INT3404E 20 - Image Processing: Homework 2

Lương Thị Linh - 22028202

1 Tóm tắt

Báo cáo này trình bày chi tiết về việc triển khai các hàm xử lý ảnh cơ bản và thuật toán Biến đổi Fourier (FT) cho Bài tập 2 của môn INT3404E - Xử lý ảnh. Báo cáo bao gồm:

- Triển khai các chức năng đệm ảnh, lọc ảnh (box/mean/median) để loại bỏ nhiễu.
- So sánh hiệu quả của các bộ lọc mean và median.
- Thuật toán Biến đổi Fourier 1D và 2D.
- Triển khai chức năng lọc miền tần số để tạo ảnh lai.

2 Mục tiêu bài tập

- Nâng cao hiểu biết về hoạt động của các bộ lọc ảnh cơ bản.
- Nắm vững kiến thức về thuật toán Biến đổi Fourier (FT).

3 Chi tiết

3.1 Lọc ảnh

Bài tập sẽ giúp bạn hiểu các bộ lọc ảnh cơ bản thông qua việc xử lý bộ lọc hộp (box), trung bình (mean) và trung vị (median).

3.1.1 Triển khai một Replicate padding

Hàm padding_img(img, filter_size=3): nhân rộng phần đệm của hình ảnh sao cho khi áp dụng kernel có kích thước filter_size, hình ảnh được đệm sẽ có cùng kích thước với ảnh gốc. Viền sao chép này được tạo ra bằng cách sao chép giá trị pixel từ các phần tử ở mép của ảnh gốc và thêm vào các phần tử mép của ảnh mới.

Listing 1: Triển khai hàm padding_img

```
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.
    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
    20 pad_w = (filter_size - 1) // 2
    padded_img = np.pad(img, ((pad_h, pad_h), (pad_w, pad_w)), mode='edge')
    return padded_img
```

Kết quả thực hiện với filter_size = 50 như Figure 1

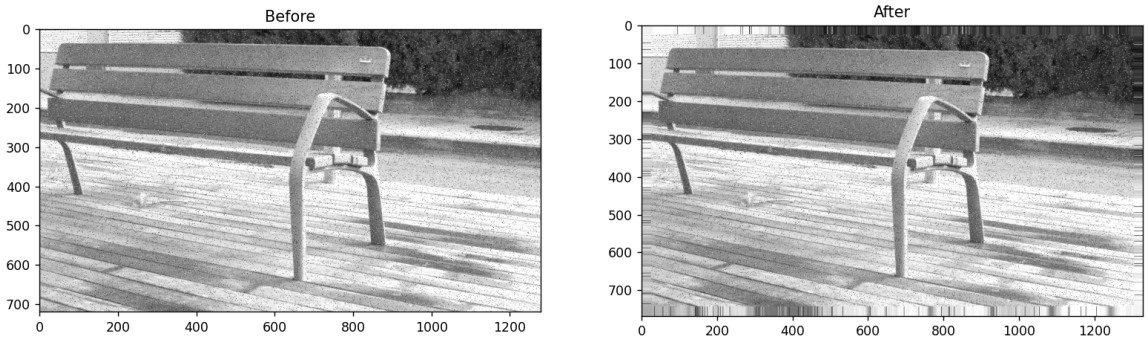

Figure 1: Ảnh kết quả sau khi áp dụng hàm padding_img

3.1.2 Triển khai bộ lọc trung bình (box/mean filters) để loại bỏ nhiễu

Hàm mean_filter thực hiện việc làm mịn hình ảnh bằng cách sử dụng mean square filter có kích thước filter_size.

- **Padding ảnh:** Trước tiên, hàm sẽ thêm viền đệm cho ảnh gốc bằng cách sử dụng phương pháp đệm sao chép. Viền đệm này giúp đảm bảo rằng không gian xung quanh các pixel ở mép của ảnh cũng được xem xét khi thực hiện phép toán mean filter.
- **Áp dụng mean filter:** Sau khi có ảnh đã được đệm, hàm sẽ lặp qua từng pixel của ảnh gốc và tính giá trị trung bình của các pixel trong vùng lân cận xác định bởi kích thước filter.
- **Trả về ảnh đã làm mịn:** Kết quả của quá trình trên là một ảnh mới, trong đó mỗi pixel được thay thế bằng giá trị trung bình của các pixel lân cận.

