
TUTORIAL 2

Image Processing in the Frequency Domain

Pengolahan Citra - Semester Gasal 2023/2024

1 Frequency Domain

Di dalam *frequency domain*, nilai dan letak dari suatu citra dapat direpresentasikan sebagai hubungan sinusoidal yang bergantung pada frekuensi kemunculan piksel dalam suatu citra. Hal ini dapat menentukan apakah suatu piksel mengandung informasi lebih penting dan apakah terdapat suatu pola yang berulang.

Dalam pemrosesan citra, transformasi ke *frequency domain* disebut sebagai 2D Discrete Fourier Transformation atau 2D-DFT. Salah satu implementasinya, Fast Fourier Transform, dianggap sebagai algoritma DFT tercepat dengan kompleksitas $O(n \log n)$. Pada bagian selanjutnya, akan ditunjukkan bagaimana cara melakukan transformasi tersebut.

2 Fourier Transform

Dengan Fourier Transform, Kita dapat mengubah suatu citra pada *spatial domain* ke *frequency domain* menggunakan modul `fftpack` dari `Scipy`. Untuk suatu citra diskret dua dimensi, kita perlu menerapkan Fourier Transform yang sifatnya diskret dan ke dua arah (karena ada dua dimensi pada citra), yaitu 2D Discrete Fourier Transform.

Berikut adalah fungsi matematis untuk mengubah citra pada *spatial domain* $f(x, y)$ dengan ukuran (M, N) ke frequency domain $F(u, v)$:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-i2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

Sebaliknya, berikutnya adalah fungsi matematika ke untuk mengubah citra pada *frequency domain* ke *spatial domain* dengan invers DFT:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{i2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

Berikut adalah langkah-langkah untuk merepresentasikan suatu citra pada *frequency domain* dan sebaliknya:

```
1 # Assume that all packages have been imported
2
3 # Load image as grayscale
4 # Image credit to Sutirta Budiman
#   (https://unsplash.com/photos/DxmBSgUYKis)
5 image = color.rgb2gray(io.imread("raja_ampat.jpg"))

6
7 # Calculate DFT of the image
8 ft = fp.fft2(image)

9
10 # Calculate magnitude of imaginary and real numbers of the DFT
11 ft_norm = abs(ft)

12
13 # Scale the image (frequency domain) magnitude
14 ft_scale = np.log(1 + ft_norm)
```

```

15
16 # Shift low frequency to the center of image
17 ft_shift = fp.fftshift(ft_scale)
18
19 # Calculate IDFT to get the image back to spatial domain
20 ift = fp.ifft2(ft).real
21
22 # Show the differences
23 plt.figure(figsize=(20, 15))
24 plt.subplot(1, 4, 1); plt.imshow(image, cmap="gray");
    plt.title('Original'); plt.axis('off')
25 plt.subplot(1, 4, 2); plt.imshow(ft_scale, cmap="gray");
    plt.title('Hasil Fourier Transform'); plt.axis('off')
26 plt.subplot(1, 4, 3); plt.imshow(ft_shift, cmap="gray");
    plt.title('Hasil Fourier Transform (Shifted)');
    plt.axis('off')
27 plt.subplot(1, 4, 4); plt.imshow(ift, cmap="gray");
    plt.title('Hasil Inverse Fourier Transform');
    plt.axis('off')
28 plt.show()

