Digital Image Processing
Homework #2

Mustafa Al-Barahmeh……0175630

Osama Salman……0171927

Zeyad Al-Najjar ……0175642

Problem 1

To solve this problem, we used predefined functions from the OpenCV library. We used *cv2.erode* to get the minimum filter mask, cv2.dilate for the maximum filter mask and cv2.medianBlur for the median filter mask. So, our code looks like this:

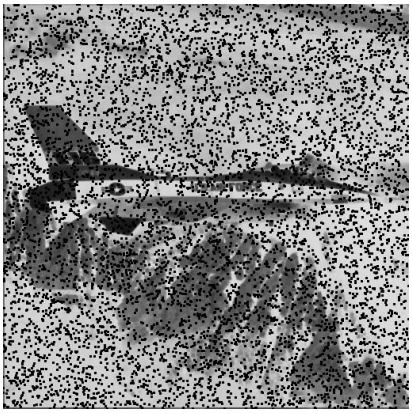
```
kernel = np.ones((3, 3), 'uint8')
f16_min = cv2.erode(f16_noisy.astype('uint8'), kernel)
f16_max = cv2.dilate(f16_noisy.astype('uint8'), kernel)
f16_median = cv2.medianBlur(f16_noisy.astype('uint8'), 3)
```

First, let us look at the noisy image we are working with, and the original image we are trying to obtain:

Noisy image

Original image

The functions used gave the following result:



Minimum-filtered image

Maximum-filtered image

Median-filtered image

The minimum filter function produced an even noisier image because it amplified the noisy pixels due to their low values. The PSNR for this image when comparing it with the original clear image was (8.943), the visual corruption this filter did explains the very low value of the PSNR.

The maximum filter function produced a fairly acceptable image as it succeeded in eliminating the low-valued noisy pixels. Although, it pushed the image to become brighter than it is. The PSNR for this image was (22.621). This is an acceptable value with a good resultant image.

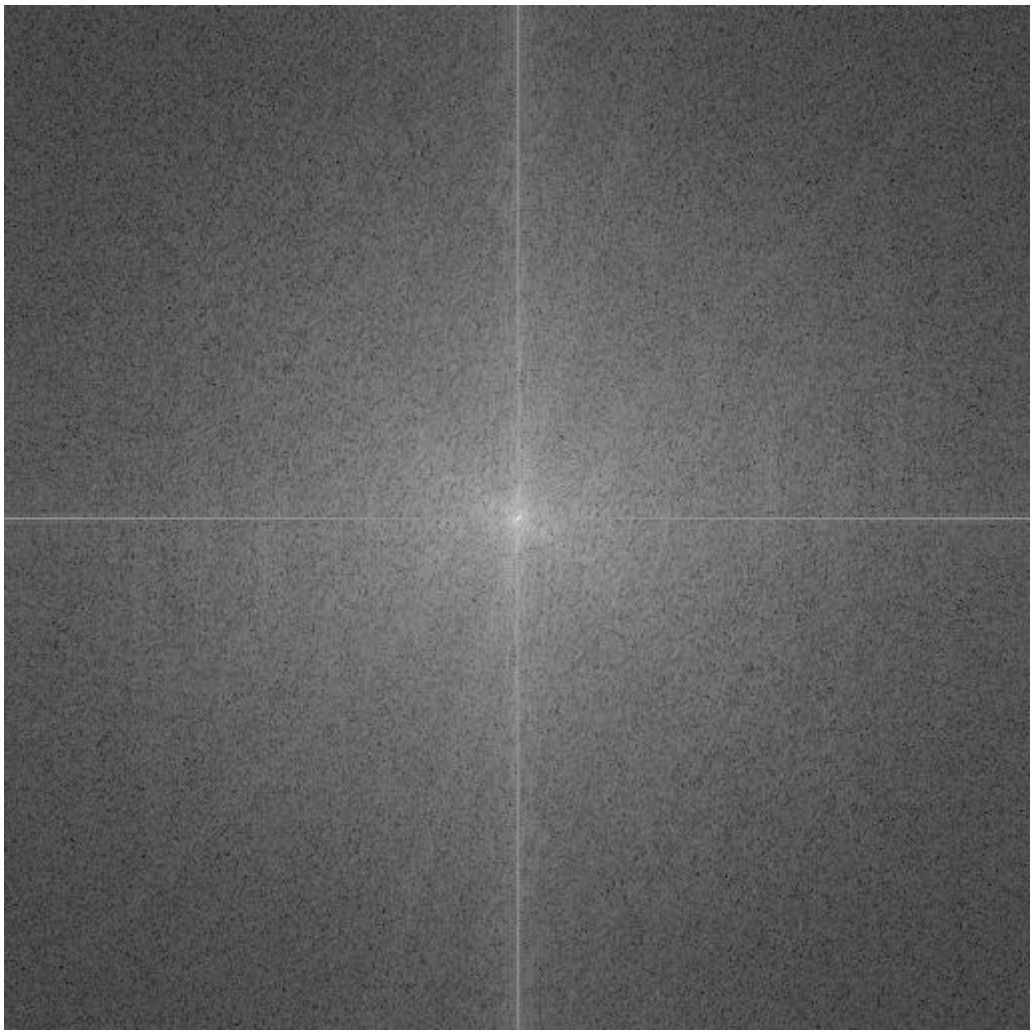
The median filter function produced the best result yet. It eliminated the noisy pixels and managed to maintain the image's brightness. The PSNR for this image was (34.272) which is considered a high value and it suggests a high similarity between the median-blurred image and the original clear image which is visually apparent.

