2022-1 컴퓨터 비전 project1 보고서

2018147558_김정주

Task1.

Noise.py

-Add_gaussian_noise

```
def add_gaussian_noise(image):
    # Use mean of 0, and standard deviation of image itself to generate gaussian noise
    # Use the shape function to obtain the height and width values of the image.

    Height = image.shape[0]
    Width = image.shape[1]

    # Variable that amplifies the noise value
    intensity = 28.3
    # for loop to apply noise per pixel

    for h in range(Height):
        for w in range(Width):
            # function of Numpy that randomly returns normal distribution information values
            random_normal_value = np.random.normal()
            noise = intensity * random_normal_value
            # Add the generated noise value to the original image and store it.
            image[h][w] = image[h][w] + noise

    return image
```

먼저 shape를 사용하여 해당 이미지의 높이와 넓이 픽셀을 가져온다. Intensity는 노이즈 값을 조절하는 변수이다. Numpy 내장함수인 random.normal을 사용하여 정규분포에 대한 랜덤값을 받아고 랜덤값에 intensity값을 곱하여 각 이미지 픽셀에 추가하여 구현

-Add_uniform_noise

```
def add_uniform_noise(image):
    # Generate noise of uniform distribution in range [0, standard deviation of image)
    # Use the shape function to obtain the height and width values of the image.

    Height = image.shape[0]
    Width = image.shape[1]

    # Variable that amplifies the noise value
    intensity = 29
    # for loop to apply noise per pixel

    for h in range(Height):
        for w in range(Width):
            # function of Numpy to obtain random equal distribution values
            random_uniform_value = np.random.uniform()
            noise = intensity * random_uniform_value
            # Add the generated noise value to the original image and store it.
            image[h][w] = image[h][w] + noise
    return image
```

구현 방식은 가우시안 노이즈를 했을 때와 비슷하다 달라는 것은 random.uniform함수를 통해 균

등분포의 값을 랜덤으로 가져와 각 픽셀이 더하는 방식이다.

-apply_impulse_noise

```python
def apply_impulse_noise(image):
    # Implement pepper noise so that 20% of the image is noisy
    # Use the shape function to obtain the height and width values of the image.

    Height = image.shape[0]
    Width = image.shape[1]

    # adjustment of noise strength and weakness(0~1)
    Criteria = 0.0776
    # for loop to apply noise per pixel

    for h in range(Height):
        for w in range(Width):
            # function for obtaining random values from 0 to 1
            random_value = random.random()
            if random_value < Criteria :
                # The smallest value in color
                image[h][w] = 0
            elif random_value > (1-Criteria) :
                # The color value is 8 bits.
                image[h][w] = 255
    return image
```

위에서의 두 노이즈 방식과 비슷하다. 하지만 impulse 노이즈의 경우, 소금 후추와 같이 하얀 픽셀과 검은 픽셀이 랜덤으로 존재하기 이를 처리하기 위해 기준값을 정의하여 if문 내부에 배치함으로 해당 기준값에 따라 0과 255 값을 대입하는 방식으로 구현했다.

Denoise.py

-apply_median_filter

```python
def apply_median_filter(img, kernel_size):

    """
    You should implement median filter using convolution in this function.
    It takes 2 arguments,
    'img' is source image, and you should perform convolution with median filter.
    'kernel_size' is an int value, which determines kernel size of median filter.

    You should return result image.
    """
    #Receive information on the image.
    Height = img.shape[0]
    Width = img.shape[1]
    ch = img.shape[2]

    #Middle index of kernel
    index = kernel_size // 2

    #Copy body to retain the original image value
    img_copy = img.copy()
    for c in range(ch):
        for h in range(Height):
            for w in range(Width):
                    mid_mat = []
                    for i in range(kernel_size):
                        for j in range(kernel_size):
                            x = h-index+i
                            y = w-index+j
                            #Processing Boundary Values
                            if x < 0 or y < 0 : pass
                            elif x >= Height or y >= Width : pass
                            else :
                                mid_mat.append(img_copy[x][y][c])
                    #Process for obtaining intermediate values
                    mid_mat.sort();
                    idx = len(mid_mat) // 2
                    img[h][w][c] = mid_mat[idx]

    return img
```