Listing 2: Triển khai hàm mean_filters

```
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.
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter,
    Return:
        smoothed_img: cv2 image: the smoothed image with mean filter.
    """
    # Padding the image
    padded_img = padding_img(img, filter_size)
    # Initialize smoothed image
    smoothed_img = np.zeros_like(img)
    # Get the height and width of the input image
    h, w = img.shape

    # Loop through each pixel of the input image
    for i in range(h):
        for j in range(w):
            # Compute the mean value of the pixel's neighborhood defined by the filter size
            smoothed_img[i, j] = np.mean(padded_img[i:i+filter_size, j:j+filter_size])
    return smoothed_img
```

Kết quả thực hiện như Figure 2

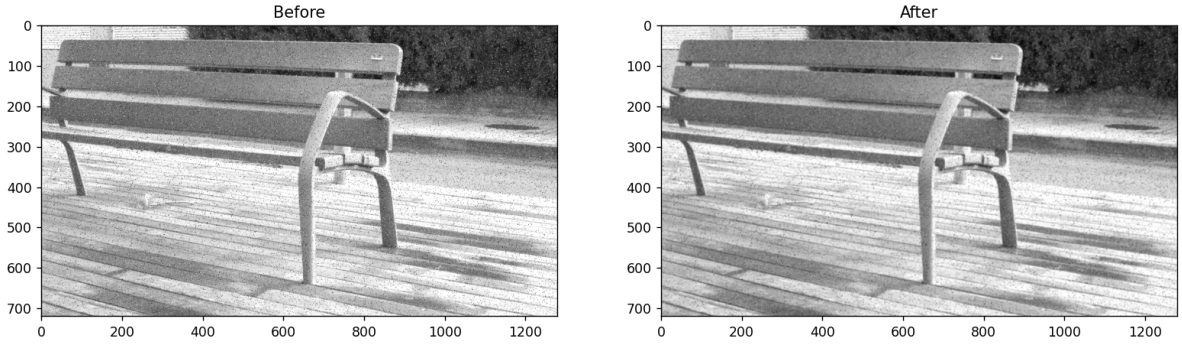

Figure 2: Ảnh kết quả sau khi áp dụng hàm mean_filter

3.1.3 Triển khai bộ lọc trung vị (median filter) để loại bỏ nhiễu

Hàm median_filter thực hiện việc làm mịn hình ảnh bằng cách sử dụng median square filter có kích thước filter_size.

- **Padding ảnh:** Thêm viền đệm cho ảnh gốc sử dụng phương pháp đệm sao chép.
- **Áp dụng median filter:** Lặp qua từng pixel của ảnh gốc và tính giá trị trung vị của các pixel trong vùng lân cận xác định bởi kích thước filter.
- **Trả về ảnh đã làm mịn:** Trả về ảnh mới, trong đó mỗi pixel được thay thế bằng giá trị trung vị của các pixel lân cận.

Listing 3: Triển khai hàm median_filter

```
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.
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter
    Return:
        smoothed_img: cv2 image: the smoothed image with median filter.
    """
    # Padding the image
    padded_img = padding_img(img, filter_size)
    # Initialize smoothed image with the same size as the input image
    smoothed_img = np.zeros_like(img)
    # Get the height and width of the input image
    h, w = img.shape

    # Loop through each pixel of the input image
    for i in range(h):
        for j in range(w):
            # Compute the median value of the pixel's neighborhood defined by the filter size
            smoothed_img[i, j] = np.median(padded_img[i:i+filter_size, j:j+filter_size])
    return smoothed_img
```