```

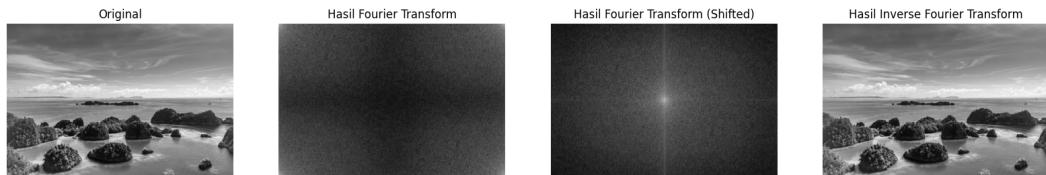


Figure 1: Contoh Fourier Transform

3 Spatial Filtering vs Frequency Filtering

Pada *spatial domain*, *filtering* dilakukan dengan mengkonvolusi citra dengan suatu *kernel* filter tertentu. Pada *frequency domain*, *filtering* dapat dilakukan dengan melakukan operasi perkalian pada citra dengan suatu *kernel* filter tertentu. Jika citra hasil *filtering* pada *frequency domain* ditransformasikan kembali ke *spatial domain* dengan invers DFT, maka hasil *filtering* dapat dilihat juga. Pada tutorial ini, akan dibahas tiga jenis filter, yaitu *low-pass*, *high-pass*, dan *notch*. Berikut adalah diagram yang menggambarkan bagaimana *filtering* dalam *domain frequency* bekerja.

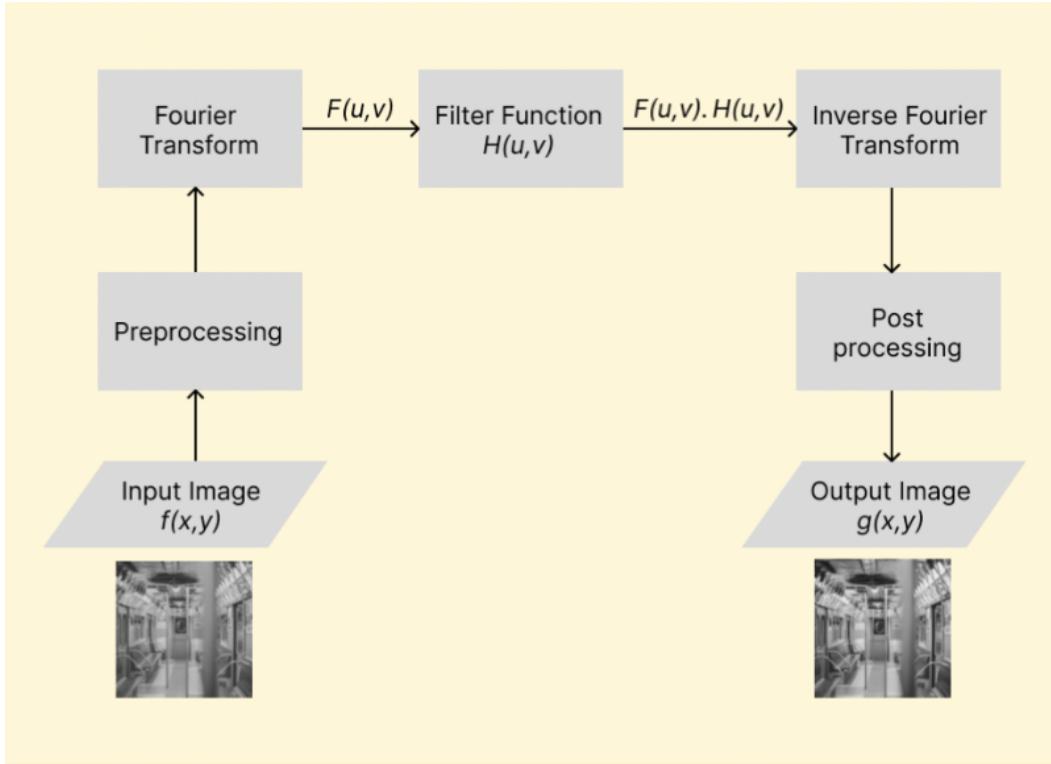


Figure 2: Proses *Filtering* Citra pada *Frequency Domain*

3.1 Low-Pass Filtering

Low-pass filtering digunakan untuk menyamarkan (*blurring*) atau memperhalus (*smoothing*) suatu citra. Citra yang telah ditransformasi ke *spatial domain* akan diminimalkan komponen berfrekuensi tingginya dan dimaksimalkan komponen berfrekuensi rendahnya. Pada umumnya, terdapat tiga teknik yang dapat digunakan, yakni *ideal low-pass filtering*, *butterworth low-pass filtering*, dan *gaussian low-pass filtering*. Perbedaan terletak pada seberapa tegas batas antara frekuensi yang diteruskan dan yang tidak. Berikut adalah ilustrasinya.

