INT3404E 20 - Image Processing Homework 2

Lê Thị Hải Anh - 22028162

April 2024

1 Tổng quan báo cáo

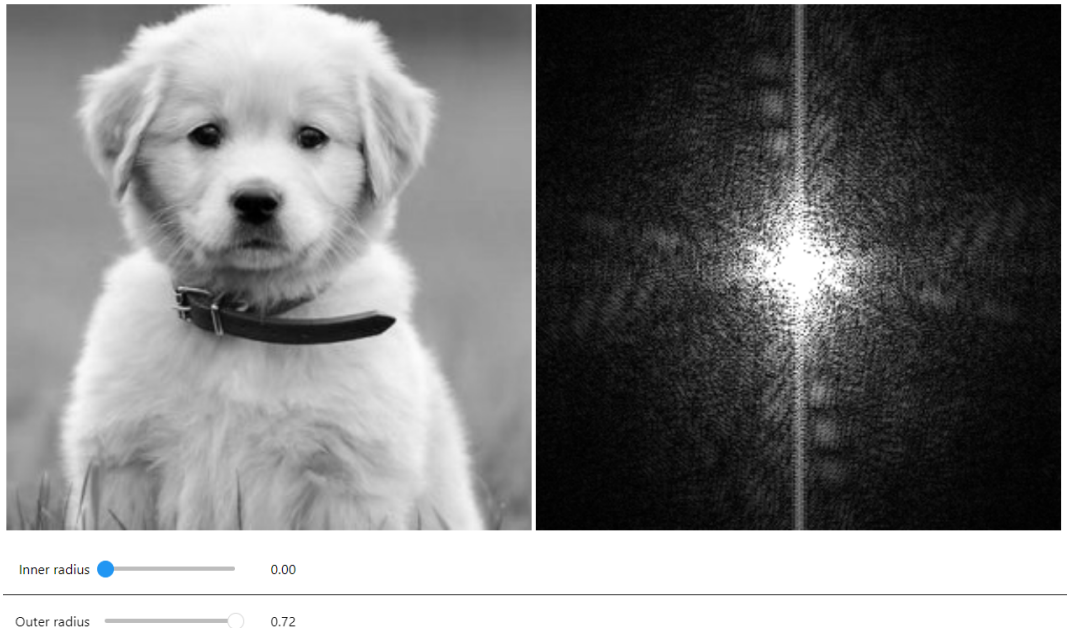
1.1 Image Filtering

Báo cáo kết quả xử lý nhiễu ảnh thông qua bộ lọc trung bình và bộ lọc trung vị

Hình 1: Ảnh nhiễu muối tiêu cần lọc

1.2 Fourier Transform

Trình bày các biến đổi Fourier 1 chiều và 2 chiều và báo cáo kết quả của một số ứng dụng của xử lý ảnh trên miền tần số, cụ thể, phép lọc tần số (lọc thông thấp, lọc thông cao) và phép lai ảnh (Hybrid image)



