ข้อ 1

Notch filters

```
def notch_filter(rows, cols, r, filter_type="low-pass"):
    center_x, center_y = rows // 2, cols // 2
    filter = np.zeros((rows, cols), np.float32)

    for x in range(rows):
        for y in range(cols):
            distance = np.sqrt((x - center_x)**2 + (y - center_y)**2)
            if filter_type == "low-pass":
                if distance <= r:
                    filter[x, y] = 1
            elif filter_type == "high-pass":
                if distance > r:
                    filter[x, y] = 1

    return filter

images = ["flower1.jpg", "fruit.jpg"]
for each in images:
    img = cv2.imread(each, cv2.IMREAD_GRAYSCALE)

    # Discrete Fourier Transform
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shifted = np.fft.fftshift(dft)

    # Create the filter masks
    r_values = [10, 50, 100]
    for r in r_values:
        low_pass_filter = notch_filter(img.shape[0], img.shape[1], r, "low-pass")
        high_pass_filter = notch_filter(img.shape[0], img.shape[1], r, "high-
pass")


        # Apply the filters
        low_pass = dft_shifted * low_pass_filter[:, :, np.newaxis]
        high_pass = dft_shifted * high_pass_filter[:, :, np.newaxis]

        # Inverse DFT for Low-Pass filtered image
        idft_low_pass = np.fft.ifftshift(low_pass)
        img_reconstructed_low = cv2.idft(idft_low_pass,
flags=cv2.DFT_REAL_OUTPUT)
```

```
        img_reconstructed_low = cv2.normalize(img_reconstructed_low, None, 0,
255, cv2.NORM_MINMAX).astype(np.uint8)

        # Inverse DFT for High-Pass filtered image
        idft_high_pass = np.fft.ifftshift(high_pass)
        img_reconstructed_high = cv2.idft(idft_high_pass,
flags=cv2.DFT_REAL_OUTPUT)
        img_reconstructed_high = cv2.normalize(img_reconstructed_high, None, 0,
255, cv2.NORM_MINMAX).astype(np.uint8)

        output_folder = "./notch_filters"
        if not os.path.exists(output_folder):
            os.makedirs(output_folder)

        cv2.imwrite(os.path.join(output_folder, f"{each} low pass r = {r}.jpg"),
img_reconstructed_low)
        cv2.imwrite(os.path.join(output_folder, f"{each} high pass r = {r}.jpg"),
img_reconstructed_high)
```

