INT3404E 20 - Image Processing: Homework 2

Lê Xuân Hùng - 22028172

1 Các bộ lọc ảnh

1.1 Hàm *padding_img*

padding_img(*img*, *filter_size* = 3): Hàm này thêm viền sao chép vào ảnh đầu vào *img*, sao cho kích thước cuối cùng của ảnh được thêm viền giống với kích thước ảnh gốc. Kích thước của viền được xác định bởi tham số *filter_size*. Kết quả chạy như hình 1

```python
def padding_img(img, filter_size=3):
    """
    The surrogate function for the filter functions. The goal of the function:
    replicate padding the image such that when applying the kernel with the
    size of filter_size, the padded image will be the same size as the
    original image.
    WARNING: Do not use the exterior functions from available libraries such as
    OpenCV, scikit-image, etc. Just do from scratch using function from the
    numpy library or functions in pure Python.

    Inputs:
    img: cv2 image: original image
    filter_size: int: size of square filter

    Return:
    padded_img: cv2 image: the padding image
    """
    h, w = img.shape
    pad_h = (filter_size - 1) // 2
    pad_w = (filter_size - 1) // 2
    padded_img = np.pad(img, ((pad_h, pad_h), (pad_w, pad_w)), mode='edge')
    return padded_img
```

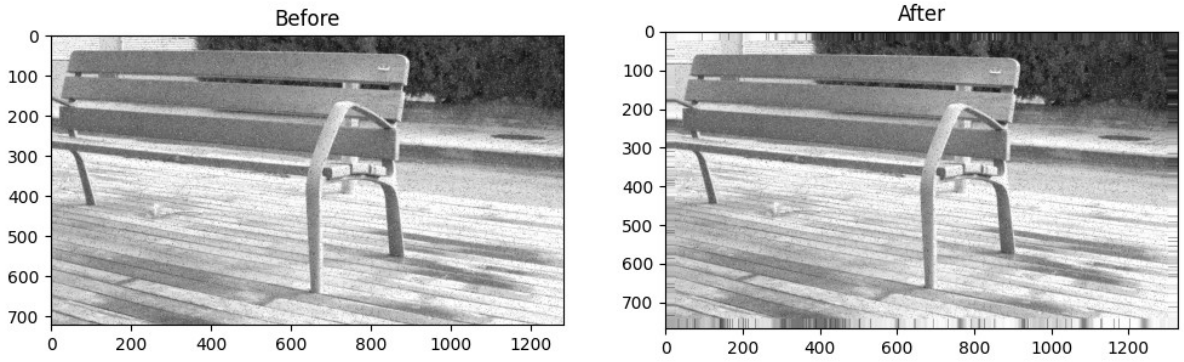

Figure 1: Kết quả hàm *padding_img* khi *filter_size* = 50

1.2 Hàm *mean_filter*

mean_filter(*img*, *filter_size* = 3): Hàm này áp dụng bộ lọc trung bình vào ảnh đầu vào *img* sử dụng *filter_size* được chỉ định. Trước tiên, thêm viền vào ảnh bằng cách sử dụng hàm *padding_img*, sau đó tính giá trị trung bình của mỗi cửa sổ *filter_sizexfilter_size* trong ảnh được thêm viền và gán giá trị đó vào điểm tương ứng trong ảnh đầu ra. Kết quả chạy như hình 2

```python
def mean_filter(img, filter_size=3):
    """
    Smoothing image with mean square filter with the size of filter_size.
    Use replicate padding for the image.
    WARNING: Do not use the exterior functions from available libraries such as
    OpenCV, scikit-image, etc. Just do from scratch using function from the
    numpy library or functions in pure Python.

    Inputs:
    img: cv2 image: original image
    filter_size: int: size of square filter,

    Return:
    smoothed_img: cv2 image: the smoothed image with mean filter.
    """
    padded_img = padding_img(img, filter_size)
    h, w = img.shape
    smooth_img = np.zeros_like(img)

    for i in range(h):
        for j in range(w):
            start_i = i - filter_size // 2
            start_j = j - filter_size // 2
            end_i = start_i + filter_size
            end_j = start_j + filter_size
            smooth_img[i, j] = np.mean(padded_img[start_i:end_i, start_j:end_j])

    return smooth_img
```