이 필터의 경우, 마스크의 중간값을 대상 픽셀에 적용하는 필터로서, 먼저 픽셀의 높이와 넓이, 채널 값을 받는다. 이 필터의 경우 적용되는 대상이 컬러이기에 3개의 채널이 존재함으로 각 채널에 대해서 각각 미디안 필터를 적용하여 구현하였다. Index의 경우 커널 사이즈의 중간값으로 경계를 나누는 지점이 되기에 이런식으로 설정하였다. 각 이미지의 가장자리인 경계값에 대한 처리는 경계에 있는 값은 데이터로 인식하지 않고 나머지 마스크에서의 중간값을 찾는 것으로 처리하였다.

-apply_bilateral_filter

```
def gauss(x, sigma):
    return 1/(sigma**2*2*np.pi) * np.exp(-(x*x)/(sigma**2*2))
```

2개의 가우스 연산을 해줄 gauss 함수, 가우스 함수의 연산을 수행한다.

```
def apply_bilateral_filter(img, kernel_size, sigma_s, sigma_r):

    """
    You should implement bilateral filter using convolution in this function.
    It takes at least 4 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(gaussian function for space)
    'sigma_r' is a int value, which is a sigma value for G_r(gaussian function for range)

    You should return result image.
    """

    #Receive information on the image.
    Height = img.shape[0]
    Width = img.shape[1]
    ch = img.shape[2]

    #Middle index of kernel
    index = kernel_size // 2

    #Copy body to retain the original image value
    img_copy = img.copy().tolist()

    for h in range(Height):
        for w in range(Width):
            for c in range(ch):
                #The value of the pixel with the filter applied and the variable to normalize it
                ft_pixel = 0
                normal = 0
                for i in range(kernel_size):
                    for j in range(kernel_size):
                        #Specify the scope to which the filter applies
                        hindex = h - index + i
                        windex = w - index + j
                        #Processing Boundary Values
                        if hindex >= Height : hindex -= Height
                        if windex >= Width : windex -= Width
                        #Gaussian operations for range values
                        rge = img_copy[hindex][windex][c] - img_copy[h][w][c]
                        rge = gauss(rge, sigma_r)
                        #Gaussian operations for distance values
                        dit = np.sqrt(((h-hindex)**2) + ((w-windex)**2))
                        dit = gauss(dit, sigma_s)
                        #Convolution two operations and obtaining pixe values
                        bilater = dit * rge
                        ft_pixel = ft_pixel + (img_copy[hindex][windex][c] * bilater)
                        normal = normal + bilater
                #Apply obtained pixel value
                ft_pixel = ft_pixel // normal
                ft_pixel = int(ft_pixel)
                img[h][w][c] = ft_pixel

    return img
```

위의 중간값 필터와 동일하게 이미지에 대한 정보를 가져오고, 커널 크기에 따라 연산을 수행해 준다. 여기서 ft_pixel은 대상 픽셀의 값을 저장해주는 변수이고, normal을 해당 픽셀의 값을 정규화 해주는 변수이다. 경계값에 대한 처리는 경계의 전 픽셀 즉 4면을 같은 이미지로 두었을 때 해당하는 값으로 치환하는 것으로 하였다. rge값은 대상 픽셀과 주변 픽셀과의 값의 차이를 기준으로 가우스 함수 값을 나타내고, dit값은 대상 픽셀과 주변 픽셀과의 거리에 대한 가우스 함수 값을 나타낸다. 그리고 모든 연산이 끝나면 해당 픽셀 값에 정규화를 시키고 이미지에 적용시켜 구현하였다.

-apply_my_filter

```python
#Functions that return Gaussian kernels that fit the kernel size
def gaussian(kernel_size,sigma_s):
    kernel = np.zeros((kernel_size,kernel_size))
    mid = kernel_size // 2
    for i in range(kernel_size):
        for j in range(kernel_size):
            temp = ((i-mid)**2) + ((i-mid)**2)
            kernel[i][j] = np.exp(-temp/(sigma_s**2*2))/(sigma_s**2*2)
    kernel = kernel/np.sum(kernel)
    return kernel
```