Kết quả thực hiện như Figure 3

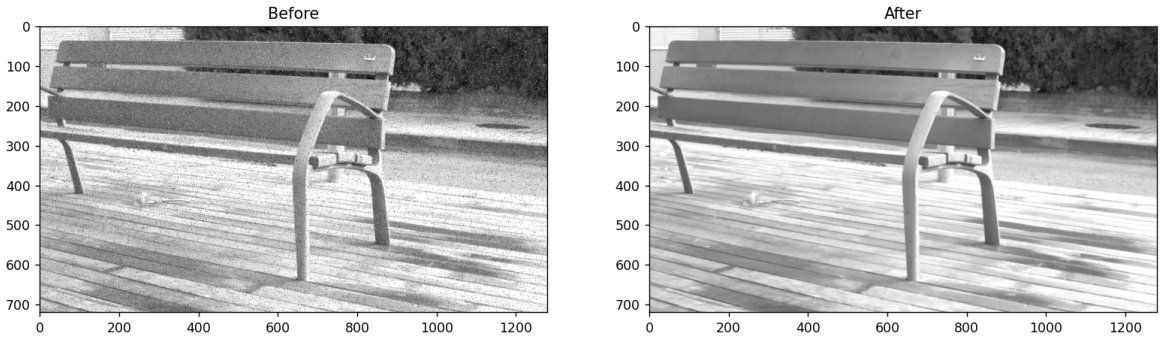

Figure 3: Ảnh kết quả sau khi áp dụng hàm median_filter

3.1.4 Đánh giá chất lượng ảnh (PSNR)

PSNR (Peak Signal-to-Noise Ratio) là một chỉ số để đánh giá chất lượng ảnh sau khi xử lý so với ảnh gốc. Nó được tính toán bằng công thức:

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

Trong đó:

- **MAX** là giá trị pixel lớn nhất (thường là 255 đối với ảnh 8-bit).
- **MSE** là lỗi bình phương trung bình giữa ảnh gốc và ảnh sau khi xử lý.

Giá trị PSNR càng cao, chất lượng ảnh sau khi xử lý càng tốt.

Listing 4: Triển khai hàm tính PSNR

```
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
    """
    # Calculate the Mean Squared Error (MSE)
    mse = np.mean((gt_img - smooth_img) ** 2)
    if mse == 0:
        return np.Infinity # PSNR is infinite, set to a high value
    # Maximum possible pixel value (typically 255 for 8-bit images)
    max_pixel = 255.0
    # Calculate PSNR using the formula
    psnr_score = 10 * np.log10(max_pixel ** 2 / mse)
    return psnr_score
```

3.1.5 Đánh giá và lựa chọn bộ lọc

Sử dụng các hàm đã triển khai để đánh giá PSNR của ảnh sau khi lọc bằng bộ lọc trung bình (mean filter) và trung vị (median filter).

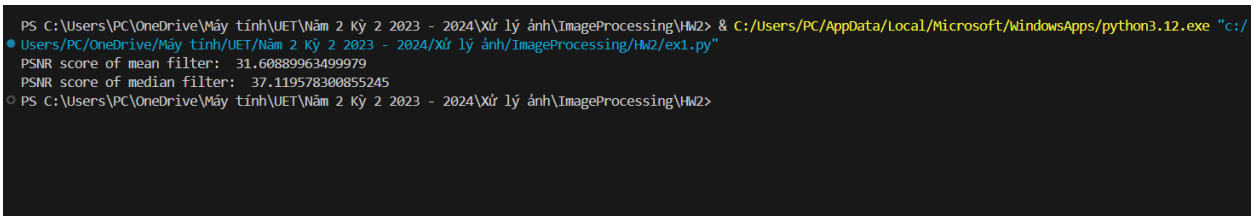

Figure 4: Ảnh kết quả

PSNR score of mean filter: 31.60889963499979
PSNR score of median filter: 37.119578300855245

Dựa trên điểm số PSNR thu được, nhận thấy rằng bộ lọc trung vị (median filter) có hiệu suất cao hơn so với bộ lọc trung bình (mean filter). Điều này cho thấy rằng bộ lọc trung vị (median filter) giúp bảo tồn chất lượng hình ảnh tốt hơn và giảm biến dạng hơn so với bộ lọc trung bình (mean filter).

Khuyến nghị sử dụng bộ lọc trung vị (median filter) hơn là bộ lọc trung bình (mean filter) cho hình ảnh cụ thể này. Điều này sẽ giúp cải thiện chất lượng hình ảnh và giảm nhiễu một cách hiệu quả.