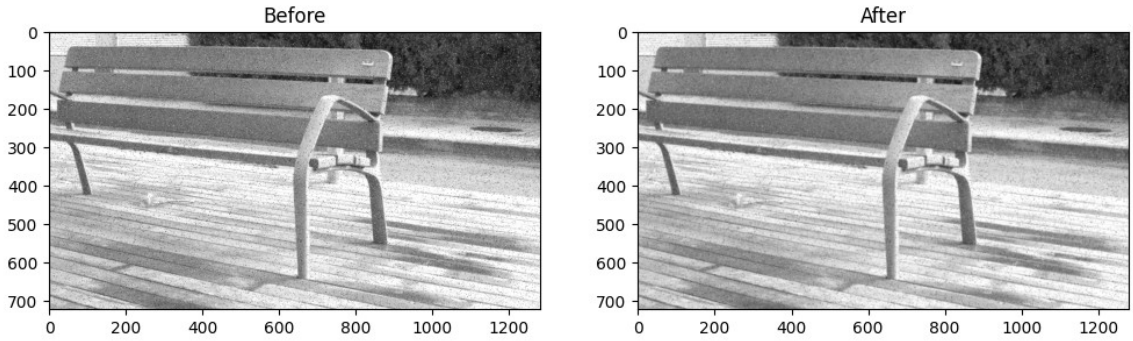

Figure 2: Kết quả hàm *mean_filter*

1.3 Hàm *median_filter*

median_filter(*img*, *filter_size* = 3): Hàm này áp dụng bộ lọc trung vị vào ảnh đầu vào *img* sử dụng *filter_size* được chỉ định. Trước tiên, thêm viền vào ảnh bằng cách sử dụng hàm *padding_img*, sau đó tính giá trị trung vị của mỗi cửa sổ *filter_sizexfilter_size* trong ảnh được thêm viền và gán giá trị đó vào điểm tương ứng trong ảnh đầu ra. Kết quả chạy như hình 3

```python
def median_filter(img, filter_size=3):
    """
    Smoothing image with median square filter with the size of filter_size.
    Use replicate padding for the image.
    WARNING: Do not use the exterior functions from available libraries such as
    OpenCV, scikit-image, etc. Just do from scratch using function from the
    numpy library or functions in pure Python.

    Inputs:
    img: cv2 image: original image
    filter_size: int: size of square filter

    Return:
    smoothed_img: cv2 image: the smoothed image with median filter.
    """
    padded_img = padding_img(img, filter_size)
    h, w = img.shape
    smooth_img = np.zeros_like(img)

    for i in range(h):
        for j in range(w):
            start_i = i - filter_size // 2
            start_j = j - filter_size // 2
            end_i = start_i + filter_size
            end_j = start_j + filter_size
            smooth_img[i, j] = np.median(padded_img[start_i:end_i, start_j:end_j])

    return smooth_img
```

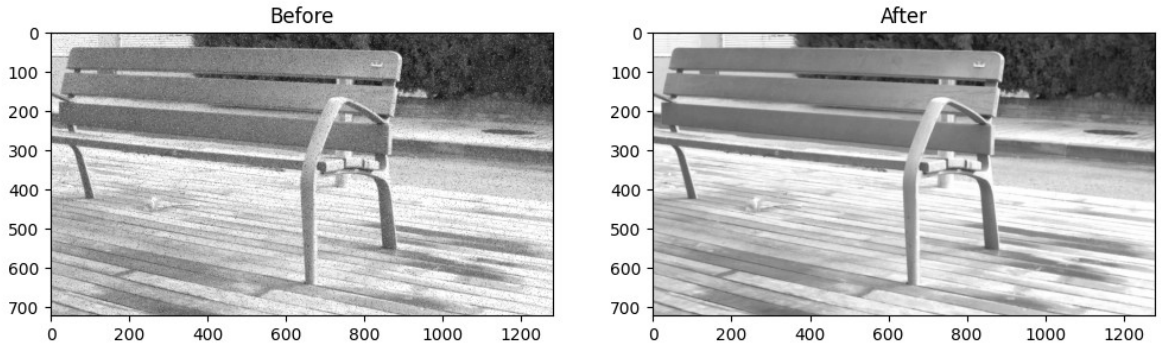

Figure 3: Kết quả hàm *median_filter*

1.4 Hàm tính độ đo *psnr*

Độ đo $PSNR$ được tính như sau:

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

$psnr(gt\_img, smooth\_img)$: Hàm này tính toán tỷ số tín hiệu-nhiễu cực đại ($PSNR$) giữa ảnh chuẩn gt_img và ảnh được làm mượt $smooth\_img$. Trước tiên, nó tính toán lỗi bình phương trung bình (MSE) giữa hai ảnh, sau đó sử dụng MSE để tính toán điểm $PSNR$.

```python
def psnr(gt_img, smooth_img):
    """
    Calculate the PSNR metric

    Inputs:
    gt_img: cv2 image: groundtruth image
    smooth_img: cv2 image: smoothed image

    Outputs:
    psnr_score: PSNR score
    """
    mse = np.mean((gt_img - smooth_img)**2)
    if mse == 0:
        return float('inf')
    max_pixel = 255.0
    psnr_score = 20 * np.log10(max_pixel / np.sqrt(mse))
    return psnr_score
```

Kết quả thu được sau khi tính độ đo của 2 hàm $mean\_filter$ và $median\_filter$ như sau:

- PSNR score of mean filter: 30.455473526910275
- PSNR score of median filter: 32.366668307426586

=> Dựa vào các kết quả PSNR được cung cấp, lọc trung vị ($median\_filter$) có điểm PSNR là 32.37, cao hơn so với lọc trung bình ($mean\_filter$) có điểm PSNR là 30.46.
Vì vậy, nếu mục tiêu là tối ưu hóa PSNR, chúng ta nên chọn lọc trung vị ($median\_filter$) để xử lý các ảnh đã cho.
Lọc trung vị thường tốt hơn trong việc loại bỏ nhiễu và giữ lại chi tiết hơn so với lọc trung bình, do đó nó có thể cho điểm PSNR cao hơn.

2 Fourier Transform

2.1 Biến đổi cơ bản

2.1.1 1D-Fourier Transform

Đây là cách hoạt động của hàm:

1. Đầu tiên, chúng ta lấy độ dài của mảng đầu vào *data* và lưu trữ nó vào biến N.
2. Sau đó, chúng ta tạo một mảng DFT với độ dài là N và kiểu dữ liệu là $np.complex\_$. Đây sẽ là mảng chứa kết quả của DFT.
3. Tiếp theo, chúng ta sử dụng hai vòng lặp lồng nhau để tính toán giá trị của từng phần tử trong mảng DFT:
   - Vòng lặp ngoài lặp qua các chỉ số k từ 0 đến $N-1$. Chỉ số k đại diện cho các tần số khác nhau trong DFT.
   - Vòng lặp trong lặp qua các chỉ số n từ 0 đến $N-1$. Chỉ số n đại diện cho các điểm dữ liệu trong tín hiệu đầu vào *data*.
   - Tại mỗi lần lặp, chúng ta tính toán giá trị $data[n] * np.exp(-1j * 2 * np.pi * n * k/N)$ và cộng nó vào phần tử tương ứng $DFT[k]$.
4. Sau khi hoàn thành hai vòng lặp, chúng ta sẽ có mảng DFT chứa kết quả của DFT trên tín hiệu đầu vào *data*.

```python
def DFT_slow(data):
    """
    Implement the discrete Fourier Transform for a 1D signal

    params:
      data: Nx1: (N, ): 1D numpy array

    returns:
      DFT: Nx1: 1D numpy array
    """
    N = len(data)
    DFT = np.zeros(N, dtype=np.complex_)

    for k in range(N):
        for n in range(N):
            DFT[k] += data[n] * np.exp(-1j * 2 * np.pi * n * k / N)
    return DFT
```