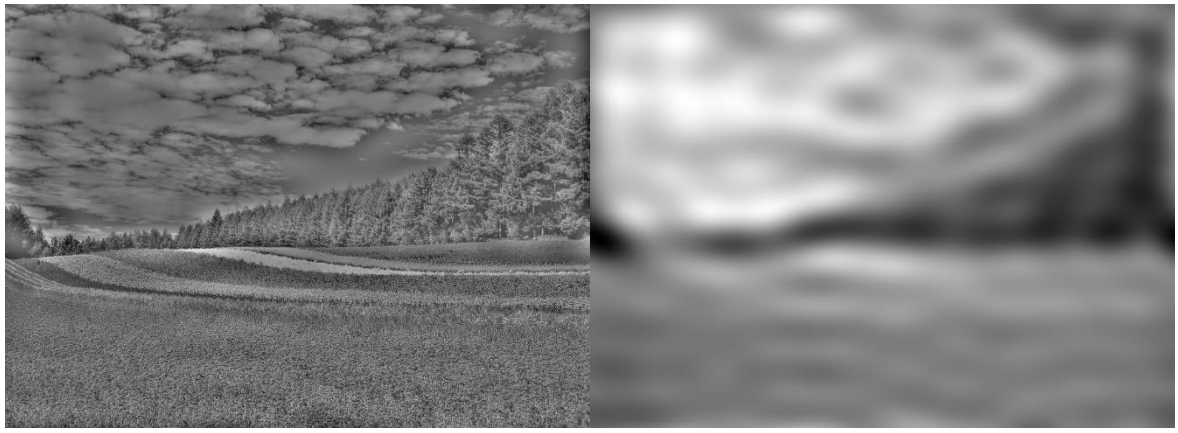
ภาพต้นฉบับ

flower1.jpg

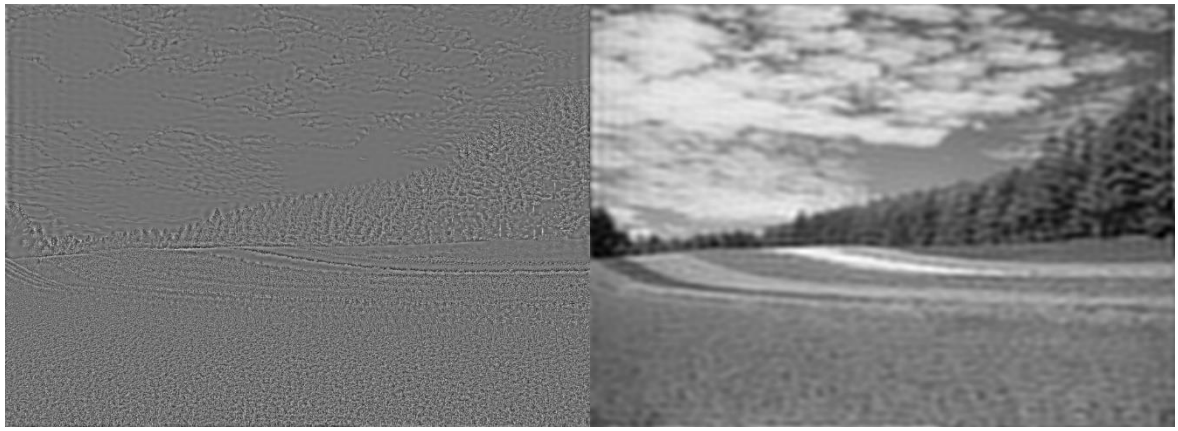
fruit.jpg

high pass filter

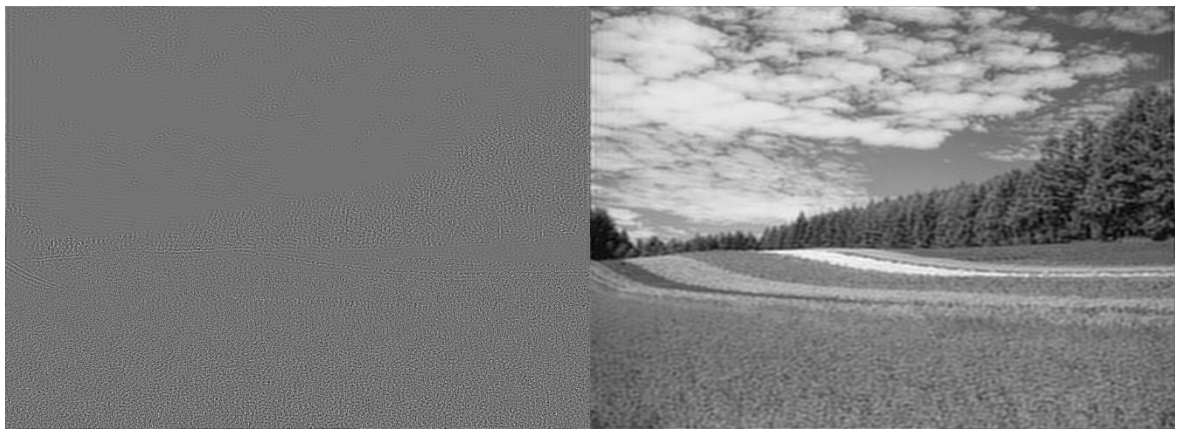
low pass filter



r = 10



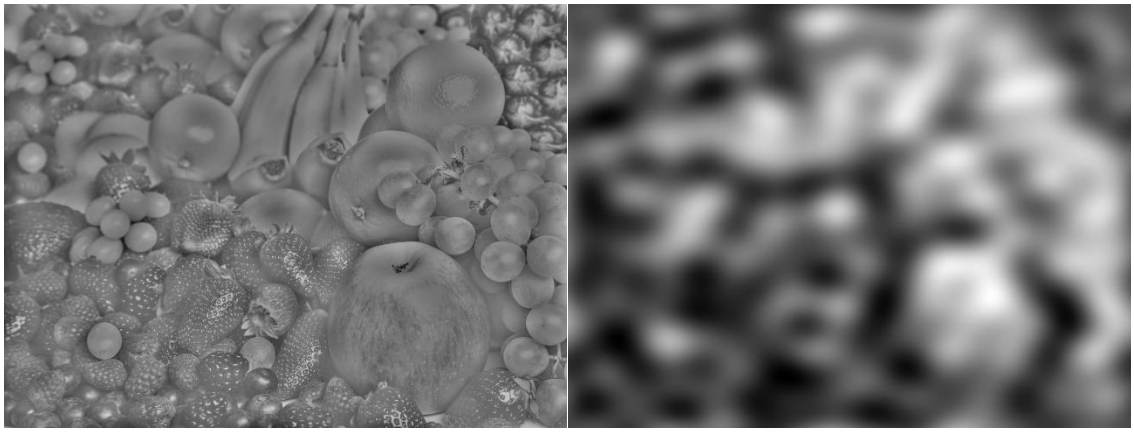
r = 50



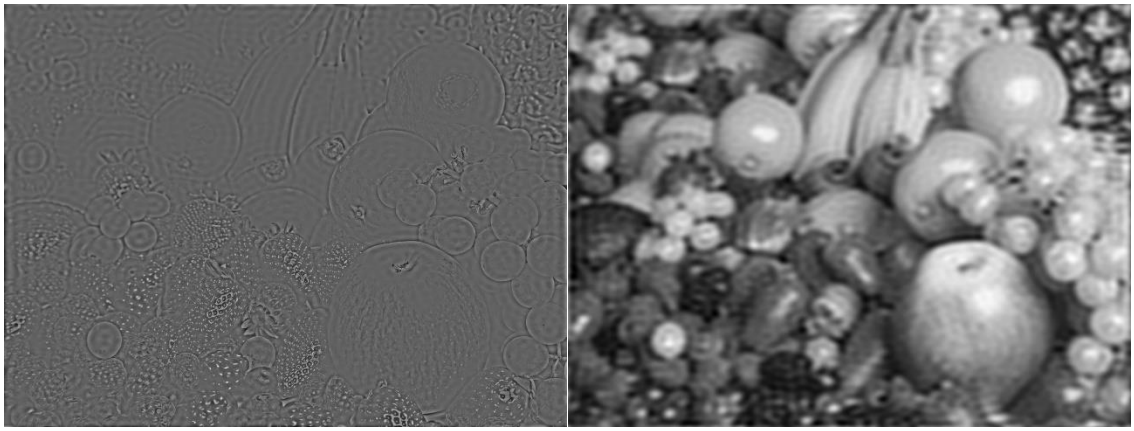
r = 100

high pass filter

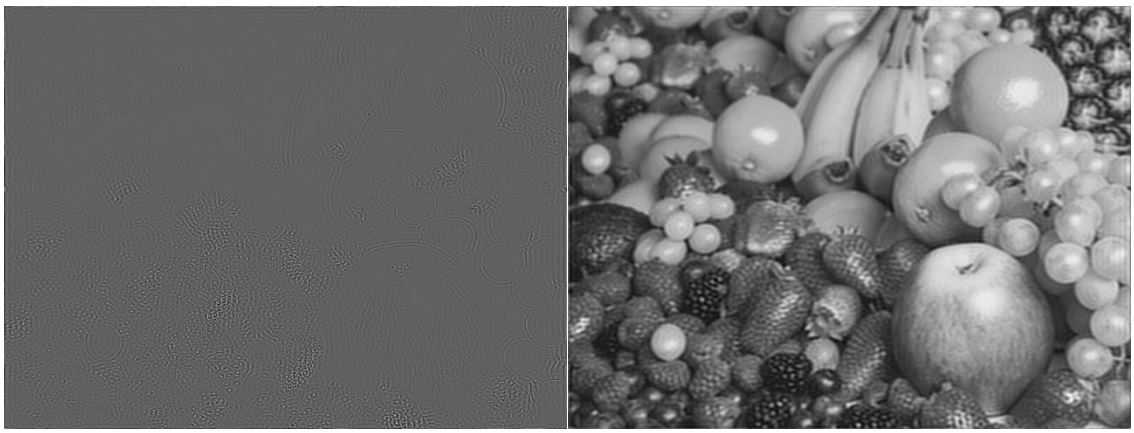
low pass filter



r = 10



r = 50



r = 100

อธิบายเพิ่มเติม : การทำ notch filters เป็นการ filter ค่าจากความถี่ โดย low-pass filter จะทำการ filter ค่าความถี่ที่มากกว่า r ออกและเก็บค่าที่อยู่ในรัศมี r ไว้ ส่วน high-pass filter จะตรงกันข้ามคือทำการ filter ค่าความถี่ที่น้อยกว่า r ออกและเก็บค่าที่อยู่นอกรัศมี r ไว้ ซึ่งการทำอย่างนี้ส่งผลให้ภาพจาก low-pass filter จะมีลักษณะที่เบลอเพราะส่วนที่ไม่มีการเปลี่ยนแปลง (ส่วนที่ไม่ใช่ขอบ) จะมีค่าความถี่ที่ต่ำและ low-pass filter เก็บไว้ ส่วนภาพที่ได้จาก high-pass filter จะมีลักษณะเน้นขอบในรูปภาพ เพราะ ส่วนที่มีการเปลี่ยนแปลง (ส่วนที่เป็นขอบๆ) ความถี่จะสูงแล้ว high-pass filter เก็บไว้