Hình 2: Ảnh trên miền tần số

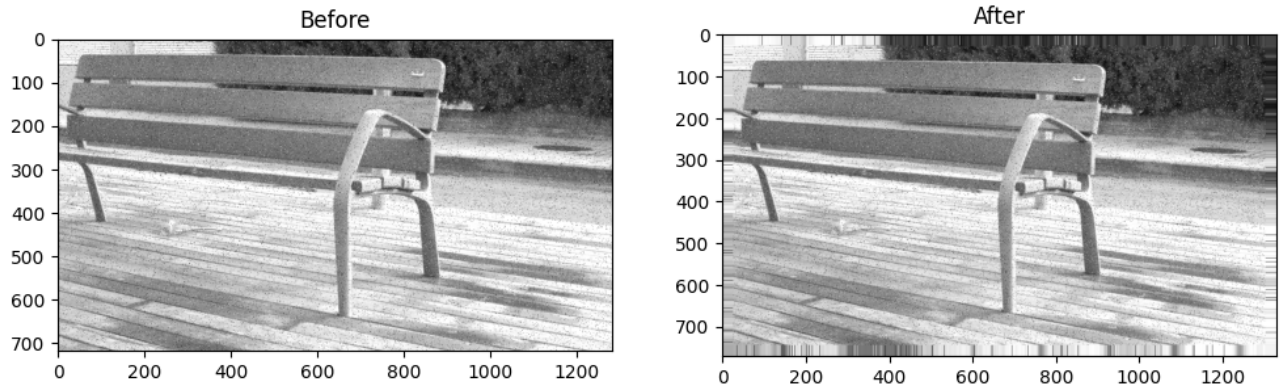
2 Image Filtering

Trong các hàm xử lý chính sẽ được trình bày, thư viện Numpy sẽ được sử dụng để tăng hiệu quả xử lý.

2.1 Hàm padding

Mục đích:
Bổ sung ảnh bằng các pixel xung quanh theo phương pháp *nearest neightbor* để phục vụ cho việc áp dụng cửa số lọc không bị lỗi khi *anchor* nằm ở biên ảnh.
Input:
img: Ảnh gốc cần padding.
$filter\_size$: Kích thước của cửa sổ lọc được sử dụng (mặc định là 3).
Output:
padded_img: Ảnh sau khi padding.

```python
def padding_img(img, filter_size=3):
    #set up for padding
    height, width = img.shape
    pad_size = filter_size // 2
    padded_img = np.zeros((height + 2 * pad_size, width + 2 * pad_size), dtype=img.dtype)
    #copy the original image to the center of padded image
    padded_img[pad_size:pad_size + height, pad_size:pad_size + width] = img
    #replicate padding at non-corner borders
    padded_img[:pad_size, pad_size:-pad_size] = img[0, :]        #top rows
    padded_img[-pad_size:, pad_size:-pad_size] = img[-1, :]      #bottom rows
    padded_img[pad_size:-pad_size,
               :pad_size] = img[:, 0].reshape(-1, 1)             #left columns
    padded_img[pad_size:-pad_size,
               -pad_size:] = img[:, -1].reshape(-1, 1)           #right columns
    #replicate padding at corners
    padded_img[:pad_size, :pad_size] = img[0, 0]                 #top-left
    padded_img[:pad_size, -pad_size:] = img[0, -1]               #top-right
    padded_img[-pad_size:, :pad_size] = img[-1, 0]               #bottom-left
    padded_img[-pad_size:, -pad_size:] = img[-1, -1]             #bottom-right
    return padded_img
```



Hình 3: Kết quả padding với $filter\_size = 50$

2.2 Hàm lọc trung bình

Mục đích:

Áp dụng bộ lọc trung bình cho ảnh để giảm nhiễu (làm mịn) bằng cách thay thế mỗi pixel bằng giá trị trung bình của các pixel lân cận trong cửa sổ bộ lọc (*kernel*).

Input:

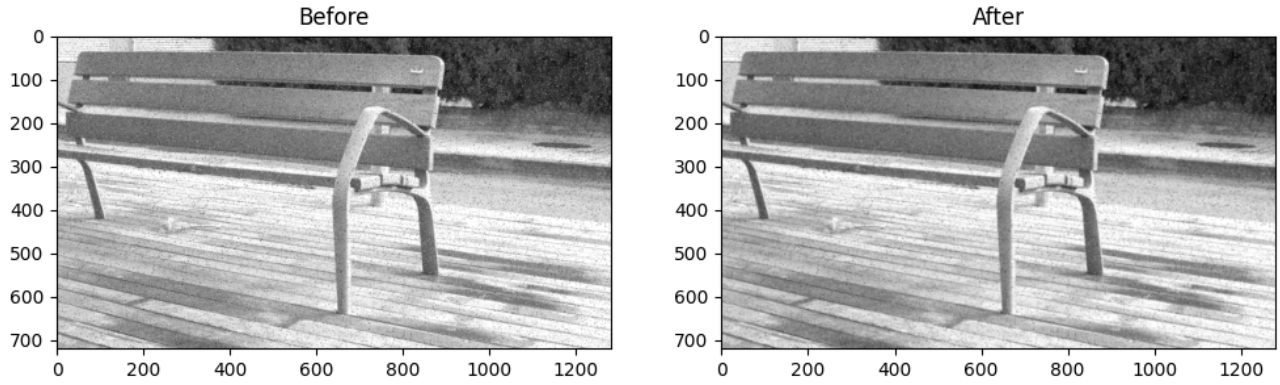
img: Ảnh cần làm mịn.