Ideal Low-pass Filter	Butterworth Low-pass Filter	Gaussian Low-pass Filter

Figure 3: Komparasi *Low-Pass Filtering*

Berikut adalah kode untuk menerapkan *low-pass filtering*.

```

1 # Assume that all packages have been imported
2
3 # Load the image (assuming the image is already in grayscale)
4 i1 = io.imread(image_path)
5 w, l = paddedsize(i1.shape[0], i1.shape[1])
6
7 # Perform ideal low-pass filtering (you can change it to
8 # gaussian/butterworth)
9 h = lpfilter('ideal', w, l, constanta * w)
10
11 # Calculate DFT
12 f = fp.fft2(i1,(w,l))
13
14 # Apply lowpass filter by multiplication
15 LPFS_image = h * f
16
17 # Calculate IDFT for spatial domain transformation
18 LPF_image = fp.ifft2(LPFS_image).real
19 LPF_image = LPF_image[:i1.shape[0],:i1.shape[1]]
20
21 # Shifting for fourier spectrum display
22 Fc = fp.fftshift(f)
23 Fcf = fp.fftshift(LPFS_image)
24
25 # Scaling for fourier spectrum display
26 S1 = np.log(1+abs(Fc))
27 S2 = np.log(1+abs(Fcf))
28
29 # Show image
30 plt.figure(figsize=(12,8))
31 plt.subplot(2,2,1); plt.imshow(i1, cmap='gray')
32 plt.title("Original")
33 plt.axis("off")
34 plt.subplot(2,2,2); plt.imshow(S1, cmap='gray')
35 plt.title("Original Fourier Spectrum")
36 plt.axis("off")
37 plt.subplot(2,2,3); plt.imshow(LPF_image, cmap='gray')
38 plt.title("Result {0} Filtering".format(filter_type))
39 plt.axis("off")
40 plt.subplot(2,2,4); plt.imshow(S2, cmap='gray')
41 plt.title("{0} Filtering Result Fourier
42 Spectrum".format(filter_type))
43 plt.axis("off")
44 plt.show()