ข้อ 2

Gaussian filters

```
def gaussian_lpf(rows, cols, D0):
    center_x, center_y = rows // 2, cols // 2
    filter = np.zeros((rows, cols, 2), np.float32)
    for x in range(rows):
        for y in range(cols):
            distance = np.sqrt((x - center_x) ** 2 + (y - center_y) ** 2)
            filter[x, y] = np.exp(-(distance ** 2) / (2 * (D0 ** 2)))
    return filter

def gaussian_hpf(rows, cols, D0):
    return 1 - gaussian_lpf(rows, cols, D0)

images = ["flower1.jpg", "fruit.jpg"]
for each in images:
    # Load the image in grayscale
    img = cv2.imread(each, cv2.IMREAD_GRAYSCALE)

    # Apply FFT and shift the zero-frequency component to the center
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shifted = np.fft.fftshift(dft)

    # Apply Gaussian Filters
    D0_values = [10, 50, 100]
    for D0 in D0_values:
        # Low-pass filter
        lpf = gaussian_lpf(img.shape[0], img.shape[1], D0)
        filtered_lpf = dft_shifted * lpf
        idft_lpf = np.fft.ifftshift(filtered_lpf)
        img_reconstructed_lpf = cv2.idft(idft_lpf, flags=cv2.DFT_REAL_OUTPUT)
        img_display_lpf = cv2.normalize(img_reconstructed_lpf, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)
```

```
        # High-pass filter
        hpf = gaussian_hpf(img.shape[0], img.shape[1], D0)
        filtered_hpf = dft_shifted * hpf
        idft_hpf = np.fft.ifftshift(filtered_hpf)
        img_reconstructed_hpf = cv2.idft(idft_hpf, flags=cv2.DFT_REAL_OUTPUT)
        img_display_hpf = cv2.normalize(img_reconstructed_hpf, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)
    #     cv2.imshow(f'Gaussian HPF D0={D0}', img_display_hpf)

        output_folder = "./gaussian_filters"
        if not os.path.exists(output_folder):
            os.makedirs(output_folder)

        cv2.imwrite(os.path.join(output_folder, f"{each} low pass d0 =
{D0}.jpg"), img_display_lpf)
        cv2.imwrite(os.path.join(output_folder, f"{each} high pass d0 =
{D0}.jpg"), img_display_hpf)
```

