Image Processing in Frequency Domain

May 10, 2025

Abstract

This document describes an image processing workflow that involves adding periodic noise to an image, filtering in the frequency domain, and evaluating the results using PSNR. The process demonstrates how to remove structured noise using Fourier transform techniques.

1 Libraries Used

- `cv2` (OpenCV): For image loading and basic processing
- `numpy`: For numerical operations and array manipulation
- `math`: For mathematical functions like logarithms
- `matplotlib.pyplot`: For image visualization and plotting
- `matplotlib.colors.LogNorm`: For logarithmic scaling of colormaps
- `itertools.chain`: For efficient iteration over multiple ranges

2 Step-by-Step Process

2.1 Step 1: Import Libraries

```
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from itertools import chain
```

2.2 Step 2: Download Images

The image is downloaded from a GitHub repository using wget command.

```
!wget https://raw.githubusercontent.com/AsadiAhmad/Filtering-in-
   Frequency-Domain/main/Pictures/original_image.png -O
   original_image.png
```

2.3 Step 3: Load Image

The image is loaded in grayscale mode and displayed.

```
original_image = cv2.imread("original_image.png", cv2.
   IMREAD_GRAYSCALE)

plt.imshow(original_image, cmap='gray')
plt.show()
```

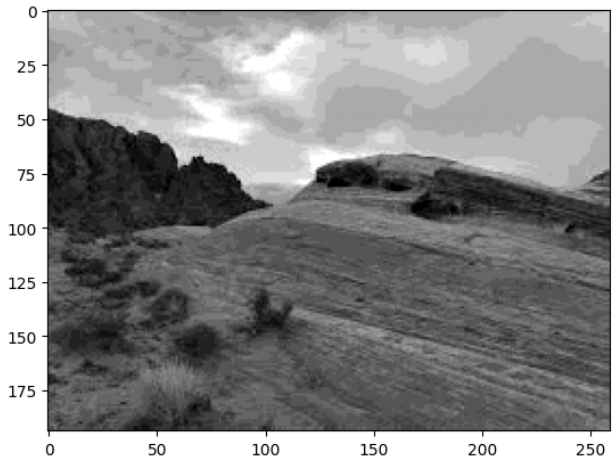

Figure 1: Original grayscale image

2.4 Step 4: Add Periodic Noise

A periodic noise function is created and added to the original image. The noise pattern consists of sinusoidal components in both x and y directions.

```
def f(x,y):
    return np.sin((1/2)*np.pi*x)+np.cos((1/3)*np.pi*y)

X, Y = original_image.shape
noise = np.zeros((X, Y))
for i in range(X):
    for j in range(Y):
        noise[i,j] = f(i,j)*100
```

```
noisy_image = original_image + noise

plt.imshow(noisy_image, cmap='gray')
plt.show()
```

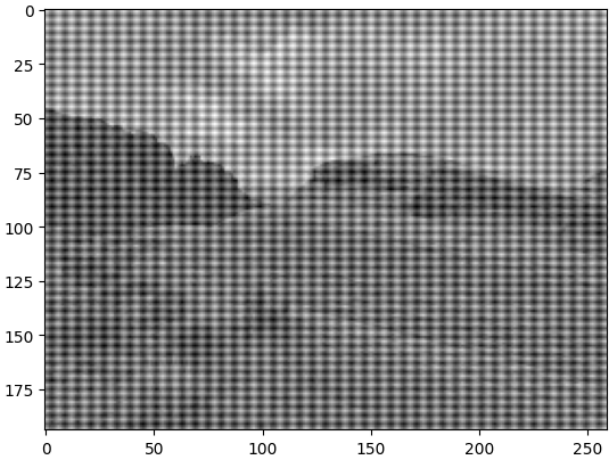

Figure 2: Image with added periodic noise

2.5 Step 5: Convert Image into the Frequency Domain

The image is transformed to frequency domain using Fast Fourier Transform (FFT) to analyze its frequency components.

```
def convert_image_frequency(noisy_image):
    f_transform = np.fft.fft2(noisy_image)
    f_shifted = np.fft.fftshift(f_transform)
    return f_shifted

f_shifted = convert_image_frequency(noisy_image)

magnitude_spectrum = 20 * np.log(np.abs(f_shifted) + 1)
plt.figure(figsize=[13, 6])
plt.subplot(121),plt.imshow(noisy_image, cmap='gray'),plt.title('
   noisy_image'), plt.axis('off');
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap='gray'),plt.
   title('magnitude_spectrum'), plt.axis('off');
plt.show()
```

2.6 Step 6: Define a Cross Filter Mask

A cross-shaped filter mask is created to remove the periodic noise components visible in the frequency domain.

