INT3404E 20 - Image Processing: Homeworks week 10

22028129 Tăng Vĩnh Hà

1 Tóm tắt báo cáo

Mô tả bài toán: Một bức ảnh có thể được biểu diễn ở miền không gian và miền tần số. Trong miền không gian, một bức ảnh có kích cỡ **m x n** sẽ có giá trị tại điểm **(i,j)** là giá trị cường độ xám (hệ trục tọa độ theo Ox và Oy với điểm (0,0) nằm ở góc bên trái trên cùng bức ảnh).

Trong miền tần số, một bức ảnh có kích cỡ **m x n** sẽ biểu diễn cường độ của từng tần số tín hiệu được chuyển từ cường độ xám của ảnh gốc. Lúc này điểm (0,0) sẽ là điểm chính giữa bức ảnh đại diện cho tần số cơ bản nhất (w_0, w_0) (sẽ có tần số theo trục x và tần số theo trục y). Như vậy các vùng tần số thấp sẽ tập trung quanh trung tâm bức ảnh, các vùng có tần số cao sẽ tập trung ngoài rìa bức ảnh (những vùng có chi tiết sắc nét, nhiều cạnh trong biểu diễn miền không gian của ảnh gốc ban đầu).

Tóm tắt: Báo cáo sẽ đi qua 2 phần, các hàm phục vụ cho xử lý ảnh trong miền không gian và miền tần số

- **Miền không gian**: gồm các hàm padding pictures, mean filter và median filter, hàm tính chỉ số PNSR.
- **Miền tần số**: gồm các hàm DFT slow cho biến đổi Fourier với dữ liệu 1 chiều, hàm DFT 2D cho dữ liệu 2 chiều, hàm filter frequency cho việc lọc dữ liệu (loại bỏ những tần số thấp) theo một mask cho trước, hàm hybrid image để tạo ảnh hybrid từ 2 ảnh lọc bỏ tần số thấp - tần số cao.

Một số thư viện đã được import sẵn để phục vụ cho việc code các hàm trong báo cáo như phía dưới:

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

2 Miền không gian

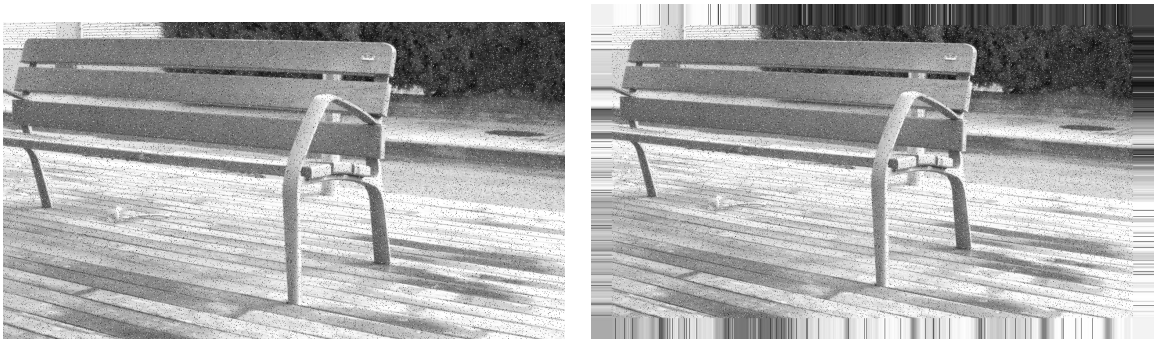
2.1 Hàm: padding pictures

- **Chức năng** : Thực hiện pad một bức ảnh theo kích cỡ của filter cho trước để sau này áp dụng các filter trong quá trình lọc ảnh (hay thực hiện tích chập của filter với ảnh). Hàm hiện tại đang sử dụng cách pad là copy các giá trị lân cận cho những vùng được pad thêm.
- **Giải thích** : Thực hiện như comment và code ở phía dưới. Hàm hiện tại đang sử dụng cách pad là copy các giá trị lân cận cho những vùng được pad thêm.

```
def padding_img(img, filter_size=3):
    # dividing to get the size of padded part
    pad_size = int(filter_size/2)
    h, w = img.shape

    # create a temporary image with the size like that
    padded_img = np.zeros((h + 2 * pad_size, w + 2 * pad_size), dtype=img.dtype)

    # copying original image into padded_img
    padded_img[pad_size : -pad_size, pad_size : -pad_size] = img

    # pad rows
    padded_img[:pad_size, pad_size : - pad_size] = img[0,:]
    padded_img[- pad_size:, pad_size : - pad_size] = img[h - 1, :]
```

```
    # pad columns
    padded_img[pad_size : - pad_size, :pad_size] = img[:,0].reshape(-1, 1)
    padded_img[pad_size : - pad_size, - pad_size:] = img[:,w - 1].reshape(-1, 1)

    # pad 4 corners
    padded_img[:pad_size, :pad_size] = img[0,0]
    padded_img[:pad_size, - pad_size:] = img[0,-1]
    padded_img[- pad_size:, :pad_size] = img[-1,0]
    padded_img[- pad_size:, - pad_size:] = img[-1,-1]

    return padded_img
```