$filter\_size$: Kích thước cửa sổ bộ lọc đang được sử dụng (mặc định là 3).

Output:

smoothed_img: Ảnh đã được làm mịn với độ nhiễu giảm bằng bộ lọc trung bình.

```python
def mean_filter(img, filter_size=3):
    #set up for smoothing
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img, dtype=img.dtype)
    height, width = img.shape
    #smoothing
    for i in range(height):
        for j in range(width):
            smoothed_img[i, j] = np.mean(padded_img[i:i + filter_size, j:j + filter_size])
    return smoothed_img
```



Hình 4: Kết quả của hàm lọc trung bình với $filter\_size = 3$

2.3 Hàm lọc trung vị

Mục đích:

Áp dụng bộ lọc trung bình cho ảnh để giảm nhiễu bằng cách thay thế mỗi pixel bằng giá trị trung vị của các pixel lân cận trong cửa sổ bộ lọc (kernel).

Input:

img: Ảnh cần làm mịn.

$filter\_size$: Kích thước của cửa sổ lọc đang được sử dụng (mặc định là 3).

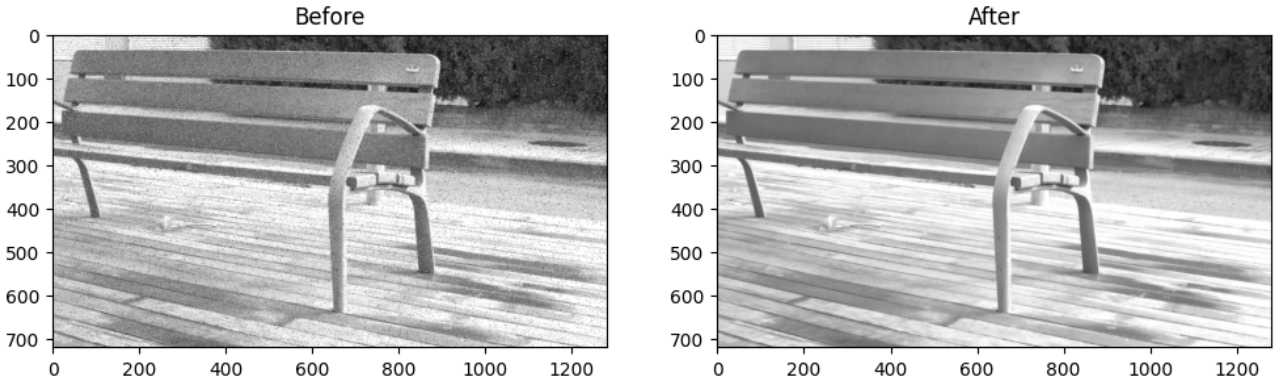
Output:

smoothed_img: Ảnh đã được làm mịn với độ nhiễu giảm bằng bộ lọc trung vị.

```python
def median_filter(img, filter_size=3):
    #set up for smoothing
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img, dtype=img.dtype)
    height, width = img.shape
```

```
    #smoothing
    for i in range(height):
        for j in range(width):
            smoothed_img[i, j] = np.median(padded_img[i:i + filter_size,
                                                      j:j + filter_size])

    return smoothed_img
```