커널 사이즈에 맞춰 가우시안 필터를 리턴하는 함수이다. 내 필터를 적용함에 있어서 필요한 함수이게 선언하였다.

```python
def apply_my_filter(img, kernel_size, sigma_s):
    """
    You should implement additional filter using convolution.
    You can use any filters for this function, except median, bilateral filter.
    You can add more arguments for this function if you need.

    You should return result image.
    """
    #Receive information on the image.
    Height = img.shape[0]
    Width = img.shape[1]
    ch = img.shape[2]

    #Middle index of kernel
    mid = kernel_size // 2

    #Copy body to retain the original image value
    img_copy = img.copy()

    #Creating a kernel to apply Gaussian filters
    gauss = gaussian(kernel_size, sigma_s)

    for h in range(Height):
        for w in range(Width):
            for c in range(ch):
                #The value of the pixel with the filter applie
                pixel = 0
                for i in range(kernel_size):
                    for j in range(kernel_size):
                        #Specify the scope to which the filter applies
                        x = h - mid + i
                        y = w - mid + j
                        #Processing Boundary Values
                        if x >= Height : x -= Height
                        if y >= Width : y -= Width
                        #intens value for applying sharpness filter
                        intens = 0
                        #intens value for median value of filter
                        if x == h and y == w : intens = 2
                        #Applying and Normalizing Intens Values
                        temp = img_copy[x][y][c] * intens - (1/(kernel_size**2))
                        #Applying and Normalizing Intens Values
                        temp = temp * gauss[i][j]
                        #Adjusted Value
                        temp *= 4.3
                        pixel += temp
                img[h][w][c] = pixel

    return img
```

내 필터의 기본적인 방식은 Sharpness filter를 먼저 적용하여 edge에 대한 값을 강조하고 이후에 가우시안 필터를 적용하는 방식으로 접근했다. 먼저 위에 필터들과 동일하게 정보를 받고 복사본을 생성하는 것부터 시작하였다. 이후 가우시안 필터를 커널 크기에 맞게 생성 하였다. 이후 커널이 적용될 범위를 지정하였고, 경계값에 대한 처리는 bilateral filter에서 한것처럼 처리하였다. 그리고 먼저 sharpness 필터를 적용하였고, 이후에 가우스 함수를 적용하였다. 여기서 한가지 문제가 발생하는데, sharpness 필터를 먼저적용하고 가우시안 필터를 적용함으로써 전체적인 픽셀의 값이 낮아지면서 어두워 지는 현상이 생겼다. 이에 구해진 픽셀 값에 적절한 값을 곱함으로써 해결할 수 있었다. 찾아낸 가장 적합한 값은 4.3이였다.

-task1_2

모든 필터를 취합하여, 가장 최적의 결과를 내주는 함수이다. 해당함수는 이미지에 대한 모든 필터의 적용 후 RMS 값을 분석하여, 가장 낮은 rms값을 가지는 이미지를 선택하여 반환한다.

-최적의 결과와 분석

cat_noisy : median_filter

fox_noisy : bilateral filter

snowman_noisy : bilateral filter

고양이 사진의 경우는 전형적인 소금, 후추 노이즈를 보인다. 소금, 후추는 각 픽셀의 값이 0 또는 255로 튀는 것을 뜻하기에 중간값 필터와 같은 메커니즘은 주변 값의 중간값을 찾아내어 적용함으로써 가장 좋은 솔루션이 되지 않았나 싶다. 최적의 kernel_size는 3이었으며 이 이상 높이면 blur가 되는 범위가 넓어져 좋은 결과는 도출해내지 못하는 듯하다.

```python
def task1_2(src_path, clean_path, dst_path):
    """
    This is main function for task 1.
    It takes 3 arguments,
    'src_path' is path for source image.
    'clean_path' is path for clean image.
    'dst_path' is path for output image, where your result image should be saved.

    You should load image in 'src_path', and then perform task 1-2,
    and then save your result image to 'dst_path'.
    """
    noisy_img = cv2.imread(src_path)
    clean_img = cv2.imread(clean_path)

    #Apply each filter to the noisy image
    temp1 = apply_bilateral_filter(noisy_img.copy(), 7, 12, 60)
    temp2 = apply_median_filter(noisy_img.copy(), 3)
    temp3 = apply_my_filter(noisy_img.copy(),3, 50)

    #rms for filtered images
    v1 = calculate_rms(clean_img,temp1)
    v2 = calculate_rms(clean_img,temp2)
    v3 = calculate_rms(clean_img,temp3)

    #Select the image with the smallest rms value as the result
    result_img = temp1
    if v2 < v1 : result_img = temp2
    if v3 < v2 : result_img = temp3

    print(calculate_rms(clean_img,result_img))

    # do noise removal

    cv2.imwrite(dst_path, result_img)

    return 0
```