3.2 Biến đổi Fourier

3.2.1 Biến đổi Fourier 1 chiều

Biến đổi Fourier rời rạc (DFT) 1 chiều chuyển đổi một tín hiệu từ miền thời gian sang miền tần số.
Công thức thực hiện DFT 1 chiều cho tín hiệu $f[n]$ với độ dài N là:

$$F(s) = \frac{1}{N} \sum_{n=0}^{N-1} f[n] \cdot e^{-2\pi i \cdot s \cdot n/N}$$

và ngược lại:

$$f[n] = \sum_{s=0}^{N-1} F(s) \cdot e^{2\pi i \cdot s \cdot n/N}$$

Trong đó s biểu thị tần số, và n biểu thị thứ tự mẫu.

Triển khai hàm DFT_slow để thực hiện biến đổi Fourier rời rạc (DFT) trên tín hiệu một chiều

1. Khởi tạo một mảng phức DFT với kích thước N để lưu trữ kết quả DFT.
2. Duyệt qua mọi giá trị của k từ 0 đến $N-1$.
3. Trong mỗi vòng lặp k, tính tổng của các phần tử $x(n) \cdot e^{-j\frac{2\pi}{N}kn}$ với n từ 0 đến $N-1$.
4. Gán giá trị của tổng vào phần tử tương ứng của mảng DFT.
5. Trả về mảng DFT chứa kết quả DFT của tín hiệu đầu vào.

Listing 5: Triển khai hàm DFT_slow

```
    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_)  # Initialize DFT array

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

      return DFT
```

3.3 Biến đổi Fourier 2 chiều

Biến đổi Fourier 2 chiều chuyển đổi một tín hiệu hai chiều từ miền thời gian sang miền tần số. Thực hiện mô phỏng hoạt động của hàm `fft2` trong thư viện `numpy.fft` bằng cách sử dụng hàm `fft` được thiết kế cho biến đổi Fourier 1 chiều.

1. **Biến đổi Fourier trên các hàng:** Trong bước này, hàm `np.fft.fft` được sử dụng để thực hiện biến đổi Fourier trên mỗi hàng của tín hiệu đầu vào.
2. **Biến đổi Fourier trên các cột:** Sau khi hoàn thành bước 1, chúng ta có một ma trận mới trong đó mỗi hàng là kết quả của biến đổi Fourier trên từng hàng của tín hiệu ban đầu. Trong bước này, chúng ta lại sử dụng hàm `np.fft.fft` để thực hiện biến đổi Fourier trên mỗi cột của ma trận đã được biến đổi ở bước trước.
3. **Kết quả và trả về:** Kết quả cuối cùng là một ma trận 2D, trong đó mỗi phần tử là kết quả của biến đổi Fourier 2D trên tín hiệu đầu vào. Hàm trả về ma trận này là kết quả của biến đổi Fourier 2D của tín hiệu đầu vào.

Listing 6: Triển khai hàm DFT_2D

```
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
    """
    # Compute row-wise FFT
    row_fft = np.fft.fft(gray_img, axis=1)

    # Compute column-wise FFT
    row_col_fft = np.fft.fft(row_fft, axis=0)

    return row_fft, row_col_fft
```