Output: Hình minh họa khi việc padding thực hiện với kích cỡ filter là 100 x 100 (kích cỡ filter lớn hẳn để thể hiện rõ kết quả của hàm).



(a) Input: Original image

(b) Output: Padded version with the filter size equals to 100

Hình 1: Kết quả áp dụng của hàm padding_img

2.2 Hàm: mean filter

- **Chức năng** : Thực hiện việc lọc ảnh sử dụng bộ lọc trung bình, thường được sử dụng khi ảnh bị nhiễu Gaussian.
- **Giải thích** : Thực hiện như comment và code ở phía dưới; được implement bằng cách sử dụng 2 vòng `for` để dịch filter theo từng cột rồi xuống theo từng hàng, `cur_sec` là vùng hiện tại của bức ảnh ta thực hiện việc nhân tích chập với filter, hiện tại trong hàm này đang sử dụng filter `mean_win` - sau khi nhân với nhau là lấy tổng các giá trị gray scale của ảnh rồi chia trung bình. Anchor point là điểm chính giữa filter.

```
def mean_filter(img, filter_size=3):
    h, w = img.shape
    # create a temporary image with the size like that
    mean_img = np.zeros((h,w), dtype=img.dtype)
    padded_img = padding_img(img)
    # select filters
    pad_size = int(filter_size/2)
    mean_win = np.full((filter_size, filter_size), 1/(filter_size * filter_size), dtype=float)
    for i in range(pad_size, h + pad_size):
        for j in range(pad_size, w + pad_size):
            cur_sec = padded_img[i - pad_size:i + pad_size + 1, j - pad_size:j + pad_size + 1]
            res = np.multiply(cur_sec, mean_win)
            mean_img[i - pad_size, j - pad_size] = np.sum(res)
```

```
    return mean_img
```

Output: có một nhận xét là dễ thấy, bức ảnh khi áp dụng bộ lọc mean trong trường hợp này không có thay đổi gì mấy so với ảnh gốc.

(a) Input: Original image

(b) Output: Original image applying mean filter

Hình 2: Kết quả áp dụng của hàm mean_filter

2.3 Hàm: median filter

- **Chức năng** : Thực hiện việc lọc ảnh sử dụng bộ lọc trung vị, thường được sử dụng khi ảnh bị nhiễu có nhiều outliers.
- **Giải thích** : Thực hiện như comment và code ở phía dưới; được implement bằng cách sử dụng 2 vòng `for` để dịch filter theo từng cột rồi xuống theo từng hàng, `cur_sec` là vùng hiện tại của bức ảnh ta thực hiện việc lọc theo filter trung vị - sắp xếp các điểm trong vùng `cur_sec` theo thứ tự từ cao xuống thấp và lấy điểm trung vị (điểm ở giữa) để gán giá trị cho anchor point. Anchor point vẫn là điểm chính giữa filter.

```
def median_filter(img, filter_size=3):
    h, w = img.shape
    # create a temporary image with the size like that
    med_img = np.zeros((h,w), dtype=img.dtype)
    padded_img = padding_img(img)
    pad_size = int(filter_size/2)
    for i in range(pad_size, h + pad_size):
        for j in range(pad_size, w + pad_size):
            cur_sec = padded_img[i - pad_size:i + pad_size + 1, j - pad_size:j + pad_size + 1]
            res = np.median(cur_sec)
            med_img[i - pad_size, j - pad_size] = res

    return med_img
```

Output: dễ thấy bức ảnh khi áp dụng bộ lọc median có thay đổi là trở nên mịn hơn hẳn so với ảnh gốc.