2.1.2 2D-Fourier Transform

Hàm DFT_2D được sử dụng để thực hiện Biến đổi Fourier rời rạc 2 chiều (2D Discrete Fourier Transform) trên một ảnh đầu vào 2 chiều $gray\_img$.
Đây là các bước thực hiện của hàm:

1. Bước 1: Áp dụng DFT 1 chiều (1D DFT) vào từng hàng của ảnh đầu vào $gray\_img$.
   - Chúng ta sử dụng hàm $np.apply\_along\_axis$ để áp dụng hàm DFT_slow (đã được định nghĩa trước đó) vào từng hàng của $gray\_img$.
   - Kết quả của bước này là mảng row_fft, chứa FFT theo chiều hàng của ảnh đầu vào.
2. Bước 2: Áp dụng DFT 1 chiều (1D DFT) vào từng cột của kết quả từ bước 1.

- Chúng ta sử dụng hàm *np.apply_along_axis* một lần nữa để áp dụng hàm DFT_slow vào từng cột của mảng row_fft.
- Kết quả của bước này là mảng row_col_fft, chứa FFT theo chiều cột của ảnh đầu vào.

Sau khi thực hiện hai bước này, chúng ta sẽ có:

- row_fft: mảng chứa FFT theo chiều hàng của ảnh đầu vào $gray\_img$.
- row_col_fft: mảng chứa FFT theo chiều cột của ảnh đầu vào $gray\_img$.

Lưu ý rằng, kết quả trả về từ hàm DFT_2D có kiểu dữ liệu là $complex\_$, như đề cập trong phần mô tả của hàm.
Hàm DFT_2D này được sử dụng để thực hiện biến đổi Fourier 2D trên ảnh đầu vào $gray\_img$, tách biệt thành các thành phần tần số theo chiều hàng và chiều cột. Kết quả chạy như hình 4

```python
def DFT_2D(gray_img):
    """
    Implement the 2D Discrete Fourier Transform
    Note that: dtype of the output should be complex_
    params:
        gray_img: (H, W): 2D numpy array

    returns:
        row_fft: (H, W): 2D numpy array that contains the row-wise FFT of the input image
        row_col_fft: (H, W): 2D numpy array that contains the column-wise FFT of the input image
    """
    # Step 1: Apply 1D DFT to each row of the input image
    row_fft = np.apply_along_axis(DFT_slow, axis=1, arr=gray_img)
    # Step 2: Apply 1D DFT to each column of the result from step 1
    row_col_fft = np.apply_along_axis(DFT_slow, axis=0, arr=row_fft)

    return row_fft, row_col_fft
```

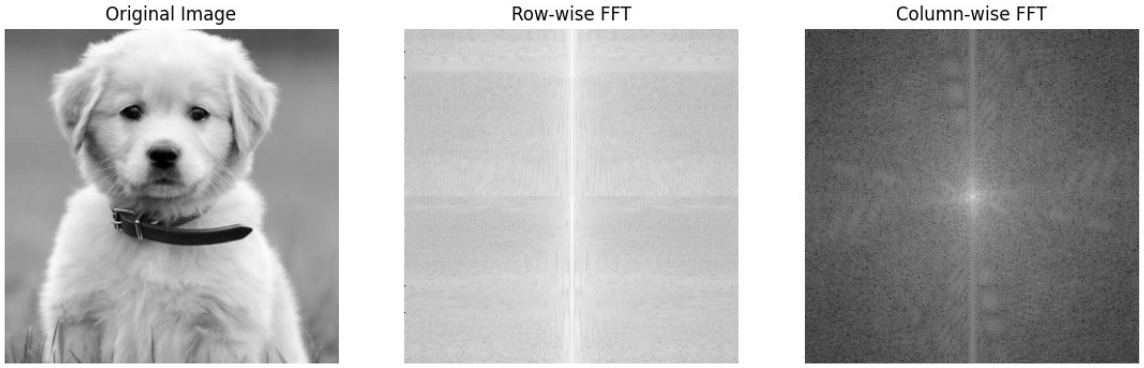

Figure 4: Kết quả khi triển khai 2D-Fourier Transform với ảnh

2.2 Một vài ứng dụng của Fourier Transform

2.2.1 Hàm lọc tần số

Hàm $filter\_frequency$ được sử dụng để lọc tần số của một ảnh đầu vào $orig\_img$ dựa trên một mặt nạ (mask) đã cho.