```

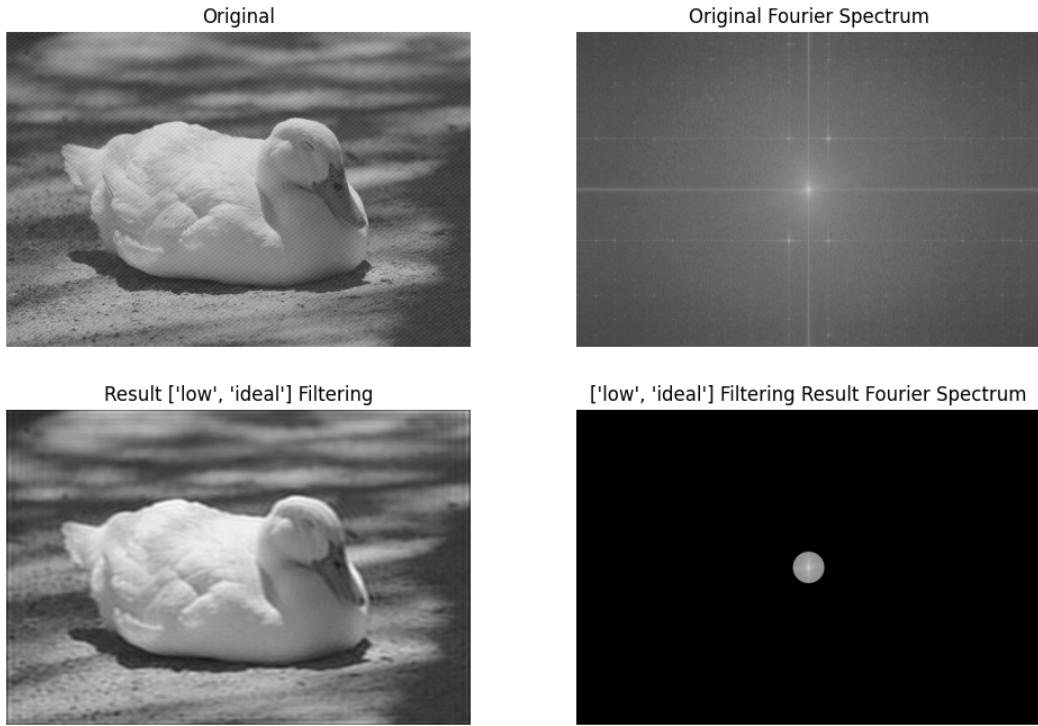


Figure 4: Contoh *Low-Pass Filtering*

3.2 High-Pass Filtering

High-pass filtering digunakan untuk menajamkan atau mempertegas suatu citra (*image sharpening*). Citra yang telah ditransformasi ke *frequency domain* akan diminimalkan komponen berfrekuensi rendahnya dan dimaksimalkan komponen berfrekuensi tingginya (kebalikan dari *low-pass filtering*). Pada umumnya, terdapat tiga teknik yang dapat digunakan, yaitu *ideal high-pass filtering*, *butterworth high-pass filtering*, dan *gaussian high-pass filtering*. Perbedaannya terletak pada seberapa tegas batas antara frekuensi yang diteruskan dan yang tidak. Berikut adalah ilustrasinya.

Ideal High-passFilter	Butterworth High-pass Filter	Gaussian High-pass Filter

Figure 5: Komparasi *High-Pass Filtering*

Berikut adalah contoh kode untuk menerapkan *high-pass filtering*.

```

1 # Assume that all packages have been imported
2
3 # Load the image (assuming the image is already in grayscale)
4 i1 = io.imread(image_path)
5 w, l = paddedsize(i1.shape[0], i1.shape[1])
6
7 # Perform butterworth high-pass filtering (you can change it
8 # to gaussian/ideal)
9 h = hpfilter('butterworth', w, l, constanta * w)
10
11 # Calculate DFT
12 f = fp.fft2(i1,(w,l))
13
14 # Apply lowpass filter by multiplication
15 LPFS_image = h * f
16
17 # Calculate IDFT for spatial domain transformation
18 LPF_image = fp.ifft2(LPFS_image).real
19 LPF_image = LPF_image[:i1.shape[0],:i1.shape[1]]
20
21 # Shifting for fourier spectrum display
22 Fc = fp.fftshift(f)
23 Fcf = fp.fftshift(LPFS_image)
24
25 # Scaling for fourier spectrum display
26 S1 = np.log(1+abs(Fc))
27 S2 = np.log(1+abs(Fcf))
28
29 # Show image
30 plt.figure(figsize=(12,8))
31 plt.subplot(2,2,1); plt.imshow(i1, cmap='gray')
32 plt.title("Original")
33 plt.axis("off")
34 plt.subplot(2,2,2); plt.imshow(S1, cmap='gray')
35 plt.title("Original Fourier Spectrum")
36 plt.axis("off")
37 plt.subplot(2,2,3); plt.imshow(LPF_image, cmap='gray')
38 plt.title("Result {0} Filtering".format(filter_type))
39 plt.axis("off")
40 plt.subplot(2,2,4); plt.imshow(S2, cmap='gray')
41 plt.title("{0} Filtering Result Fourier
42 Spectrum".format(filter_type))
43 plt.axis("off")
44 plt.show()