Hình 5: Kết quả của hàm lọc trung vị với $filter\_size = 3$

2.4 Hàm đánh giá chất lượng ảnh

Mục đích:
Tính giá trị PSNR giữa hai ảnh. PSNR là một số liệu phổ biến được sử dụng để đánh giá chất lượng của ảnh được tái tạo hoặc khử nhiễu so với ảnh gốc. PSNR cao hơn cho thấy chất lượng tốt hơn (ít nhiễu hơn). Giá trị PSNR được tính bởi công thức:

$$PSNR = 10 \log_{10} \left(\frac{MAX^2}{MSE} \right)$$

Input::
gt_img: Ảnh gốc.
smooth_img: Ảnh đã được làm mịn (khử nhiễu).
Output:
psnr_score Giá trị PSNR của ảnh được làm mịn.

```
def psnr(gt_img, smooth_img):
    mse = np.mean((gt_img - smooth_img) ** 2)
    if mse == 0:
        return float('inf')
    max_pixel = 255.0
    psnr_score = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr_score
```

Kết quả khi tính toán giá trị PNSR cho bộ lọc:

1. Bộ lọc trung bình: 31.60889963499979
2. Bộ lọc trung vị: 37.11957830085524

Như vậy, ta có thể kết luận ràng với ảnh nhiễu muối tiêu, bộ lọc trung vị hiệu quả hơn.

3 Fourier Transform

3.1 Các hàm biến đổi cơ bản

3.1.1 1D Fourier Transform

Mục đích:
Thực hiện Chuyển đổi Fourier rời rạc (DFT) cho tín hiệu một chiều (1D) theo công thức Fourier Transform 1 chiều:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi j \frac{kn}{N}}$$

Input:
data: Mảng NumPy 1D biểu thị tín hiệu muốn biến đổi. Mảng này có dạng (N,), trong đó N là độ dài của tín hiệu.
Output:
DFT: Mảng NumPy 1D chứa biểu diễn miền tần số của tín hiệu sau khi áp dụng DFT. Mảng này cũng có dạng (N,).

```
def DFT_slow(data):
  N = len(data)
  dft = np.zeros(N, dtype=complex)
  for s in range(N):
      sum = 0.0
      for n in range(N):
          e = np.exp(-2j * np.pi * s * n / N)
          sum += e * data[n]
      dft[s] = sum

  return dft
```

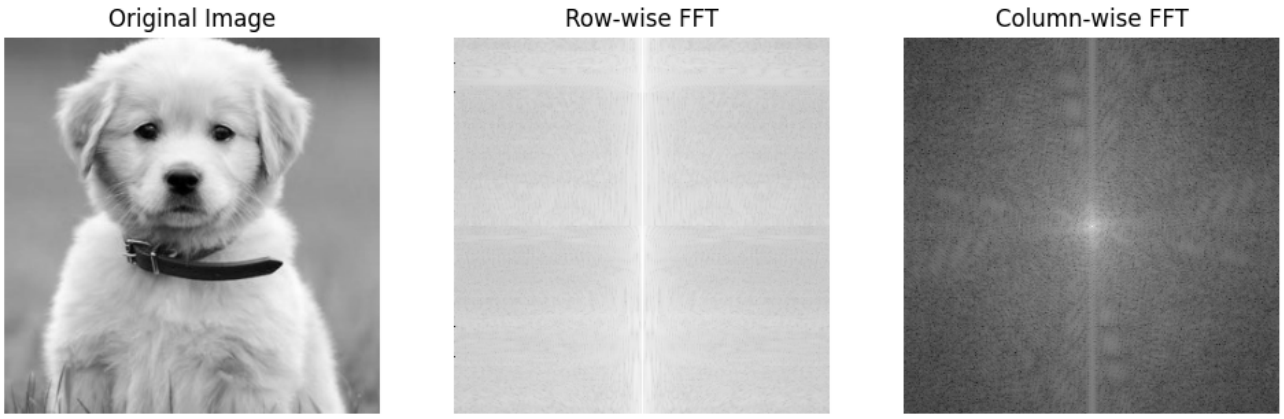
3.1.2 2D Fourier Transform

Mục đích:
Biến đổi một tín hiệu hai chiều (2D), chẳng hạn như hình ảnh, sang miền tần số của nó. Ta áp dụng Fourier Transform 1 chiều từng dòng và cột dựa trên công thức Fourier Transform 2 chiều:

$$F(u,v) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) e^{-j2\pi(um/M + vn/N)}$$