(a) Input: Original image (b) Output: Original image applying median filter

Hình 3: Kết quả áp dụng của hàm median_filter

2.4 Hàm: psnr

Peak Signal-to-Noise Ratio (PSNR) là tỷ lệ giá trị tối đa của pixel trên nhiễu (MSE) ảnh hưởng đến chất lượng của pixel. PSNR càng cao thì sai số của nhiễu càng nhỏ và được biểu thị dưới đơn vị là logarit cơ số 10. Công thức như sau:

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

MAX là giá trị pixel tối đa có thể (thường là 255 đối với hình ảnh 8 bit). Nhiễu (MSE) của một bức ảnh so với ảnh gốc (ảnh ground truth - không bị nhiễu) được tính theo công thức sau:

$$\text{MSE} = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (O(i,j) - D(i,j))^2$$

Với **O** là ma trận giá trị gray scale của ảnh gốc và **D** là ma trận giá trị gray scale của ảnh bị nhiễu.

```
def psnr(gt_img, smooth_img):
    m,n = gt_img.shape
    tmp = np.subtract(gt_img, smooth_img) * np.subtract(gt_img, smooth_img)
    mse = np.sum(tmp) * (1/(m*n))
    if (mse == 0):
        return np.Infinity
    psnr = 10 * np.log10((255 ** 2)/mse)
    return psnr
```

Output:

Giá trị chỉ số PNSR của ảnh bị nhiễu sau khi áp dụng

- Bộ lọc trung bình được implement như trên là: 31.605849430056452
- Bộ lọc trung vị được implement như trên là 37.119578300855245

Dễ thấy trong trường hợp này nên sử dụng bộ lọc trung vị khi chỉ số PSNR của nó cao hơn; chứng tỏ việc xử lý nhiễu tốt hơn và cũng thể thấy rõ từ việc ảnh dùng median filter cũng mịn, không còn nhiều nhiễu như mean filter.

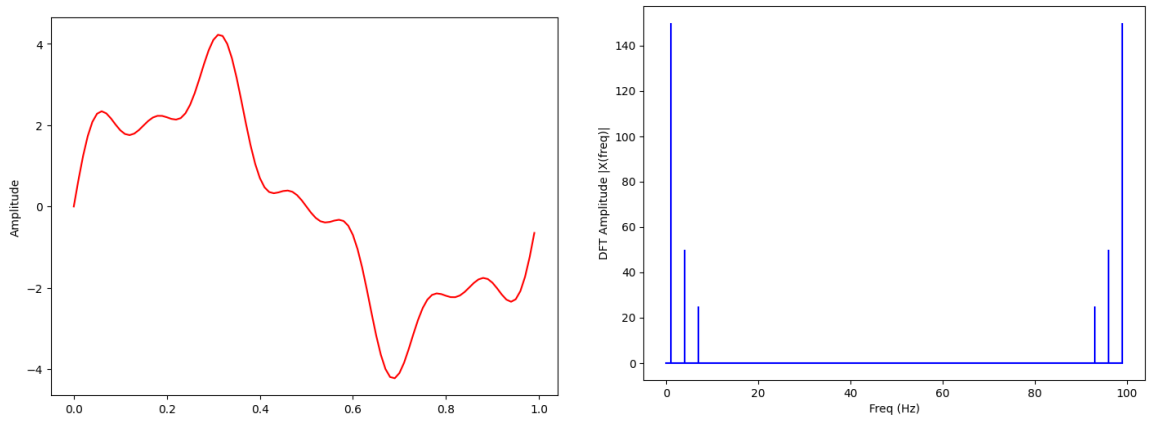
3 Miền tần số

3.1 Hàm: DFT 1D

Hàm `DFT_slow` thực hiện chuyển `data` là tín hiệu 1 chiều trong miền thời gian sang tín hiệu 1 chiều trong miền tần số bằng cách sử dụng công thức biến đổi Fourier như trong `hw2_title`, được implement cụ thể bằng tạo ra một ma trận và thực hiện phép nhân cũng như trong `hw2_title` (em xin phép không nhắc lại ạ).

```
def DFT_slow(data):
  N = len(data)
  matrix_e = np.zeros((N,N), dtype=complex)
  for k in range(N):
    for n in range(N):
      matrix_e[k,n] = np.exp(-2j * np.pi * k * n / N)
  X = np.dot(matrix_e, data)
  return X
```

Output: một ví dụ cho kết quả của việc áp dụng hàm `DFT_slow` để biến đổi tín hiệu từ miền thời gian sang miền tần số. Tín hiệu trong miền thời gian được sử dụng được thể hiện ở phần code trong đoạn sau.