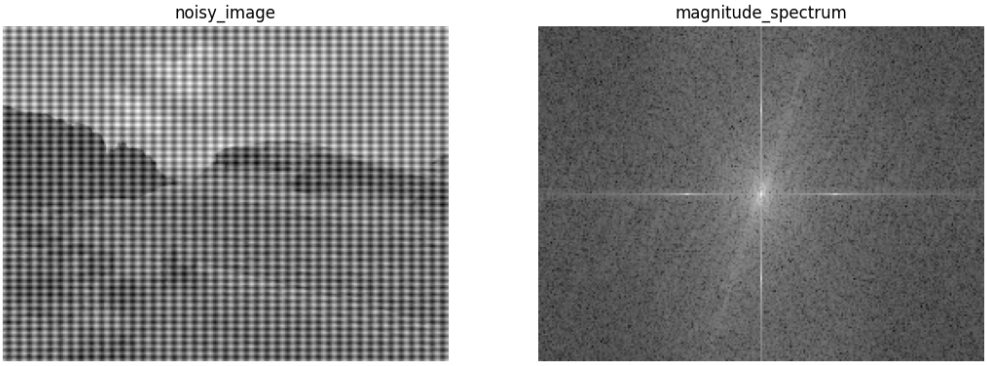

Figure 3: Noisy image (left) and its frequency spectrum (right)

```python
def define_cross_filter_mask(shape, circle_reduce):
    height, width = shape
    filter_mask = np.ones((height, width), dtype=np.uint8)
    rectangle_width = (width // 2) - circle_reduce
    rectangle_height = (height // 2) - circle_reduce

    for i in chain(range(0, rectangle_height), range(height -
       rectangle_height, height)):
        for j in range(rectangle_width, width - rectangle_width):
            filter_mask[i, j] = 0

    for i in range(rectangle_height, height - rectangle_height):
        for j in chain(range(0, rectangle_width), range(width -
           rectangle_width, width)):
            filter_mask[i, j] = 0

    return filter_mask

mask = define_cross_filter_mask(f_shifted.shape, 2)
frequency_filtered = f_shifted * mask

mask_visualization = mask * 255
magnitude_spectrum_filtered = magnitude_spectrum * mask

plt.figure(figsize=[13, 6])
plt.subplot(121), plt.imshow(mask_visualization, cmap='gray'), plt.
   title('Mask'), plt.axis('off')
plt.subplot(122), plt.imshow(magnitude_spectrum_filtered, cmap='gray
   '), plt.title('Filtered Frequency'), plt.axis('off')
plt.show()
```

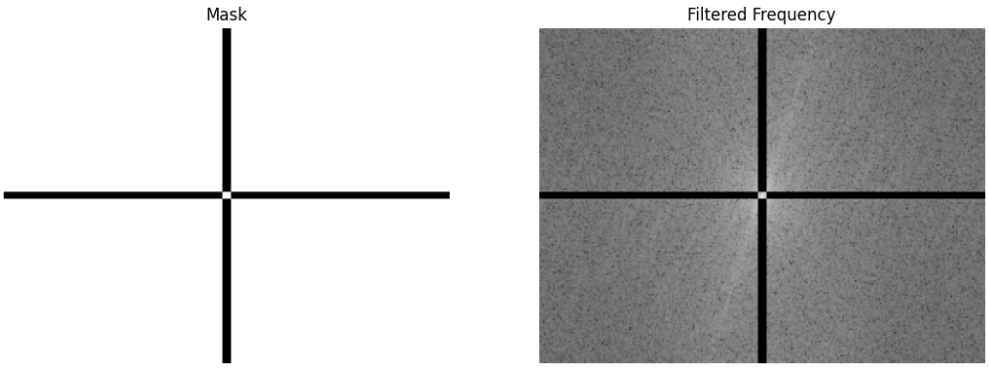

Figure 4: Filter mask (left) and filtered frequency spectrum (right)

2.7 Step 7: Inverse FFT

The filtered frequency domain representation is converted back to spatial domain using inverse FFT.

```
frequency_ishift = np.fft.ifftshift(frequency_filtered)
image_reconstructed = np.fft.ifft2(frequency_ishift)
image_reconstructed = np.abs(image_reconstructed)

plt.figure(figsize=(10, 4))
plt.subplot(121), plt.imshow(noisy_image, cmap="gray"), plt.title("
   Noisy Image"), plt.axis('off')
plt.subplot(122), plt.imshow(image_reconstructed, cmap="gray"), plt.
   title("Reconstructed Image"), plt.axis('off')
plt.show()
```