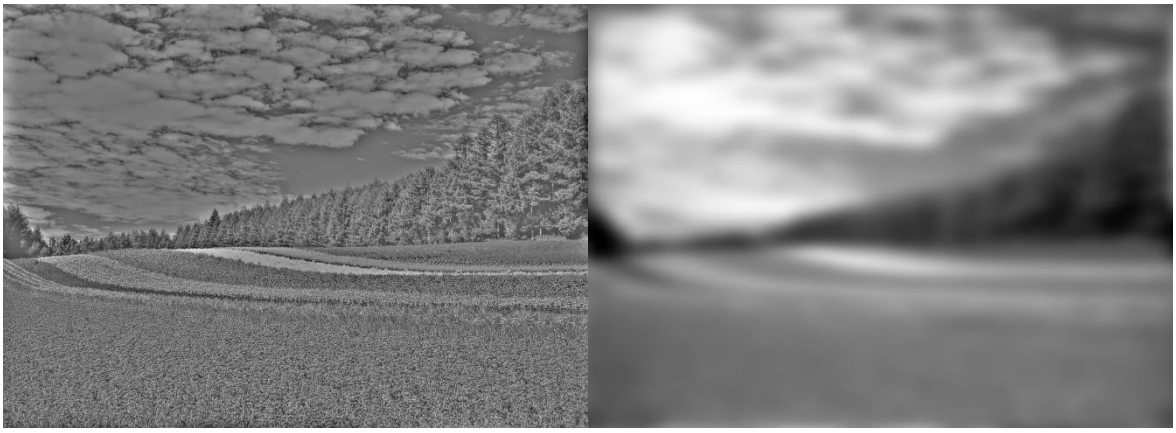
ภาพต้นฉบับ

flower1.jpg

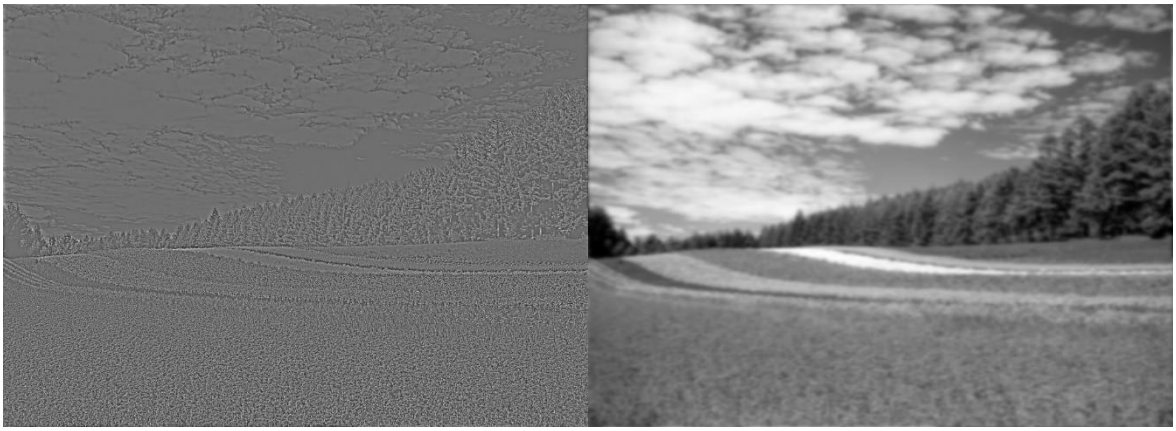
fruit.jpg

high pass filter

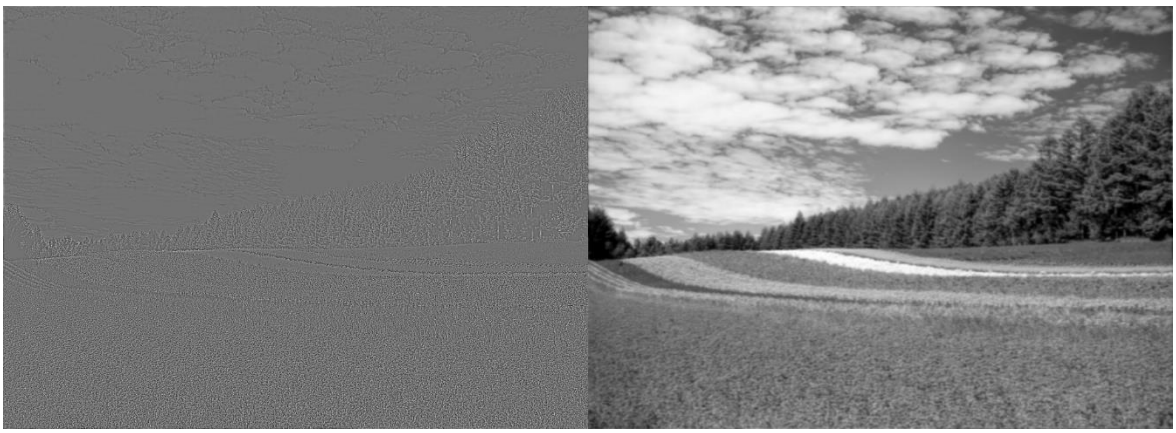
low pass filter



r = 10



r = 50



r = 100

อธิบายเพิ่มเติม : การทำ gaussian filter จะคล้ายๆ notch filter โดย low-pass filter จะทำการ filter ค่าความถี่ที่มากกว่า r ออกและเก็บค่าที่อยู่ในรัศมี r ไว้ ส่วน high-pass filter จะตรงกันข้ามคือทำการ filter ค่าความถี่ที่น้อยกว่า r ออกและเก็บค่าที่อยู่นอกรัศมี r ไว้ แต่จะแตกต่างกันตรงที่ gaussian filter จะไม่ทำการ filter แบบสีดำทันที แต่จะค่อยเปลี่ยนเป็นสีดำ หรือ ค่อยๆเปลี่ยนเป็นสีขาว (low-pass กับ high-pass ตามลำดับ) แต่คิดว่าที่ภาพ high-pass r = 50 และ r = 100 มองเห็นได้ไม่ชัดเพราะ r มีขนาดที่ใหญ่เกินไปทำให้ภาพใน frequency domain ส่วนใหญ่ถูกตัดทิ้งออกไป

3. Remove periodic noise from the given images using any of the filters that you think are the most suitable ones to be used.

noisy_flower1_horizontal.jpg

```
import os
import cv2
import numpy as np

import matplotlib.pyplot as plt

def gaussian_notch_reject(shape, d0=30, width=10):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2

    y, x = np.ogrid[:rows, :cols]
    d2_from_center = (x - ccol)**2 + (y - crow)**2
    d2_from_d0_above = (x - ccol)**2 + (y - (crow-d0))**2
    d2_from_d0_below = (x - ccol)**2 + (y - (crow+d0))**2

    # Gaussian masks for the two dots
    mask_above = np.exp(-d2_from_d0_above / (2*width**2))
    mask_below = np.exp(-d2_from_d0_below / (2*width**2))

    # Final combined mask (1 - mask) to reject the specific frequencies
    mask_gaussian = 1 - (mask_above + mask_below)

    # Extend the shape for real and imaginary parts
    mask_gaussian = mask_gaussian[:, :, np.newaxis]

    return mask_gaussian