```

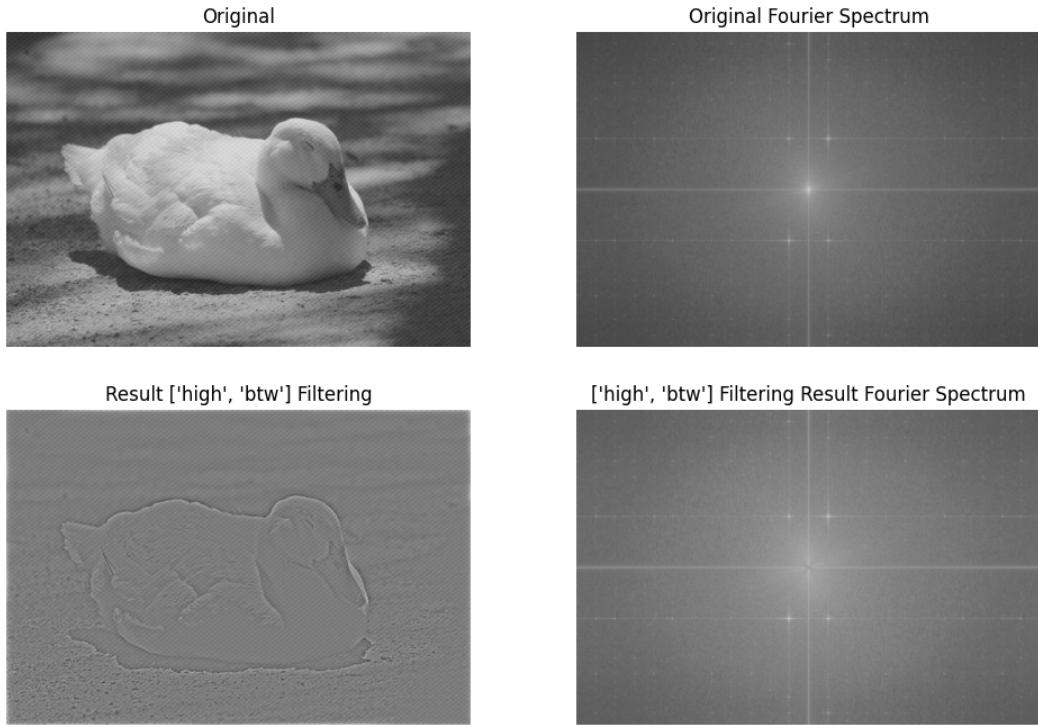


Figure 6: Contoh *High-Pass Filtering*

3.3 Notch Filtering

Notch filtering digunakan untuk menghilangkan *spectral noise* dari suatu citra. Filter ini juga dikenal sebagai *band-stop filter* dengan *narrow stop-band*. Hal ini dapat diartikan bahwa filter ini akan meminimalkan frekuensi tertentu yang telah dipilih (dan beberapa frekuensi tetangganya) dan memaksimalkan frekuensi lainnya. Berikut adalah contoh kode untuk menerapkan *notch filtering*.

```

1 # Assume that all packages have been imported
2
3 # Load the image (assuming the image is already in grayscale)
4 image = io.imread(image_path)
5 w, l = paddedsizes(image.shape[0], image.shape[1])
6
7 # Calculate DFT
8 F = fp.fft2(util.img_as_float(image), (w,l))
9
10 # Scaling & Shifting for fourier spectrum display
11 Fc = fp.fftshift(F)
12 S1 = np.log(1+abs(Fc))
13
14 # Create notch filter (notch function is provided in helper.py)
15 # You need to experiment to find the correct position
16 H1 = notch('btw', w, l, 40, -60, 40)

```

```

17 H2 = notch('btw', w, l, 40, -60, -30)
18 H3 = notch('btw', w, l, 40, 60, 40)
19 H4 = notch('btw', w, l, 40, 60, -30)
20
21 # Apply notch filter
22 FS_image = F*H1*H2*H3*H4
23
24 # Calculate IDFT for spatial domain transformation
25 F_image = fp.ifft2(FS_image).real
26 F_image = F_image[:image.shape[0], :image.shape[1]]
27
28 # Scaling & Shifting for fourier spectrum display
29 Fcf = fp.fftshift(FS_image)
30 S2 = np.log(1+abs(Fcf))
31
32 # Show image
33 plt.figure(figsize=(20,20))
34 plt.subplot(2,2,1); plt.imshow(image, cmap='gray')
35 plt.title('Noisy Image'); plt.axis("off")
36 plt.subplot(2,2,2); plt.imshow(S1, cmap='gray')
37 plt.title('Fourier Spectrum of Noisy Image'); plt.axis("off")
38 plt.subplot(2,2,3); plt.imshow(F_image, cmap='gray')
39 plt.title('Filtered Image'); plt.axis("off")
40 plt.subplot(2,2,4); plt.imshow(S2, cmap='gray')
41 plt.title('Fourier Spectrum After Filter'); plt.axis("off")
42 plt.show()