(a) Input: Tín hiệu trong miền thời gian

(b) Output: Tín hiệu trong miền tần số (phổ biên độ)

Hình 4: Kết quả áp dụng của hàm DFT_slow

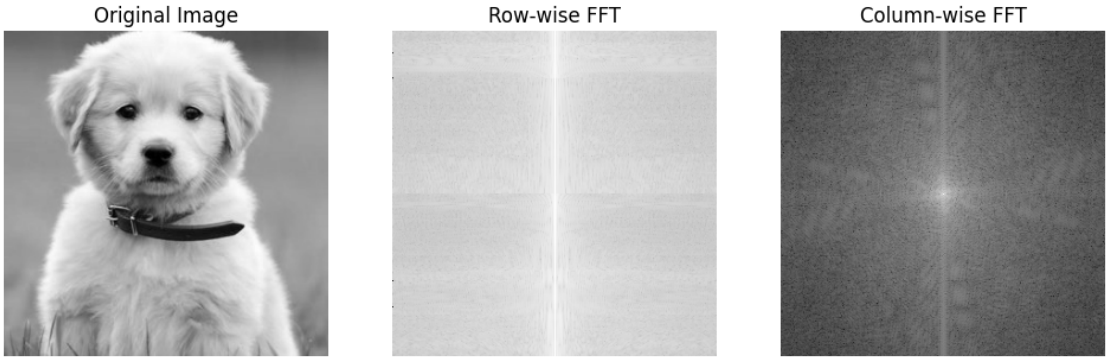
Đoạn code để generate ra tín hiệu trong hình 4.a em xin phép không trình bày trong báo cáo vì dài và em đã đăng tải lên github của em (tham khảo từ trang Python numerical methods).

3.2 Hàm: DFT 2D

Hàm `DFT_2D` thực hiện chuyển `gray_img` là tín hiệu 2 chiều của một ảnh trong miền không gian sang tín hiệu 2 chiều của một ảnh trong miền tần số bằng cách sử dụng công thức biến đổi Fourier theo từng hàng và sau đó theo từng cột. Cụ thể:

- Sau dòng 1 trong đoạn code phía dưới; ta sẽ thu được `row_fft` là một ma trận số ảo (từ đầu vào là `gray_img` - một ma trận số thực biểu diễn giá trị gray scale của bức ảnh ban đầu) với từng hàng là kết quả của việc áp dụng biến đổi Fourier 1D lên chính hàng đó trong ma trận gốc gray scale.
- Ta sẽ thu được `row_col_fft` là 1 ma trận số ảo (từ đầu vào là `row_fft` - một ma trận số ảo như giải thích phía trên). Lúc này, dòng code đang thực hiện việc áp dụng biến đổi Fourier 1D lên từng cột của `row_fft`.

```
def DFT_2D(gray_img):
    row_fft = np.fft.fft(gray_img, axis=1)
    row_col_fft = np.fft.fft(row_fft, axis=0)
    return row_fft, row_col_fft
```



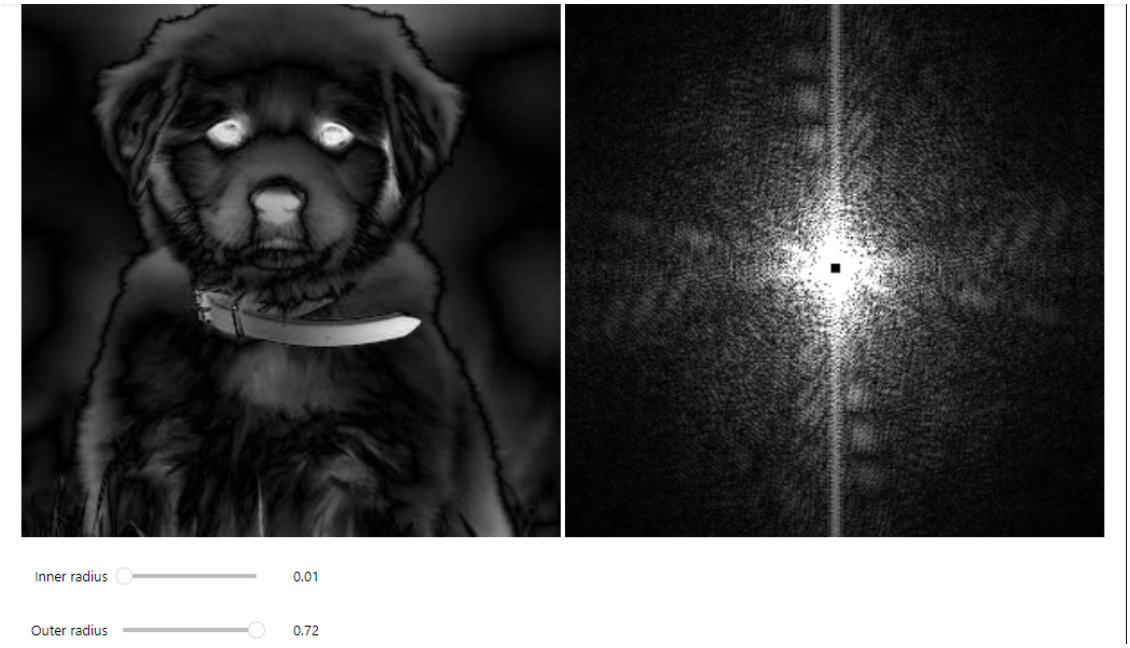
Hình 5: Kết quả áp dụng của hàm DFT_2D: áp dụng DFT lên hàng và áp dụng DFT lên hàng cột