img = cv2.imread('noisy_flower1_horizontal.jpg', cv2.IMREAD_GRAYSCALE)


dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shifted = np.fft.fftshift(dft)
```

```
notch70 = gaussian_notch_reject(img.shape, 70, 25)
notch140 = gaussian_notch_reject(img.shape, 140, 25)

notch = notch70 * notch140
# Apply the notch filter
fshift = dft_shifted * notch

# Inverse FFT to get the image back
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift, flags=cv2.DFT_REAL_OUTPUT)
img_back = cv2.normalize(img_back, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)

magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shifted[:, :, 0],
dft_shifted[:, :, 1]))
magnitude_spectrum_normalized = cv2.normalize(magnitude_spectrum, None, 0, 255,
cv2.NORM_MINMAX)

# Compute the magnitude of the filtered frequency domain
magnitude_values_filtered = cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1])
magnitude_spectrum_filtered = 20 * np.log(magnitude_values_filtered + 1)
magnitude_spectrum_filtered_normalized =
cv2.normalize(magnitude_spectrum_filtered, None, 0, 255, cv2.NORM_MINMAX)

cv2.imshow('Original Magnitude Spectrum',
magnitude_spectrum_normalized.astype(np.uint8))
cv2.imshow('Filtered Magnitude Spectrum',
magnitude_spectrum_filtered_normalized.astype(np.uint8))


output_folder = "./noisy"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

cv2.imwrite(os.path.join(output_folder, f"horizontal image d0 = 95,190 width =
25.jpg"), img_back)
cv2.imwrite(os.path.join(output_folder, f"horizontal magnitude_spectrum.jpg"),
magnitude_spectrum_normalized)
cv2.imwrite(os.path.join(output_folder, f"horizontal magnitude_spectrum_filtered
d0 = 95,190 width = 25.jpg"), magnitude_spectrum_filtered_normalized)
```