```

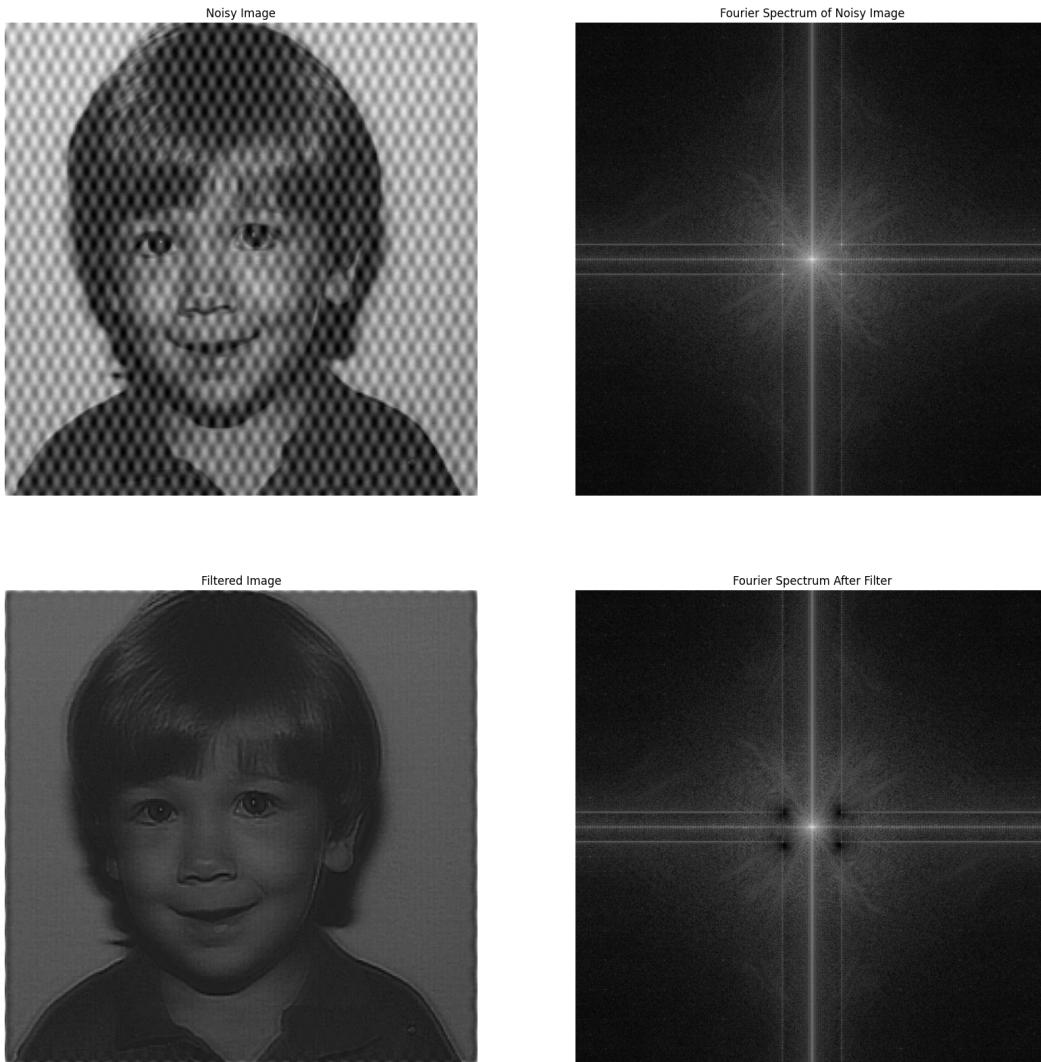


Figure 7: Contoh *Notch Filtering*

3.4 Cara Menambahkan *Noise*

Dalam menambahkan *noise* ke citra, kita bisa melakukannya secara langsung pada *spatial domain* atau bisa mencoba-coba pada *frequency domain*. Berikut adalah contoh kode untuk menambahkan *noise* langsung ke *spatial domain*.

```

1 # Assume that all packages have been imported
2
3 # Load the image (assuming the image is not grayscale)
4 image = io.imread(image_path, as_gray=True)*255
5 shape = image.shape[0], image.shape[1]
6
7 # Create periodic noise
8     (https://itqna.net/questions/1742/how-generate-noise-image-using-python)
9 noise = np.zeros(shape, dtype="float64").transpose()

