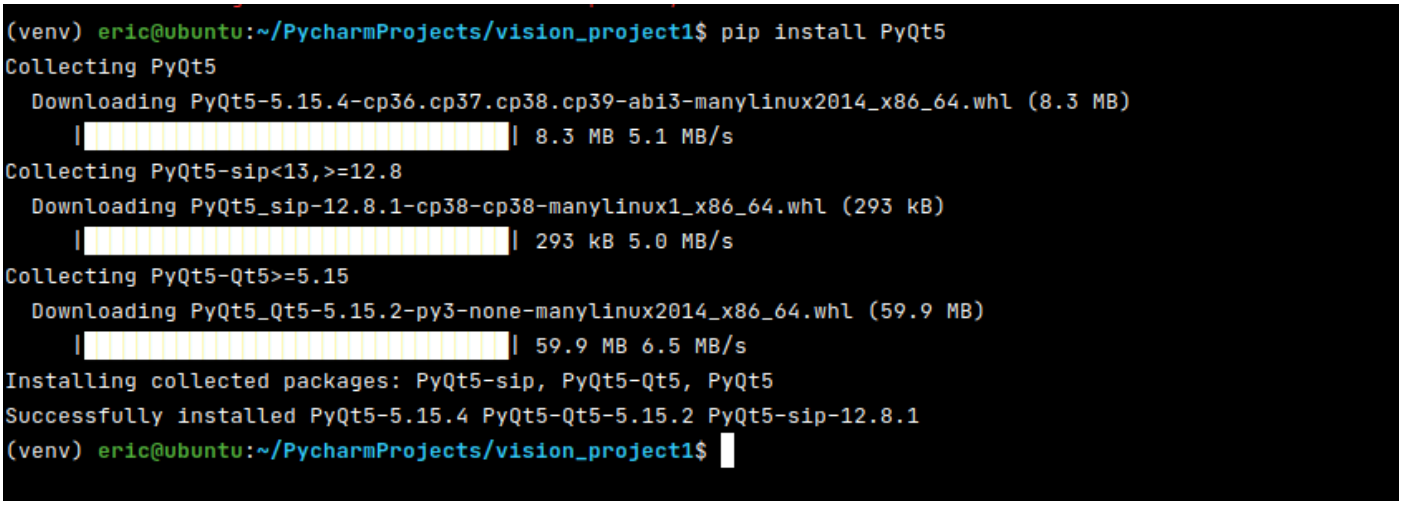
-fourier.py의 기본 제공 main function을 사용.

테스트 결과 - task2

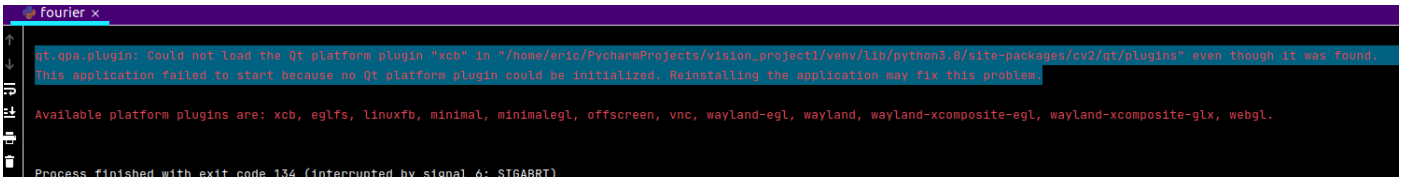


하지만 태스크 2는 위와 같은 에러가 발생했다.
에러 코드를 검색했고 검색 결과, PyQt5를 설치하면 된다는 해결책을 시도했다.

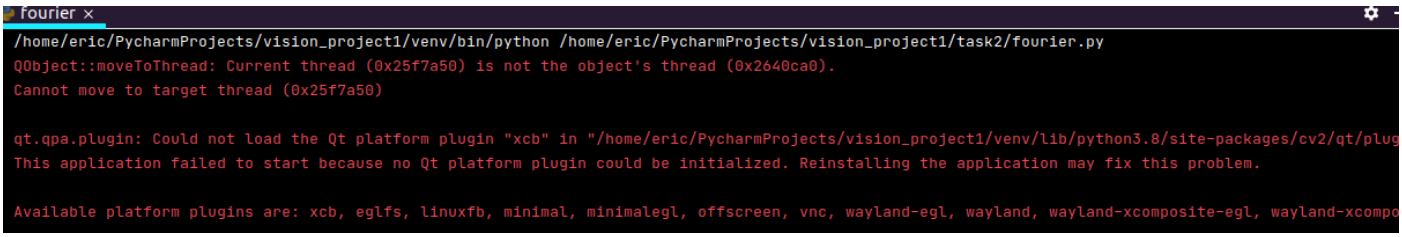
(https://blog.naver.com/s97083/222055219322)



그럼에도 불구하고 다음과 같은 에러가 발생했다.