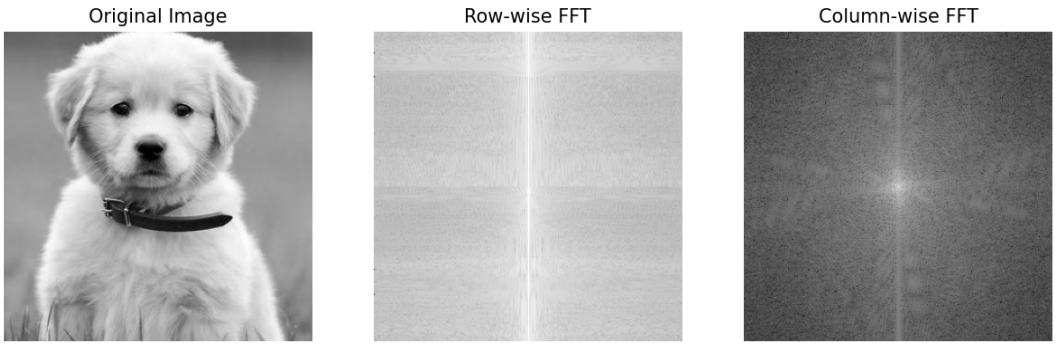

Figure 5: Ảnh kết quả cho bài tập 2D Fourier Transform

3.3.1 Xử lý tần số

Bạn sẽ thực hiện một thủ tục để lọc tần số trong ảnh. Quy trình này bao gồm:

1. Biến đổi Fourier 2 chiều của ảnh sử dụng hàm `fft2`.
2. Dịch chuyển các hệ số tần số để đưa tần số gốc vào trung tâm sử dụng hàm `fftshift`.
3. Lọc trong miền tần số sử dụng mặt nạ được chỉ định.
4. Dịch chuyển các hệ số tần số trở lại vị trí ban đầu sử dụng hàm `ifftshift`.
5. Thực hiện biến đổi nghịch Fourier sử dụng hàm `ifft2`.

Listing 7: Triển khai hàm filter_frequency

```
  def filter_frequency(orig_img, 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
"""
# 1. Transform using fft2
f_img = np.fft.fft2(orig_img)
# 2. Shift frequency coefs to center using fftshift
f_img_shifted = np.fft.fftshift(f_img)
# 3. Filter in frequency domain using the given mask
f_img_filtered = f_img_shifted * mask
# 4. Shift frequency coefs back using ifftshift
f_img_filtered_shifted = np.fft.ifftshift(f_img_filtered)
# 5. Invert transform using ifft2
img = np.abs(np.fft.ifft2(f_img_filtered_shifted))

f_img_filtered = np.abs(f_img_filtered)
return f_img_filtered, img
```