여우 사진과 눈사람 사진의 경우는 정확하기는 판단이 되지는 않으나, uniform noise 또는 gaussian noise로 추측된다. 이 같은 경우 이미지의 edge를 살리면서 blur를 해주는 bilateral filter의 특성 상 노이즈로 추정되는 픽셀을 주변픽셀과의 차이를 모호하게 해줌으로써 이와 같은 결과가 나오지 않았나 분석한다. 그리고 최적의 kernel_size, r_sigma, s_sigma 값은 이미지에 따라 변경되지만 두 이미지가 최적의 RMS값을 가지는 경우는 7, 12, 60으로 도출되었다.

Task2.

Fouier.py

-fftshift

```
def fftshift(img):
    '''
    This function should shift the spectrum image to the center.
    You should not use any kind of built in shift function. Please implement your own.
    '''
    #Receive information on the image.
    height, width = img.shape
    centerh = height // 2
    centerw = width // 2

    #Copy body to retain the original image value
    img_copy = img.copy()

    for i in range(height):
        for j in range(width):
            #Normalized value index to the center of the image.
            x = i + centerh
            y = j + centerw
            if x >= height : x = x - height
            if y >= width : y= y - width
            #Apply to Image
            img[i][j] = img_copy[x][y]

    return img
```

먼저 taxk2에서는 작업하는 이미지가 흑백이기에 채널값을 제외한 해당 이미지에 대한 정보를 받고, 각 이미지의 중간 index를 centerh, certerw에 각각 저장하고, 이 index를 중심으로 해당 이미지에 대한 pixel값을 재정렬하는 방식으로 정규화 하였다. 파이썬의 경우 index 값으로 마이너스를 허용함으로 이를 이용하여 구현하였다.

-ifftshift

```python
def ifftshift(img):
    '''
    This function should do the reverse of what fftshift function does.
    You should not use any kind of built in shift function. Please implement your own.
    '''

    #Receive information on the image.
    height, width = img.shape
    centerh = height // 2
    centerw = width // 2

    #Copy body to retain the original image value
    img_copy = img.copy()


    for i in range(height):
        for j in range(width):
            #Normalized value index to the center of the image.
            x = i - centerh
            y = j - centerw
            if x >= height : x = x - height
            if y >= width : y= y - width
            #Apply to Image
            img[x][y] = img_copy[i][j]

    return img
```

Fftshift와 방식은 똑같나, 반대로 중앙 index 값을 빼줌으로 구현가능하다.

-fm_spectrum

```python
def fm_spectrum(img):
    '''
    This function should get the frequency magnitude spectrum of the input image.
    Make sure that the spectrum image is shifted to the center using the implemented fftshift function.
    You may have to multiply the resultant spectrum by a certain magnitude in order to display it correc
    '''
    #Fourier transform for images
    fqimg = np.fft.fft2(img)
    #Switching to Image Coordinates
    fqimg = fftshift(fqimg)
    #Processing to be a valid value
    mgtspt = 20 * np.log(np.abs(fqimg))

    return mgtspt
```

해당 함수는 frequency magnitude 스펙트럼을 보여주는 함수로서 먼저 이미지를 받아서 numpy 내장함수를 사용하여 푸리에 변환을 시킨 후 좌표를 전환해준다. 이후에 푸리에 변환의 값이 허수 부분도 포함되어 있기에 절대값으로 구해주고, 그리고 푸리에 변환을 통해 나온 값은 굉장히 크기 때문에 로그를 씌어서 낮게 잡아준다. 이후 너무 낮아진 값에 20을 곱하여 이미지에서 해당 값이 유의미한 차이를 보이도록 수정해주어 구현하였다.