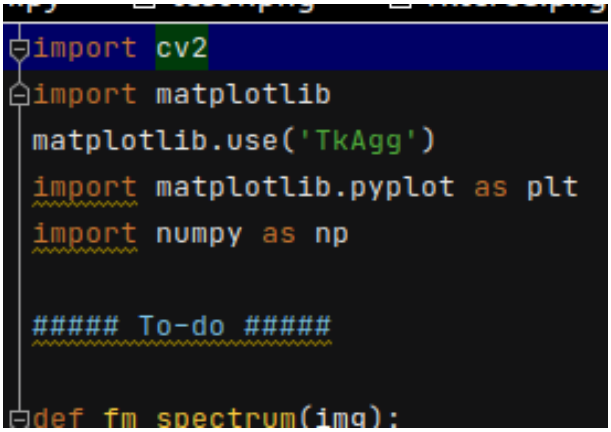
에러 메시지에서 말하는 libxb-xinerama0를 설치했음에도 다음과 같은 에러메세지가 계속 발생했다.

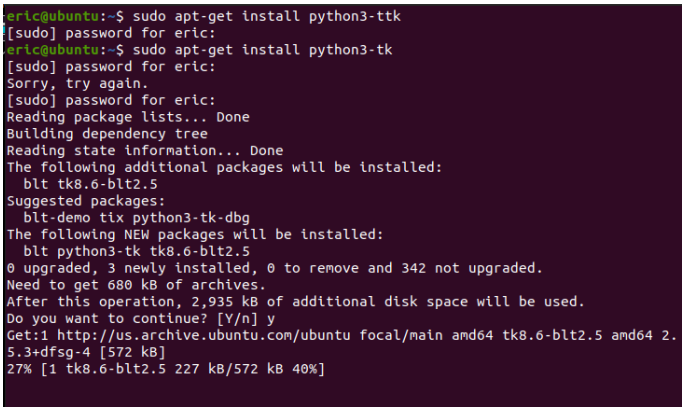


이에 추가적인 구글링을 진행했고, 다음과 같은 해결책을 발견하여 적용했다.

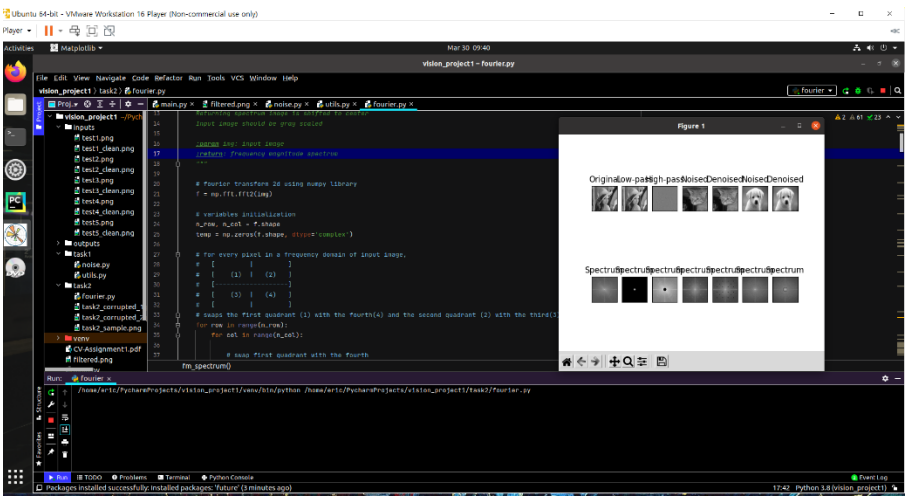
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
(https://stackoverflow.com/questions/56656777/userwarning-matplotlib-is-currently-using-agg-which-is-a-non-gui-backend-so)

foruier.py에 import관련된 위의 코드를 추가하고, python3-tk 모듈을 설치해주는 해결방안이다.





- 테스트 - Task2 실행 결과



<참고 문헌>

P6, Bilateral filter 수식 캡처

http://people.csail.mit.edu/sparis/bf_course/

http://people.csail.mit.edu/sparis/bf_course/slides08/03_definition_bf.pdf

P24~, 개발환경 설정 문제점 해결

https://blog.naver.com/s97083/222055219322

https://cypsw.tistory.com/18
https://stackoverflow.com/questions/56656777/userwarning-matplotlib-is-currently-using-agg-which-is-a-non-gui-backend-so