Input:
gray_img: Hình ảnh thang xám được biểu diễn dưới dạng mảng NumPy 2D với kích thước (H, W), trong đó H là chiều cao hình ảnh và W là chiều rộng hình ảnh.
Output:
row_fft: Mảng NumPy 2D với kích thước (H, W) chứa FFT (Fast Fourier Transform) theo hàng của hình ảnh đầu vào.
row_col_fft: Mảng NumPy 2D với kích thước (H, W) chứa FFT theo cột của hình ảnh đầu vào.

```
def DFT_2D(gray_img):
  h, w = gray_img.shape
  row_fft = np.zeros((h, w), dtype=complex)
  row_col_fft_tp = np.zeros((w, h), dtype=complex)
  #apply for each row
  for x in range(h):
      row_fft[x, :] = np.fft.fft(gray_img[x, :])
  #transpose row_fft
  row_fft_tp = np.transpose(row_fft)
```



Hình 6: Kết quả biến đổi Fourier theo 2 chiều

```
    #apply for each column of row_fft
    for x in range(w):
        row_col_fft_tp[x, :] = np.fft.fft(row_fft_tp[x, :])
    row_col_fft = np.transpose(row_col_fft_tp)
    return row_fft, row_col_fft
```

Giải thích thuật toán

1. Khởi tạo mảng đầu ra: Hai mảng NumPy rỗng row_fft và $row\_col\_fft\_tp$ được tạo với kích thước (H, W). Mảng row_fft sẽ lưu trữ kết quả FFT hàng, và $row\_col\_fft\_tp$ sẽ lưu trữ kết quả FFT cột. Cả hai mảng đều được đặt thành kiểu dữ liệu complex để chứa các hệ số DFT là số phức.

2. Tính toán FFT hàng: Một vòng lặp lồng nhau duyệt qua từng hàng x của hình ảnh đầu vào. Đối với mỗi hàng x, hàm $np.fft.fft$ được áp dụng cho hàng của ảnh $gray_img$. Kết quả FFT hàng được lưu trữ trong hàng tương ứng x của mảng row_fft.

3. Chuyển vị FFT hàng: Mảng row_fft được chuyển vị sử dụng $np.transpose()$ để chuyển đổi nó từ thứ tự hàng chính sang thứ tự cột chính. Việc này cho phép tính toán FFT cột hiệu quả trong bước FFT theo cột tiếp theo.

4. Tính toán FFT cột: Sau chuyển vị, mỗi cột của row_fft tương ứng với hàng của row_fft_tp. Như vậy, áp dụng FFT theo hàng cho row_fft_tp ta nhận được FFT theo cột của row_fft. Kết quả FFT cột được lưu trữ trong hàng tương ứng của mảng $row\_col\_fft\_tp$.

5. Chuyển vị FFT cột: Mảng $row\_col\_fft\_tp$ được chuyển vị sử dụng $np.transpose()$ để chuyển đổi nó trở lại đúng thứ tự cột.

6. Trả về kết quả FFT: Hàm trả về hai mảng: row_fft: Chứa FFT hàng của hình ảnh đầu vào. row_col_fft: Chứa FFT cột của hình ảnh đầu vào.

3.2 Xử lý ảnh trên miền tần số

Xử lý ảnh trên miền tần số là một trong những ứng dụng phổ biến nhất của Fourier Transform. Nổi bật là phép lọc thông thấp (tức là lọc tần số thấp) và lọc thông cao (tức là lọc tần số cao)

3.2.1 Hàm lọc tần số - Frequency Removal Procedure

Mục đích:
Lọc các thành phần tần số của một ảnh dựa trên một mask đầu vào.