noisy_flower1_vertical.jpg

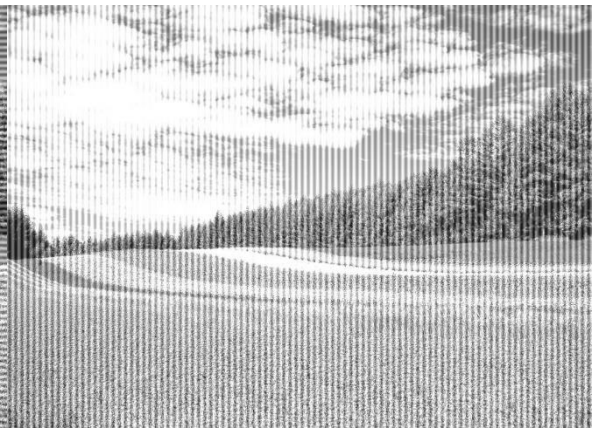
```
import os
import cv2
import numpy as np

import matplotlib.pyplot as plt

def gaussian_notch_reject(shape, d0=30, width=10):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2

    y, x = np.ogrid[:rows, :cols]
    d2_from_center = (x - ccol)**2 + (y - crow)**2

    d2_from_d0_right = (x - (ccol-d0))**2 + (y - (crow))**2
    d2_from_d0_left = (x - (ccol+d0))**2 + (y - (crow))**2

    # Gaussian masks for the two dots
    mask_above = np.exp(-d2_from_d0_right / (2*width**2))
    mask_below = np.exp(-d2_from_d0_left / (2*width**2))

    # Final combined mask (1 - mask) to reject the specific frequencies
    mask_gaussian = 1 - (mask_above + mask_below)

    # Extend the shape for real and imaginary parts
    mask_gaussian = mask_gaussian[:, :, np.newaxis]

    return mask_gaussian

img = cv2.imread('noisy_flower1_vertical.jpg', cv2.IMREAD_GRAYSCALE)

dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shifted = np.fft.fftshift(dft)

notch1 = gaussian_notch_reject(img.shape,95, 25)
notch2 = gaussian_notch_reject(img.shape, 190, 25)

notch = notch1 * notch2

# Apply the notch filter
fshift = dft_shifted * notch

# Inverse FFT to get the image back
f_ishift = np.fft.ifftshift(fshift)
```

```
img_back = cv2.idft(f_ishift, flags=cv2.DFT_REAL_OUTPUT)
img_back = cv2.normalize(img_back, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)

magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shifted[:, :, 0],
dft_shifted[:, :, 1]))
magnitude_spectrum_normalized = cv2.normalize(magnitude_spectrum, None, 0, 255,
cv2.NORM_MINMAX)

# Compute the magnitude of the filtered frequency domain
magnitude_values_filtered = cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1])
magnitude_spectrum_filtered = 20 * np.log(magnitude_values_filtered + 1)
magnitude_spectrum_filtered_normalized =
cv2.normalize(magnitude_spectrum_filtered, None, 0, 255, cv2.NORM_MINMAX)

output_folder = "./noisy"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

cv2.imwrite(os.path.join(output_folder, f"vertical image d0 = 95,190 width =
25.jpg"), img_back)
cv2.imwrite(os.path.join(output_folder, f"vertical magnitude_spectrum.jpg"),
magnitude_spectrum_normalized)
cv2.imwrite(os.path.join(output_folder, f"vertical magnitude_spectrum_filtered d0
= 95,190 width = 25.jpg"), magnitude_spectrum_filtered_normalized)
```