-low_pass_filter

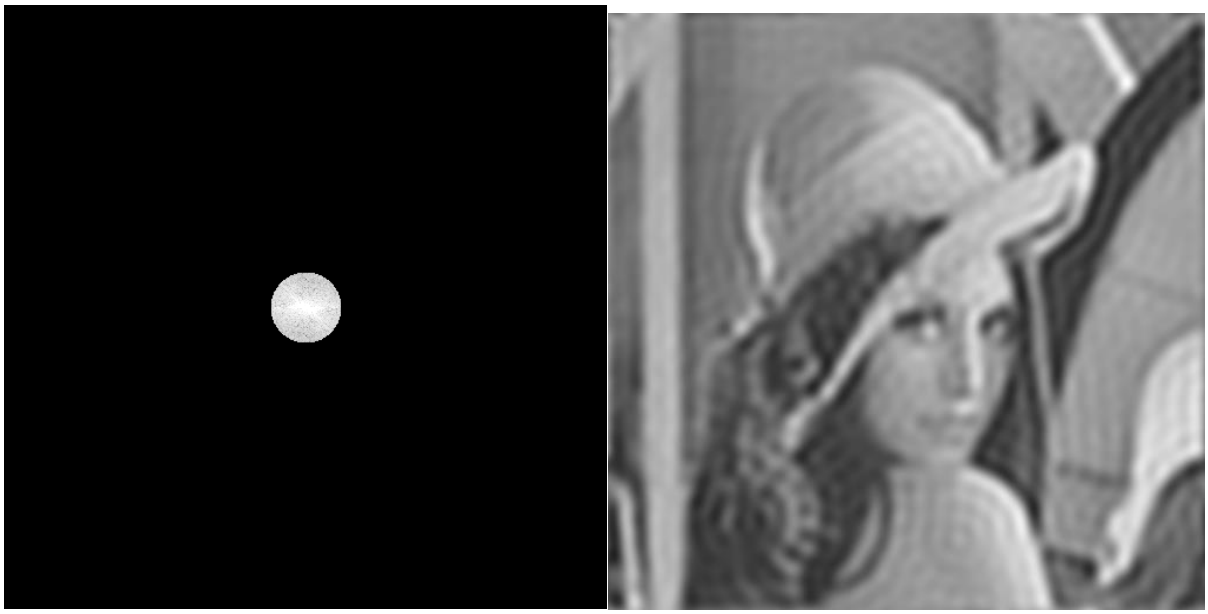
```python
def low_pass_filter(img, r=30):
    '''
    This function should return an image that goes through low-pass filter.
    '''
    #Receive information on the image.
    height, width = img.shape
    centerh = height //2
    centerw = width // 2

    #Fourier transform for images
    img_spt = np.fft.fft2(img)

    for i in range(height):
        for j in range(width):
            #Normalized value index to the center of the image.
            x = i - centerh
            y = j - centerw
            if x >= height : x = x - height
            if y >= width : y = y - width
            #the equation of a circle
            if (x**2) + (y**2) >= (r**2) : img_spt[x][y] = 0

    #Reverse Fourier transform
    img_result = np.fft.ifft2(img_spt)
    img_result = np.abs(img_result)

    return img_result
```



먼저 매개변수로 받은 이미지를 푸리에 변환 해주고 푸리에 변환을 해준 이미지의 중심을 기준으로 r만큰의 크기만큼의 내부를 제외하고는 전부 0으로 수정해주어 구현하였다. 기본적으로 원의 방정식을 사용하기에 좌표가 중심으로 전환되기에 따로 fftshift, ifftshift를 사용하지 않았으며 마지막으로 적용된 이미지의 역푸리에 변환을 해주고, 유의미한 값을 받기 위해 np.abs를 사용하여 필요없는 값을 제거했다.