Đây là các bước thực hiện của hàm:

1. Bước 1: Chuyển ảnh đầu vào $orig\_img$ sang miền tần số bằng cách sử dụng hàm $np.fft.fft2()$. Kết quả của bước này là mảng f_img chứa biểu diễn tần số của ảnh đầu vào.
2. Bước 2: Dịch chuyển các hệ số tần số về vị trí trung tâm của mảng bằng cách sử dụng hàm $np.fft.fftshift()$. Kết quả của bước này là mảng $f\_img\_shifted$, trong đó các hệ số tần số đã được dịch chuyển về vị trí trung tâm.
3. Bước 3: Lọc miền tần số bằng cách nhân $f\_img\_shifted$ với mặt nạ $mask$. Kết quả của bước này là mảng $f\_img\_filtered$, chứa miền tần số đã được lọc.
4. Bước 4: Dịch chuyển các hệ số tần số về vị trí ban đầu bằng cách sử dụng hàm $np.fft.ifftshift()$. Kết quả của bước này là mảng $f\_img\_filtered\_shifted$, chứa các hệ số tần số đã được dịch chuyển về vị trí ban đầu.
5. Bước 5: Chuyển ngược biểu diễn tần số về miền không gian bằng cách sử dụng hàm $np.fft.ifft2()$. Kết quả của bước này là mảng img, chứa ảnh đã được lọc trong miền không gian.

Cuối cùng, hàm trả về cả mảng $f\_img\_filtered$ (chứa biểu diễn tần số đã được lọc) và mảng img (chứa ảnh đã được lọc trong miền không gian).
Mặt nạ $mask$ được sử dụng trong bước 3 để chỉ định các tần số cần giữ lại hoặc loại bỏ. Nó có cùng kích thước với $orig\_img$.
Kết quả chạy như các hình 5, 6, 7 ứng với các trường hợp **Loc đi miền chứa các tần số thấp**, **Loc đi miền chứa các tần số thấp và tần số cao**, **Loc đi miền chứa các tần số cao**

```python
def filter_frequency(orig_img, mask):
    """
    You need to remove frequency based on the given mask.
    Params:
        orig_img: numpy image
        mask: same shape with orig_img indicating which frequency hold or remove
    Output:
        f_img: frequency image after applying mask
        img: image after applying mask
    """
    # Step 1: Transform the image to the frequency domain using fft2
    f_img = np.fft.fft2(orig_img)
    # Step 2: Shift the frequency coefficients to the center using fftshift
    f_img_shifted = np.fft.fftshift(f_img)
    # Step 3: Filter the frequency domain representation using the given mask
    f_img_filtered = f_img_shifted * mask
    # Step 4: Shift the frequency coefficients back to their original positions using ifftshift
    f_img_filtered_shifted = np.fft.ifftshift(f_img_filtered)
    # Step 5: Invert the transform using ifft2 to get the filtered image in the spatial domain
    img = np.abs(np.fft.ifft2(f_img_filtered_shifted))
    f_img_filtered = np.abs(f_img_filtered)
    return f_img_filtered, img
```