ภาพต้นฉบับ

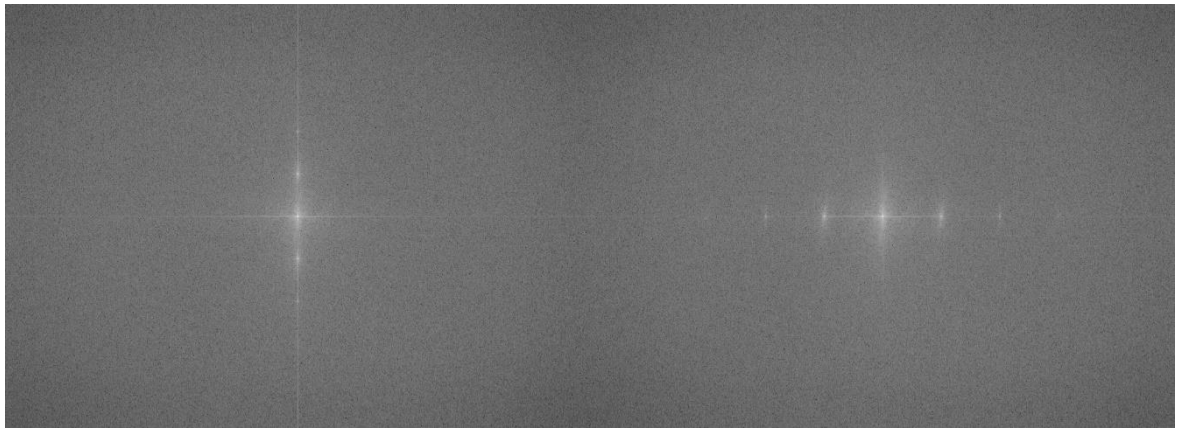
noisy_flower1_horizontal.jpg

noisy_flower1_vertical.jpg



Horizontal magnitude spectrum

Vertical magnitude spectrum

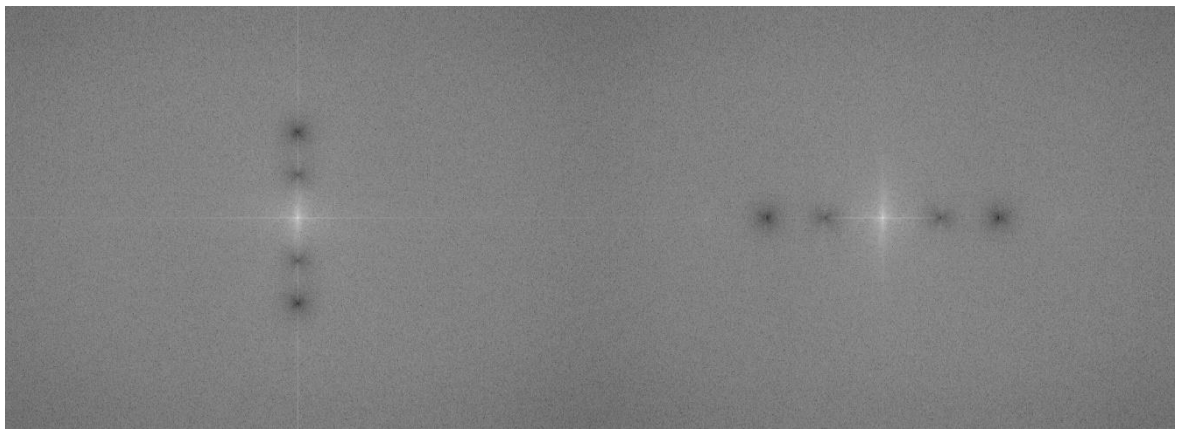


Gaussian filter horizontal d0 = 70, 140 width = 25

Gaussian filter vertical d0 = 95, 190 width = 25

Filtered horizontal magnitude spectrum

Filtered vertical magnitude spectrum



อธิบายเพิ่มเติม : ภาพที่กำหนดมามี noise ที่มีลักษณะเป็นเส้นในแนวนอนและแนวตั้งที่ห่างด้วยความยาวเท่าๆกัน ทำให้การใช้ notch filter มีความเหมาะสมมากในการลบ noise ประเภทนี้ และเหตุผลที่ใช้ gaussian เนื่องจากลักษณะจุด noise บน magnitude spectrum มีลักษณะเป็นวงกลม ซึ่งหลังจากลบ noise ออกจะทำให้ภาพมีลักษณะที่ชัดเจนมากยิ่งขึ้นเส้น noise มองเห็นได้ยากขึ้น แต่อาจมีหลงเหลือบางจากการใช้ notch filter ที่ลบออกไม่หมด