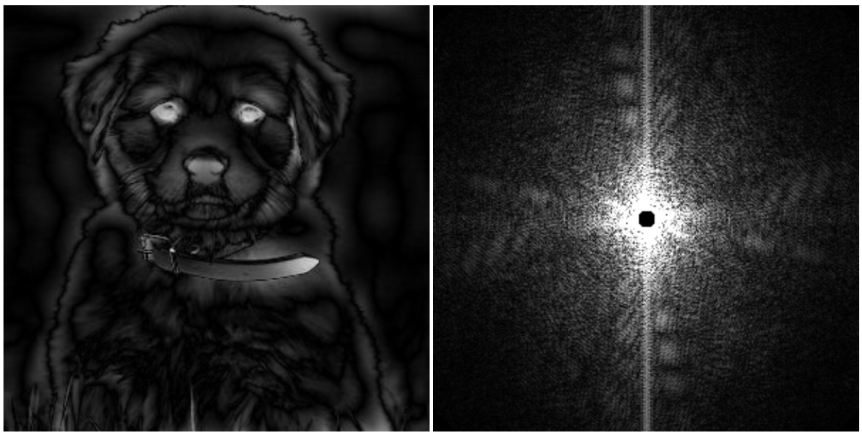

Figure 6: Ảnh kết quả lọc đi miền tần số thấp

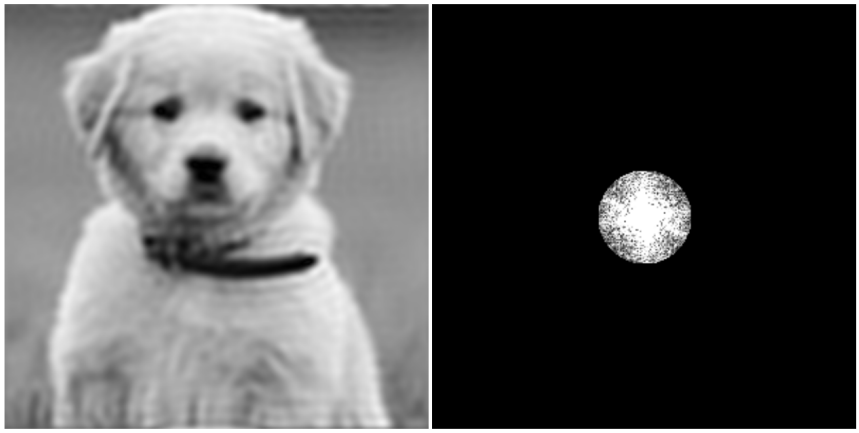

Figure 7: Ảnh kết quả lọc đi miền tần số cao

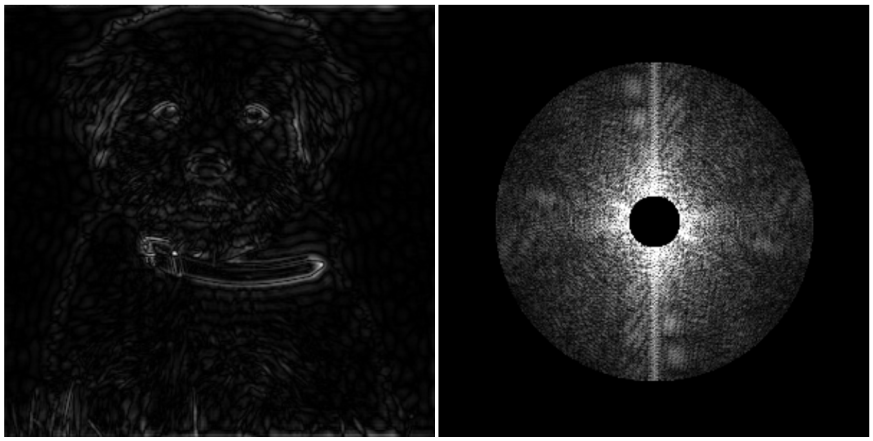

Figure 8: Ảnh kết quả lọc đi miền tần số thấp và cao

3.3.2 Tạo ảnh Hybrid

Để tạo ảnh lai, sử dụng biến đổi Fourier 2 chiều. Quy trình này bao gồm:

1. Biến đổi Fourier 2 chiều của hai ảnh đầu vào.
2. Dịch chuyển các hệ số tần số của ảnh đầu vào để đưa tần số gốc vào trung tâm.
3. Tạo mặt nạ dựa trên bán kính được chỉ định.
4. Kết hợp tần số của hai ảnh sử dụng mặt nạ.
5. Dịch chuyển các hệ số tần số trở lại vị trí ban đầu.
6. Thực hiện biến đổi nghịch Fourier để tái tạo ảnh lai.

Listing 8: Triển khai hàm create_hybrid_img

```
  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.
"""
# 1: Transform using fft2
f_img1 = np.fft.fft2(img1)
f_img2 = np.fft.fft2(img2)

# 2: Shift frequency coefs to center using fftshif
f_img1_shifted = np.fft.fftshift(f_img1)
f_img2_shifted = np.fft.fftshift(f_img2)

# 3: Create a mask based on the given radius (r) parameter
mask = np.zeros_like(f_img1_shifted)
H, W = mask.shape
center_h, center_w = H // 2, W // 2
for i in range(H):
  for j in range(W):
    if np.sqrt((i - center_h)**2 + (j - center_w)**2) < r:
      mask[i, j] = 1

# 4: Combine frequency of 2 images using the mask
f_img_hybrid = f_img1_shifted * mask + f_img2_shifted * (1 - mask)

# 5: Shift frequency coefs back using ifftshift
f_img_hybrid_unshifted = np.fft.ifftshift(f_img_hybrid)

# Step 6: Invert transform using ifft2
hybrid_img = np.abs(np.fft.ifft2(f_img_hybrid_unshifted))

return hybrid_img
```

Figure 9: Ảnh Hybrid

4 Kết luận

Bài tập này đã giúp hiểu rõ về cách hoạt động của các bộ lọc ảnh cơ bản và thuật toán Biến đổi Fourier trong xử lý ảnh. Qua đó, nắm vững kiến thức về các khái niệm và phương pháp thường được sử dụng trong lĩnh vực xử lý ảnh.
Chi tiết về mã nguồn có thể được tìm thấy tại:
Link Github: INT3404E_HW2