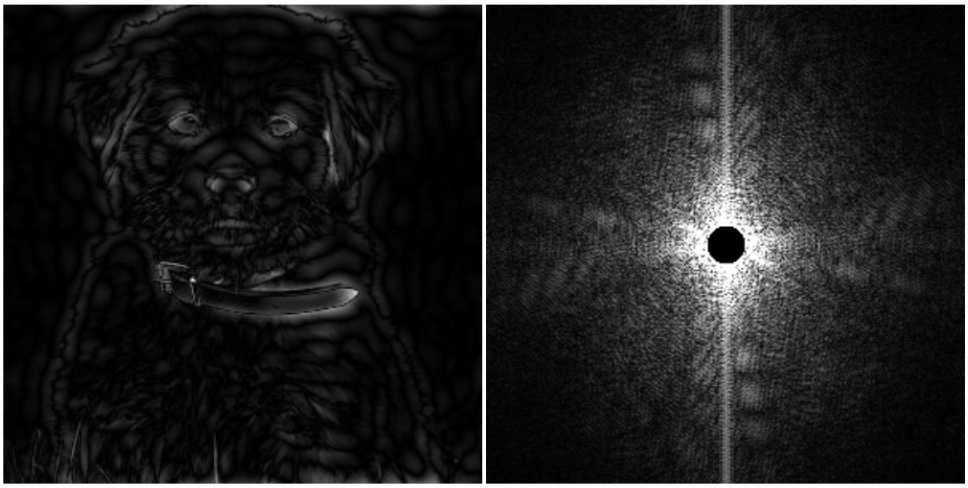

Figure 5: Loc đi miền chứa các tần số thấp

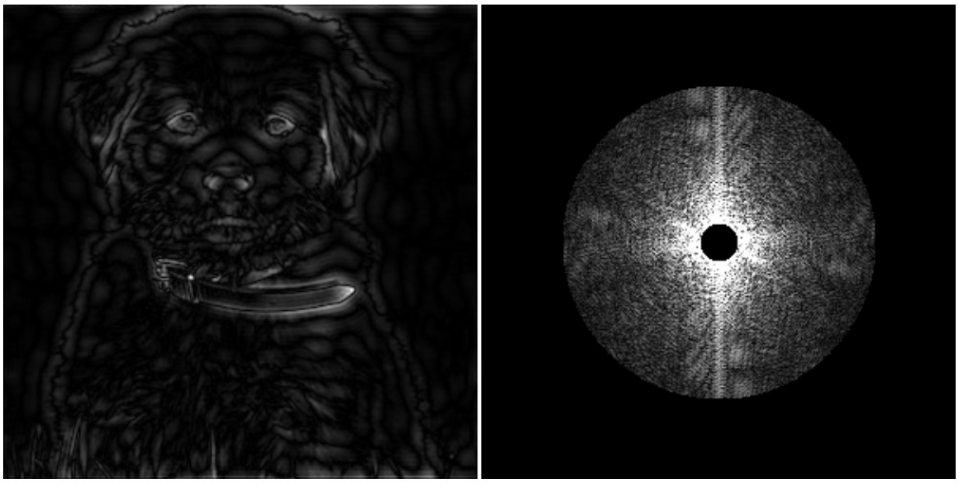

Figure 6: Loc đi miền chứa các tần số thấp và tần số cao

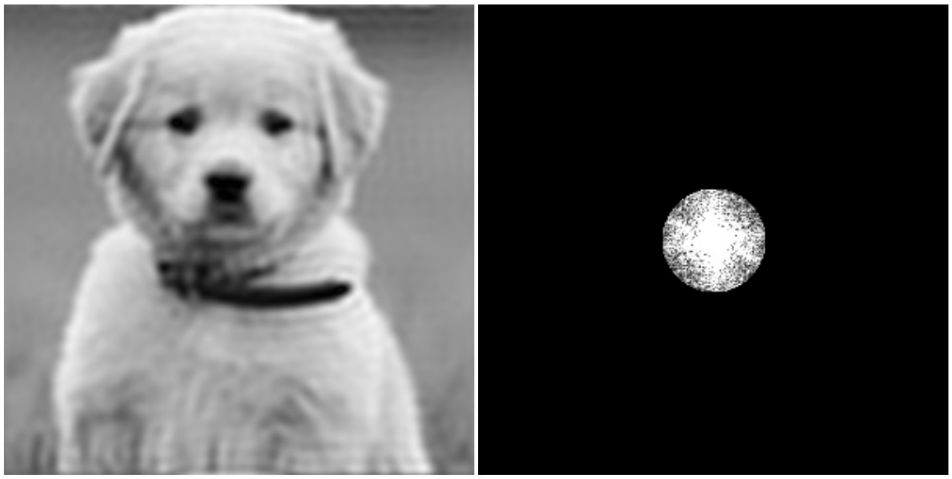

Figure 7: Loc đi miền chứa các tần số cao

2.2.2 Tạo ảnh Hybrid

Hàm $create\_hybrid\_img$ được sử dụng để tạo ra một ảnh hybrid (ảnh lai) từ hai ảnh đầu vào $img1$ và $img2$, dựa trên một tham số bán kính r.
Dây là các bước thực hiện của hàm:

1. Bước 1: Chuyển các ảnh đầu vào $img1$ và $img2$ sang miền tần số bằng cách sử dụng hàm $np.fft.fft2()$. Kết quả của bước này là mảng f_img1 và f_img2, chứa biểu diễn tần số của hai ảnh đầu vào.

2. Bước 2: Dịch chuyển các hệ số tần số về vị trí trung tâm của mảng bằng cách sử dụng hàm $np.fft.fftshift()$. Kết quả của bước này là mảng $f\_img1\_shifted$ và $f\_img2\_shifted$, trong đó các hệ số tần số đã được dịch chuyển về vị trí trung tâm.

3. Bước 3: Tạo ra một mặt nạ $mask$ dựa trên tham số bán kính r. Chúng tôi tính khoảng cách từ mỗi điểm ảnh đến tâm của ảnh, sau đó tạo ra mặt nạ $mask$ với các giá trị là 1 ở trong vòng tròn bán kính r, và 0 ở ngoài vòng tròn.

4. Bước 4: Kết hợp các thành phần tần số của hai ảnh bằng cách sử dụng mặt nạ $mask$. Chúng ta nhân $f\_img1\_shifted$ với $mask$ và $f\_img2\_shifted$ với $1 - mask$, sau đó cộng lại để được $f\_hybrid\_shifted$.

5. Bước 5: Dịch chuyển các hệ số tần số về vị trí ban đầu bằng cách sử dụng hàm $np.fft.ifftshift()$. Kết quả của bước này là mảng f_hybrid, chứa các hệ số tần số đã được dịch chuyển về vị trí ban đầu.

6. Bước 6: Chuyển ngược biểu diễn tần số về miền không gian bằng cách sử dụng hàm $np.fft.ifft2()$. Kết quả của bước này là mảng $hybrid\_img$, chứa ảnh hybrid trong miền không gian.

Cuối cùng, hàm trả về mảng $hybrid\_img$, đây là ảnh hybrid được tạo ra từ hai ảnh đầu vào $img1$ và $img2$, dựa trên tham số bán kính r.
Kết quả chạy như hình 8

```python
def create_hybrid_img(img1, img2, r):
    """
    Create hydrid image
    Params:
        img1: numpy image 1
        img2: numpy image 2
        r: radius that defines the filled circle of frequency of image 1. Refer to the homework title to know more.
    """
    # You need to implement the function
    # Step 1: Transform the images to the frequency domain using fft2
    f_img1 = np.fft.fft2(img1)
    f_img2 = np.fft.fft2(img2)

    # Step 2: Shift the frequency coefficients to the center using fftshift
    f_img1_shifted = np.fft.fftshift(f_img1)
    f_img2_shifted = np.fft.fftshift(f_img2)

    # Step 3: Create a mask based on the given radius
    rows, cols = img1.shape
    center_x, center_y = rows // 2, cols // 2
    y, x = np.ogrid[:rows, :cols]
    dist_from_center = np.sqrt((x - center_x)**2 + (y - center_y)**2)
    mask = dist_from_center <= r
    mask = np.float32(mask)

    # Step 4: Combine the frequency components of the two images using the mask
    f_hybrid_shifted = f_img1_shifted * mask + f_img2_shifted * (1 - mask)

    # Step 5: Shift the frequency coefficients back to their original positions using ifftshift
    f_hybrid = np.fft.ifftshift(f_hybrid_shifted)

    # Step 6: Invert the transform using ifft2 to get the hybrid image in the spatial domain
    hybrid_img = np.real(np.fft.ifft2(f_hybrid))

    return hybrid_img
```

Figure 8: Ảnh Hybrid