```

```

9
10 X, Y = np.meshgrid(range(0, shape[0]), range(0, shape[1]))
11
12 A = 40
13 u0 = 45
14 v0 = 50
15 noise += A * np.sin(X * u0 + Y * v0)
16
17 A = -18
18 u0 = -45
19 v0 = 50
20 noise += A * np.sin(X * u0 + Y * v0)
21
22 # Add the noise into image
23 noisy_image = image + noise.transpose()
24
25 # Convert both to frequency domain to compare
26 # Show the comparison
27 plt.figure(figsize=(15, 10))
28 plt.subplot(2, 2, 1)
29 plt.imshow(image, cmap="gray")
30 plt.title("Original Image")
31 plt.axis("off")
32 plt.subplot(2, 2, 2)
33 plt.imshow(ft_shift, cmap="gray")
34 plt.title("Fourier Spectrum of Original Image")
35 plt.axis("off")
36 plt.subplot(2, 2, 3)
37 plt.imshow(noisy_image, cmap="gray")
38 plt.title("Noisy Image")
39 plt.axis("off")
40 plt.subplot(2, 2, 4)
41 plt.imshow(ft_shift1, cmap="gray")
42 plt.title("Fourier Spectrum of Noisy Image")
43 plt.axis("off")
44 plt.show()

```

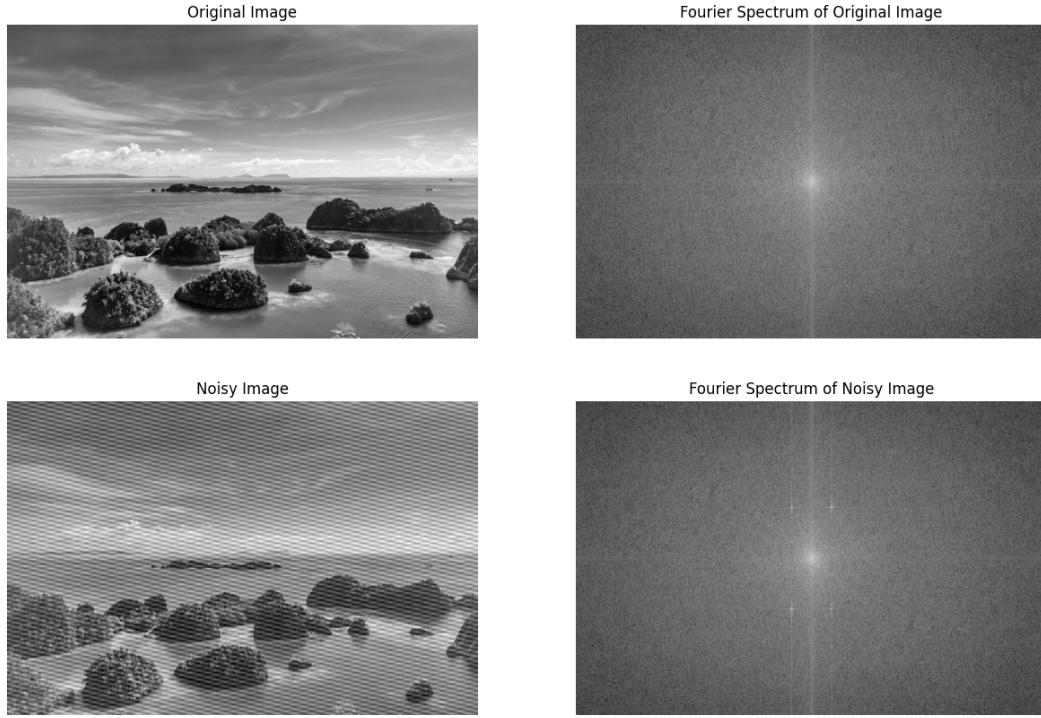


Figure 8: Contoh Menambahkan *Noise* ke *Spatial Domain*

Selanjutnya, berikut adalah contoh kode untuk menambahkan *noise* pada *frequency domain*.

```

1 # Assume that all packages have been imported
2
3 # Load the image (assuming the image is not grayscale)
4 image = io.imread(image_path, as_gray=True)*255
5
6 # Transform the image using 2D DFT
7 ft = fp.fft2(image)
8 ft_noise = ft.copy()
9
10 # Add noise, you can try adding it to another place
11 ft_noise[20:25, 20:25] = ft_noise.max()
12
13 # Calculate magnitude of imaginary and real numbers of the DFT
14 ft_norm = abs(ft)
15 ft_norm_noise = abs(ft_noise)
16
17 # Scale the image (frequency domain) magnitude
18 ft_scale = np.log(1 + ft_norm)
19 ft_scale_noise = np.log(1 + ft_norm_noise)
20
21 # Shift low frequency to the center of image