Problem 2

We read the *f16.gif* image and computed its *FFT* using the built-in functions in the numpy library. Then, we extracted the magnitudes and the phases in sperate objects to apply the necessary operations. This is how we did it:

```
ft = fft.fft2(original_image)
magnitude, phase = get_dft_components(ft)
log_scaled_magnitude = log_scale_image(magnitude)
```

This is how the shifted, log-scaled magnitude looks like:



Log-scaled magnitude spectrum

To calculate the total power in the image, we need the sum of the squared magnitudes of the image FFT. The DC component power can be calculated by squaring the magnitude of *F(0, 0)*. This is how we did it:

```
total_power = np.sum(image_magnitude ** 2)
dc_power = np.real(image_ft[0, 0]) ** 2
percentage = ((dc_power / total_power) * 100)
```

The total power was calculated to be 2346758362628097 and the DC component power 2206878835450041. The ratio of the DC component power to the total power was 94.04% which confirms that the DC component is the largest component in the spectrum.

We transformed the image back to the spatial domain excluding the magnitude property (setting it to 1) and then we scaled it linearly using this code:

```
phase_inverse = np.real(fft.ifft2(np.exp(1j * image_phase)))
scaled = linear_scale_image(phase_inverse)
```

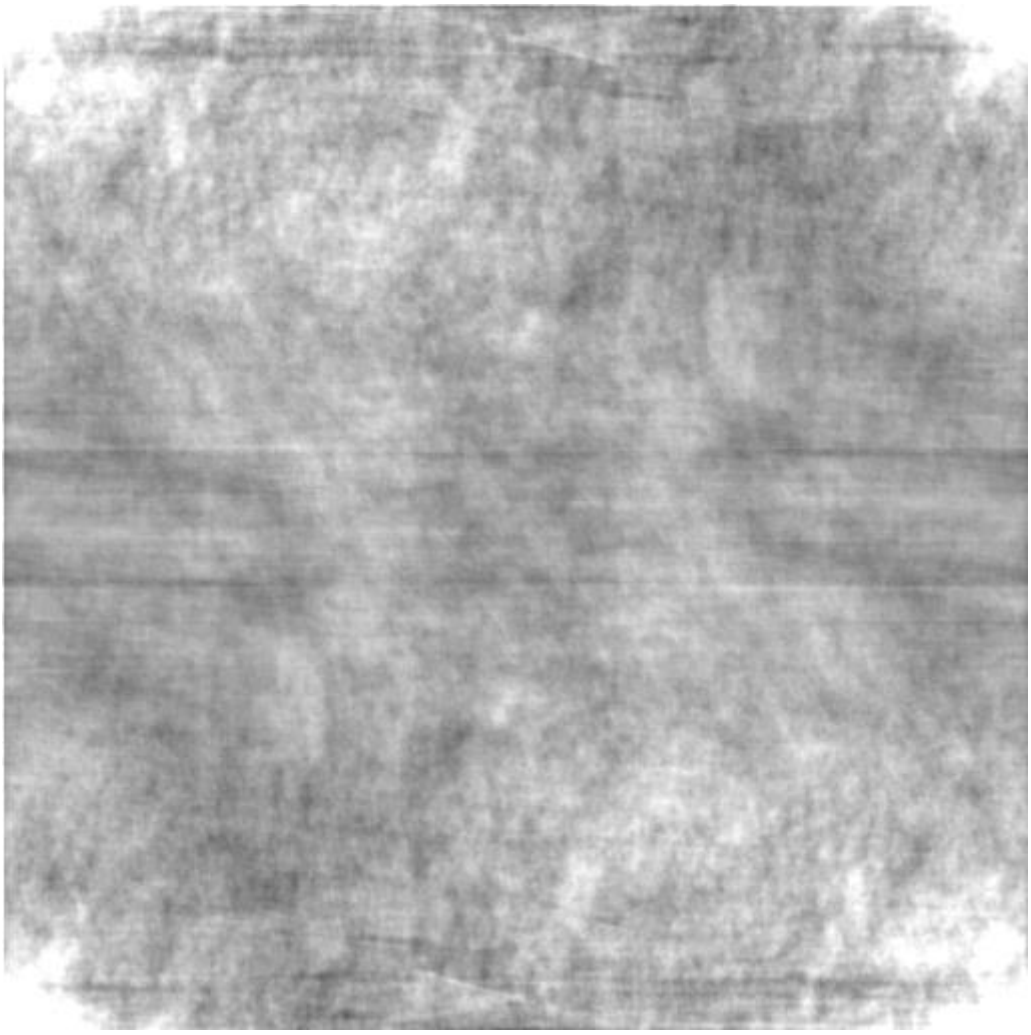
This was the resultant image:

Linearly-scaled, phase-reconstructed image

We multiplied the phase component by 0.1 and reconstructed the image in the spatial domain. This is how we did it:

```
reconstructed_phase = get_inverse_ft(image_magnitude, 0.1j * image_phase)
```

This is what the operation yielded:



Original image after multiplying its magnitude by 0.1

The resultant image was viciously distorted because the phase component carries the image details. So, multiplying it by 0.1 causes a huge loss of details.

Problem 3

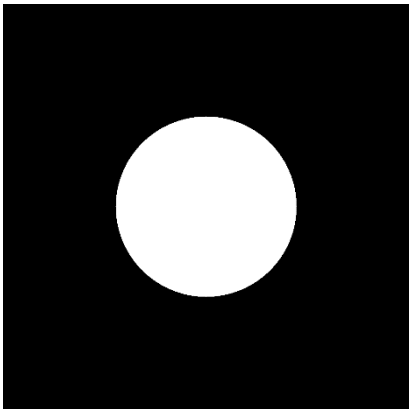
For this problem, we figured out that the notch filter can be used to produce all the other filters. So, we implemented it and the code looked like this:

```
def ideal_notch_reject_filter(image, center, radius):
    i, j = np.indices(image.shape)
    y, x = center
    distance = np.sqrt((i - y) ** 2 + (j - x) ** 2)
    filter_mask = np.where(distance <= radius, 0, 1)
    return filter_mask
```

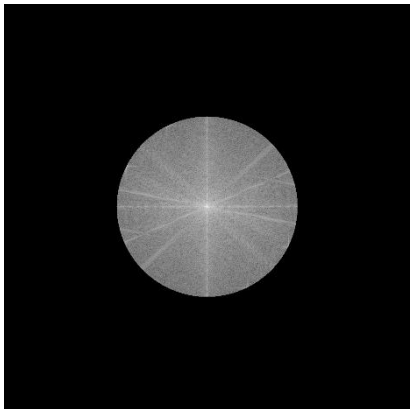
For the ideal low-pass filter, we tested out a couple of possible radii that reduce the noise and settled on a radius of 228 pixels as it had the highest PSNR when comparing it to the original, clear image. To apply a low-pass filter of this radius, we pass this value to the notch-reject filter function along with the center set as the image's center. The notch-reject filter returns a mask that rejects the area specified by the center and radius, so, to obtain a low-pass filter, we simply invert the mask. This is the code to this operation:

```
def ideal_lowpass_filter(image, radius):
    middle_y, middle_x = get_middle(image)
    center = (middle_y, middle_x)
    lowpass_filter = 1 - ideal_notch_reject_filter(image, center, radius)
    return lowpass_filter
```

These are the filter mask obtained, the filtered spectrum, and the filtered image in the frequency domain:



Low-pass filter mask



Filtered spectrum

Filtered image

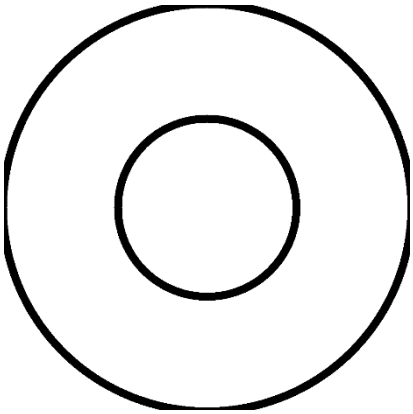
The PSNR for the filtered image when comparing it with the original, noise-free image turned out to be (28.081) which is not that high. The filter reduced the noise in the image, but in order to do that it had to wipe out a portion of the original image details. It kept only 77.75% of the original magnitude. The ringing effect produced by this filter is quite visible.