Input:

orig_img: Ảnh đầu vào dạng ma trận NumPy.

mask: Ma trận NumPy có cùng kích thước với *orig_img* được dùng để xác định giữ lại hoặc loại bỏ các thành phần tần số. Giá trị 1 trong mask tương ứng với giữ lại tần số tại vị trí đó, 0 tương ứng với loại bỏ.

Output:

f_img: Ảnh miền tần số sau khi lọc.

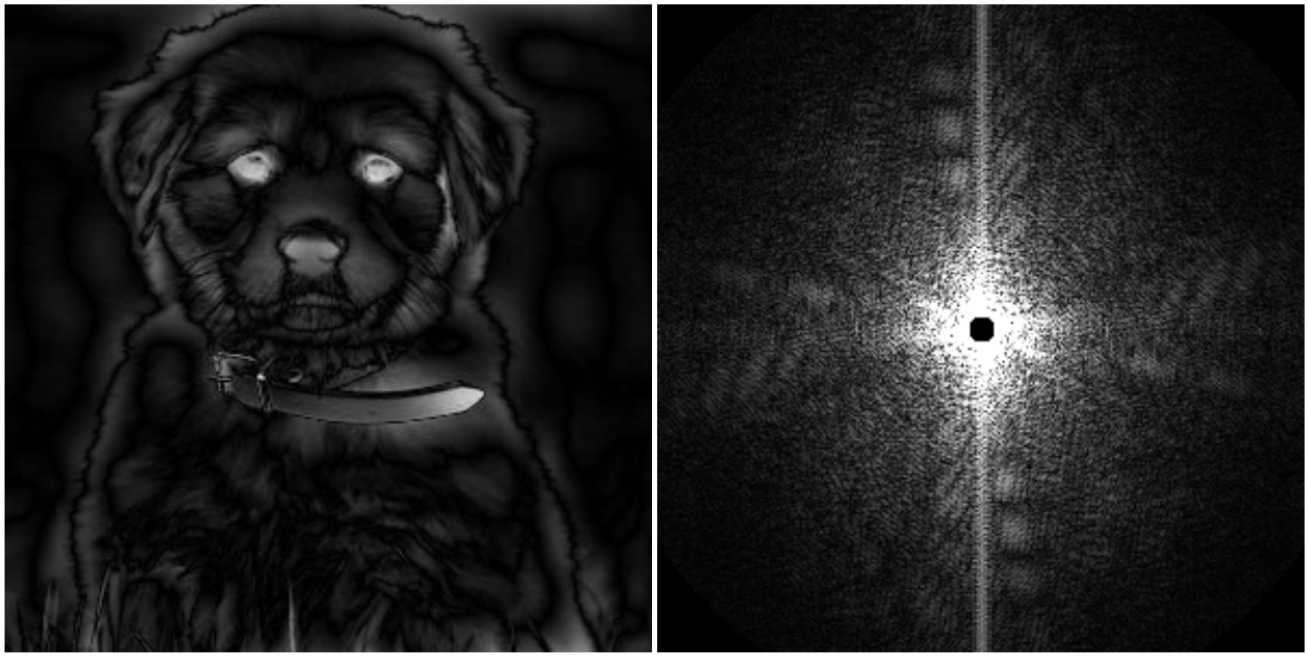
img: Ảnh kết quả sau khi lọc miền tần số và chuyển về miền ảnh.

```
def filter_frequency(orig_img, mask):
  # Transform using fft2
  img_fft = np.fft.fft2(orig_img)
  # Shift frequency coefs to center using fftshift
  img_ffshift = np.fft.fftshift(img_fft)
  # Filter in frequency domain using the given mask
  f_img = img_ffshift * mask
  # Shift frequency coefs back using ifftshift
  f_img_shift = np.fft.ifftshift(f_img)
  # Invert transform using ifft2
  img = np.fft.ifft2(f_img_shift)
  # Ensure real values (absolute value for complex numbers)
  img = np.abs(img)
  f_img = np.abs(f_img)

  return f_img, img
```