```

```

22 ft_shift = fp.fftshift(ft_scale)
23 ft_shift_noise = fp.fftshift(ft_scale_noise)
24
25 # Calculate IDFT to get the image (noisy) back to spatial
26 # domain
27 noisy_image = fp.ifft2(ft_noise).real
28
29 # Comparing image
30 plt.figure(figsize=(15, 10))
31 plt.subplot(2, 2, 1)
32 plt.imshow(image, cmap="gray")
33 plt.title("Original Image")
34 plt.axis("off")
35 plt.subplot(2, 2, 2)
36 plt.imshow(ft_shift, cmap="gray")
37 plt.title("Fourier Spectrum of Original Image")
38 plt.axis("off")
39 plt.subplot(2, 2, 3)
40 plt.imshow(noisy_image, cmap="gray")
41 plt.title("Noisy Image")
42 plt.axis("off")
43 plt.subplot(2, 2, 4)
44 plt.imshow(ft_shift_noise, cmap="gray")
45 plt.title("Fourier Spectrum of Noisy Image")
46 plt.axis("off")
47 plt.show()

```

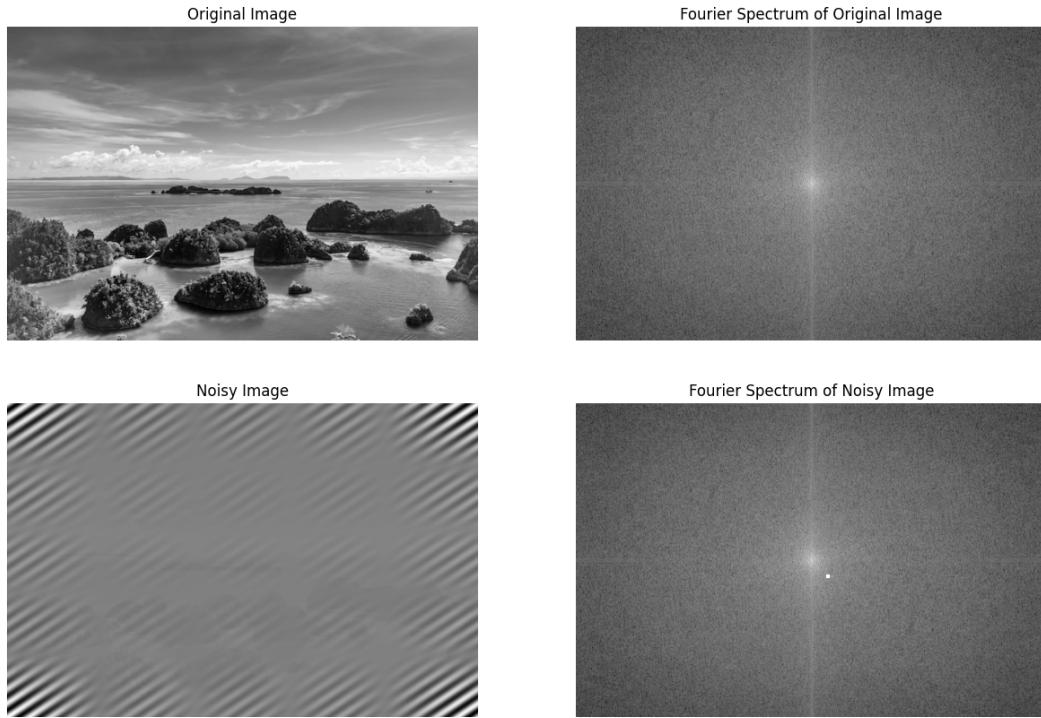


Figure 9: Contoh Menambahkan *Noise* ke *Frequency Domain*