-high_pass_filter

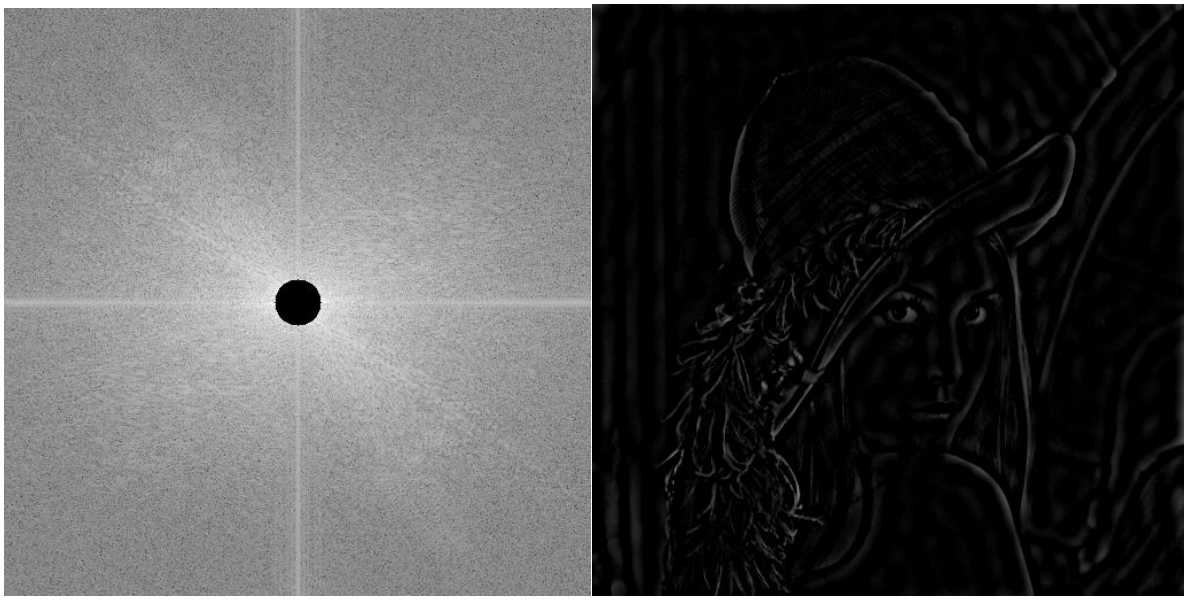
```python
def high_pass_filter(img, r=20):
    '''
    This function should return an image that goes through high-pass filter.
    '''
    #Receive information on the image.
    height, width = img.shape
    centerh = height //2
    centerw = width // 2

    #Fourier transform for images
    img_spt = np.fft.fft2(img)

    for i in range(height):
        for j in range(width):
            #Normalized value index to the center of the image.
            x = i - centerh
            y = j - centerw
            if x >= height : x = x - height
            if y >= width : y = y - width
            #the equation of a circle
            if (x**2) + (y**2) <= (r**2) : img_spt[x][y] = 0

    #Reverse Fourier transform
    img_result = np.fft.ifft2(img_spt)
    img_result = img_result.real

    return img_result
```



기본적인 구현방식은 low pass filter와 동일하나, 저역 통과 필터와는 다르게 중심좌표를 기준으로 r만큼 떨어진 곳의 내부를 0으로 변경함으로써 구현하였다. 그리고 역 푸리에 변환 후 스펙트럼 이미지를 받아낼 때 필요없는 값에 대해서 민감하게 반응함으로 .real을 사용하여 허수 값을 제거하였다.

-denoise1

```python
def denoise1(img):
    '''
    Use adequate technique(s) to denoise the image.
    Hint: Use fourier transform
    '''
    #Receive information on the image.
    height, width = img.shape
    centerh = height //2
    centerw = width // 2

    #Fourier transform for images
    img_spt = np.fft.fft2(img)

    for i in range(height):
        for j in range(width):
            #Normalized value index to the center of the image.
            x = i - centerh
            y = j - centerw
            if x >= height : x = x - height
            if y >= width : y= y - width

            #Fill the empty space through Fourier transform.
            if (51 < x and x < 57) and (51 < y and y < 57):
                img_spt[x][y] = 0
                img_spt[x][-y] = 0
                img_spt[-x][y] = 0
                img_spt[-x][-y] = 0
            if (79 < x and x < 85) and (79 < y and y < 85):
                img_spt[x][y] = 0
                img_spt[x][-y] = 0
                img_spt[-x][y] = 0
                img_spt[-x][-y] = 0

    #Reverse Fourier transform
    img_result = np.fft.ifft2(img_spt)
    img_result = np.abs(img_result)

    return img_result
```

강아지 사진의 경우 fm_spectrum으로 사진을 본 결과 x = y, x = -y에 해당하는 곳에 8개의 흰색

정사각형이 보였다. 그렇기에 해당 부분을 다시 채워주는 것으로 해결할 수 있었다. 이 노이즈를 제거하는 경우도 이미지의 정중앙을 기준으로 식을 세웠기에 fftshift, ifftshift를 사용하지 않고도 구현할 수 있었다. X를 기준으로 51과 57사이, 79와 85사이를 0으로 변경함으로서 구현하였다.