For the ideal band-reject filter, we chose to filter two bands with radii between 215 - 235 pixels and 505 - 525 pixels. For each band, we get two notch filters both centered at the center of the image, one

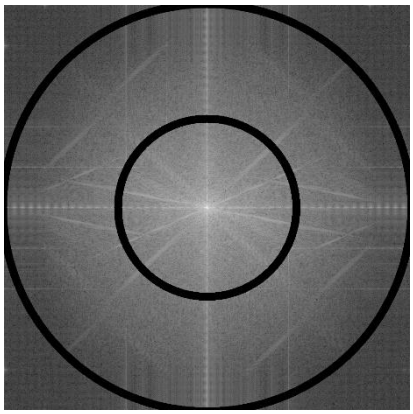
with the inner radius and the other with the outer radius, then we subtract them, giving us a band filter:

```
def ideal_band_reject_filter(image, radii):
    filter_mask = np.zeros(magnitude.shape)
    for inner_radius, outer_radius in radii:
        middle_y, middle_x = get_middle(image)
        center = (middle_y, middle_x)
        inner_filter_mask = ideal_notch_reject_filter(image, center, inner_radius)
        outer_filter_mask = ideal_notch_reject_filter(image, center, outer_radius)
        band_filter_mask = inner_filter_mask - outer_filter_mask
        filter_mask += band_filter_mask
    return 1 - filter_mask
```

We got much better results than the ideal lowpass filter here. It gave a PSNR value of 33.28 and it kept 95.38% of the original magnitude.



Band-reject filter mask



Filtered spectrum

Filtered image

We noticed that the band-reject filter produces deeper blacks which is closer to the original image and that is why it got a much better PSNR value. But that made the ringing effect a bit more visible due to the high contrast between the ringing effect and the actual image.

Low-pass filtered, zoomed-in image

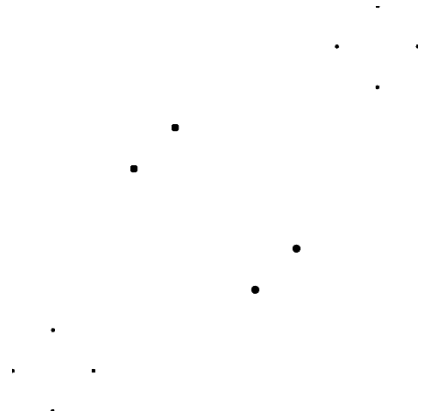
Band-reject filtered, zoomed-in image

Original, zoomed-in image

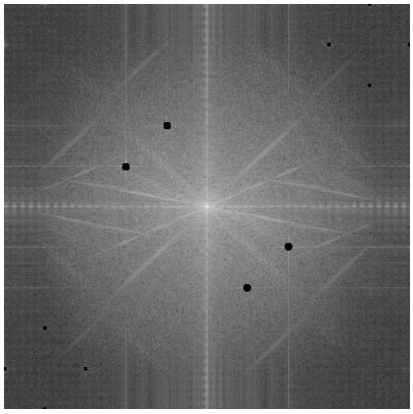
In the case of the notch-reject filter, we picked 12 notches to apply to the magnitude. We picked notches centered at the abnormalities in the magnitude (411, 307), (307, 411), (102, 819), (205, 921), (0, 922), (102, 1023) and their opposites about the center of the image to produce the notch-reject filter mask. This is our implementation for this process:

```
def notch_reject_filters_with_opposite(image, centers, radii):
    middle_y, middle_x = get_middle(image)
    filter_mask = np.ones(magnitude.shape)
    for center, radius in zip(centers, radii):
        y, x = center
        y_opposite = 2 * middle_y - y
        x_opposite = 2 * middle_x - x
        center_opposite = (y_opposite, x_opposite)
        filter_mask *= ideal_notch_reject_filter(image, center, radius)
        filter_mask *= ideal_notch_reject_filter(image, center_opposite, radius)
    return filter_mask
```

This filter produced the best results. We got a PSNR value of 41.56 and it kept 99.08% of the original magnitude.



Notch-reject filter mask



Filtered spectrum

Filtered image

The notch-reject filter produced even deeper blacks than the other two filters because it kept most of the original magnitude and it also had a way less visible ringing effect producing the best results and getting the closest image to the original.

Notch-reject filtered, zoomed-in image

Original, zoomed-in image