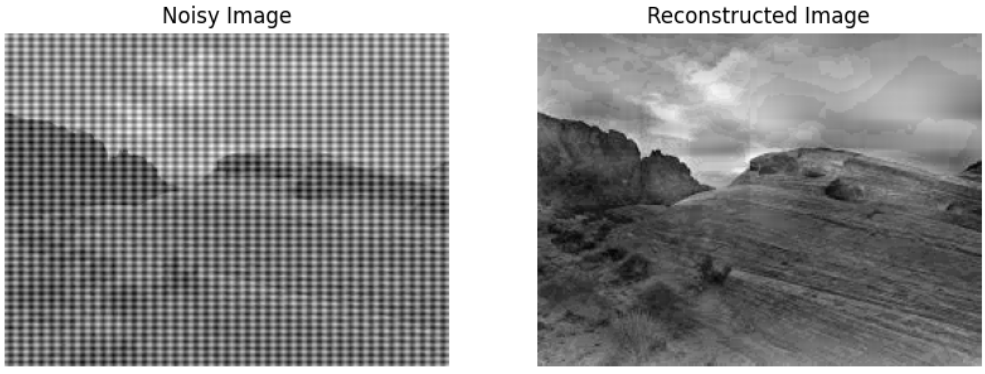

Figure 5: Comparison between noisy (left) and reconstructed (right) images

2.8 Step 8: Compare Original Image with Reconstructed Image

The final comparison between the original and reconstructed images.

```
plt.figure(figsize=(10, 4))
plt.subplot(121), plt.imshow(original_image, cmap="gray"), plt.title("Original Image"), plt.axis('off')
plt.subplot(122), plt.imshow(image_reconstructed, cmap="gray"), plt.title("Reconstructed Image"), plt.axis('off')
plt.show()
```

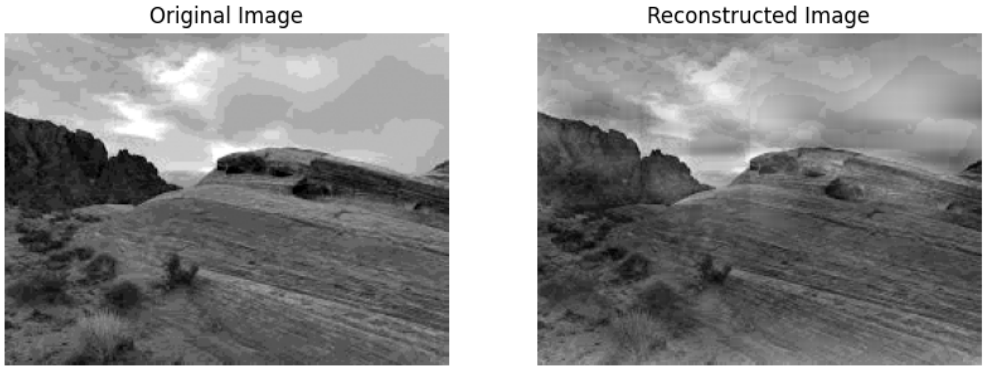

Figure 6: Final comparison between original (left) and reconstructed (right) images

2.9 Step 9: Quality Assurance with PSNR

The Peak Signal-to-Noise Ratio (PSNR) is calculated to quantify the improvement.

```
def psnr(original, noisy):
    mse = np.mean((original - noisy) ** 2)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    psnr_value = 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
    return psnr_value

original_float = original_image.astype(np.float32)
noisy_float = noisy_image.astype(np.float32)
reconstructed_float = image_reconstructed.astype(np.float32)

psnr_noisy = psnr(original_float, noisy_float)
psnr_reconstructed = psnr(original_float, reconstructed_float)

print(f"PSNR (Original vs Noisy): {psnr_noisy:.2f} dB")
print(f"PSNR (Original vs Reconstructed): {psnr_reconstructed:.2f} dB")
```

The output shows:

```
PSNR (Original vs Noisy): 8.12 dB
PSNR (Original vs Reconstructed): 19.47 dB
```

3 Technical Explanations

3.1 PSNR Calculation

The Peak Signal-to-Noise Ratio (PSNR) measures the quality of reconstruction. It is defined as:

$$\text{PSNR} = 20 \cdot \log_{10} \left(\frac{\text{MAX}_I}{\sqrt{\text{MSE}}} \right)$$

where:

- MAX_I is the maximum possible pixel value (255 for 8-bit images)
- MSE is the mean squared error between original and processed images:

$$\text{MSE} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i,j) - K(i,j)]^2$$

Higher PSNR values indicate better quality reconstruction.

3.2 Noise Types

- **Additive Noise**: Noise that is added to the image signal ($I_{noisy} = I_{original} + noise$)
- **Multiplicative Noise**: Noise that is multiplied with the image signal ($I_{noisy} = I_{original} \times noise$)

In this case, we used **additive noise** (specifically periodic additive noise) as evidenced by the operation $noisy\_image = original\_image + noise$.

4 Conclusion

The frequency domain filtering successfully removed the periodic noise, improving the PSNR from 8.12 dB to 19.47 dB. The cross-shaped filter effectively targeted the noise patterns visible in the frequency spectrum while preserving most of the image's important features.

`https://github.com/AsadiAhmad/Filtering-in-Frequency-Domain`