-denoise2

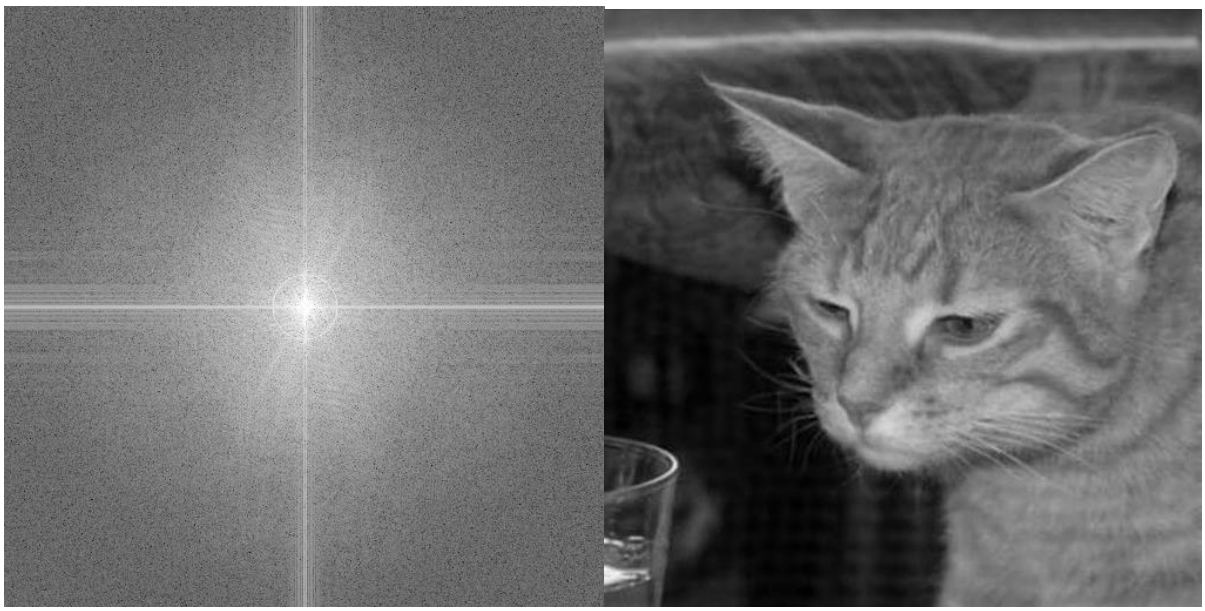
```python
def denoise2(img):
    '''
    Use adequate technique(s) to denoise the image.
    Hint: Use fourier transform
    '''
    #Receive information on the image.
    height, width = img.shape
    centerh = height //2
    centerw = width // 2

    #Fourier transform for images
    img_spt = np.fft.fft2(img)

    for i in range(height):
        for j in range(width):
            #Normalized value index to the center of the image.
            x = i - centerh
            y = j - centerw
            if x >= height : x = x - height
            if y >= width : y= y - width
            #Fill the empty space through Fourier transform.
            if (x**2) + (y**2) < 800 and (x**2) + (y**2) > 720: img_spt[x][y] = 0

    #Reverse Fourier transform
    img_result = np.fft.ifft2(img_spt)
    img_result = np.abs(img_result)

    return img_result
```



고양이 이미지의 frequency magnitude spectrum 이미지를 보면 중앙에서 좀 떨어진 부분에 원으

로 흰색부분이 나타나 있는 것으로 보인다. 이 부분을 적절히 찾아내어 denoise1에서와 같이 0으로 바꿔준다면 해결할 수 있었다. 원의 방정식을 사용하여 r^2이 800과 720일 때 가장 denoise가 잘 된다고 판단하여 해당 이미지에 적용하여 구현하였다.