Các bước thực hiện:

1. Chuyển đổi ảnh sang miền tần số bằng np.fft.fft2.
2. Dịch chuyển phổ tần số vào trung tâm bằng np.fft.fftshift.
3. Áp dụng mask lên ảnh miền tần số để lọc các thành phần theo mask.
4. Dịch chuyển phổ tần số về vị trí cũ bằng np.fft.ifftshift.
5. Chuyển đổi ảnh từ miền tần số về miền ảnh bằng np.fft.ifft2.
6. Lấy giá trị tuyệt đối của ảnh kết quả để đảm bảo giá trị thực.



Hình 7: Kết quả lọc tần số

3.2.2 Hàm tạo ảnh lai

Mục đích:
Hàm này tạo ra một ảnh lai (hybrid image) là ảnh kết hợp của việc sử dụng lọc thông thấp và lọc thông cao cho 2 ảnh đầu vào riêng biệt.
Input:
*img*1: Ảnh đầu vào 1 dạng ma trận NumPy.
*img*2: Ảnh đầu vào 2 dạng ma trận NumPy.
r: Bán kính của vùng tần số của ảnh 1 được giữ lại.
Output:
Ảnh hybrid được tạo từ ảnh 1 và ảnh 2.

```
def create_hybrid_img(img1, img2, r):
  h, w = img1.shape

  img1_fft = np.fft.fft2(img1)
  img2_fft = np.fft.fft2(img2)

  img1_ffshift = np.fft.fftshift(img1_fft)
  img2_ffshift = np.fft.fftshift(img2_fft)

  #create mask
  x_grid, y_grid = np.meshgrid(np.arange(h), np.arange(w))
  distance = np.sqrt((w // 2 - x_grid) ** 2 + (h // 2 - y_grid) ** 2)

  mask = distance <= r
  f_img1 = img1_ffshift * mask

```

```
    mask = distance > r
    f_img2 = img2_ffshift * mask

    f_img1_shift = np.fft.ifftshift(f_img1)
    f_img2_shift = np.fft.ifftshift(f_img2)

    img1 = np.fft.ifft2(f_img1_shift)
    img2 = np.fft.ifft2(f_img2_shift)

    img = img1 + img2
    img = np.abs(img)
    return img
```

Hình 8: Kết quả hybrid image

Các bước thực hiện:

1. Chuyển đổi cả hai ảnh sang miền tần số bằng $np.fft.fft2$.
2. Dịch chuyển phổ tần số của cả hai ảnh vào trung tâm bằng $np.fft.fftshift$.
3. Tạo mask:
   (a) Tạo ma trận lưới tọa độ (x, y) ứng với ảnh.
   (b) Tính toán khoảng cách từ mỗi điểm (x, y) đến tâm của ảnh.
4. Đặt 1 vào các phần tử của mask tại những điểm có khoảng cách nhỏ hơn hoặc bằng r. Ngược lại đặt 0. Lọc miền tần số ảnh 1 theo mask để giữ lại vùng tần số trong bán kính r.
5. Đặt 1 vào các phần tử của mask tại những điểm có khoảng cách lớn hơn r. Ngược lại đặt 0. Lọc miền tần số ảnh 2 theo mask để giữ lại vùng tần số ngoài bán kính r.
6. Dịch chuyển phổ tần số của ảnh 1 và ảnh 2 về vị trí cũ bằng $np.fft.ifftshift$.
7. Chuyển đổi ảnh 1 và ảnh 2 sau lọc miền tần số về ảnh miền không gian bằng $np.fft.ifft2$.
8. Cộng ảnh kết quả từ ảnh 1 và ảnh 2. Lấy giá trị tuyệt đối của ảnh kết quả.