3.3 Hàm: filter frequency

- Chức năng: Lọc một bức ảnh trong miền tần số.
- Giải thích: Implement theo 6 bước như đã đề cập trong `hw2_title`. Chỉ có một lưu ý nhỏ là việc hiển thị ảnh gốc là phải hiển thị giá trị tuyệt đối (hay nói cách khác là pha biên độ) của ma trận sau khi đã được invert về sử dụng hàm `ifft2`.

```
def DFT_2D(gray_img):
  # 1. Transform using fft2
  freq = np.fft.fft2(orig_img)
  # 2. Shift frequency coefs to center using fftshift
  f_img_shifted = np.fft.fftshift(freq)
  # 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_unshifted = np.fft.ifftshift(f_img_filtered)
  # 5. Invert transform using ifft2
  img_filtered = np.fft.ifft2(f_img_unshifted)
  f_img_display = np.abs(f_img_filtered)
  # Convert the filtered image to the appropriate data type for display
  img_display = np.abs(img_filtered)

  return f_img_display, img_display
```

Output: kết quả của việc áp dụng hàm như phía dưới (chi tiết hơn trên github của em). Bức ảnh gốc ban đầu thay đổi khi bán kính của cường độ tần số bị loại bỏ trong ảnh miền tần số thay đổi (chấm đen đại diện cho vùng vòng tròn mà những tần số bị loại bỏ).



Hình 6: Kết quả của khi áp dụng hàm DFT_2D

3.4 Hàm: create hybrid image

- Chức năng: Tạo ra ảnh hỗn hợp từ việc cộng 2 ảnh trong miền tần số với nhau.
- Giải thích: Implement theo 6 bước như đã đề cập trong `hw2_title`. Cơ bản diễn ra như sau
  - Bước 1: Biến 2 ảnh sang miền tần số
  - Bước 2 + 3: Áp dụng mask lên 2 ảnh với bán kính r cho trước; cụ thể là 1 ảnh chỉ giữ lại vùng bán kính r ở tâm - còn đâu 1 ảnh chỉ giữ vùng ngoài bán kính r đó.
  - Bước 4: Tạo ảnh từ 2 ảnh sau khi đã áp dụng mask bằng cách cộng vào với nhau.
  - Bước 5 + 6: Chuyển lại ảnh từ miền tần số sang miền không gian. Vẫn lưu ý là phải hiển thị giá trị tuyệt đối (hay nói cách khác là pha biên độ) như ở phần `filter_frequency` đã nhắc đến.

```
def create_hybrid_img(img1, img2, r):
# 1. Transform using fft2
  f_img1 = np.fft.fft2(img1)
  f_img2 = np.fft.fft2(img2)
# 2. Shift frequency coefs to center using fftshift
  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, as described in Figure 5
  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)
```

```
20  # 6. Invert transform using ifft2
    hybrid_img = np.abs(np.fft.ifft2(f_img_hybrid_unshifted))
    return hybrid_img
```

Output: kết quả của việc áp dụng hàm `create_hybrid_img` như phía dưới (chi tiết hơn trên github của em). 2 bức ảnh gốc ban đầu là: ảnh gray scale của một cô gái (ảnh được nhân với `1 - mask` hay chỉ giữ vùng ngoài bán kính r; ngoài mask) và ảnh gray scale của một chú chó (ảnh được nhân với `mask` hay chỉ giữ vùng ngoài bán kính r; trong mask).

Hình 7: Kết quả của khi áp dụng hàm create_hybrid_img

Dễ thấy; thao tác thực hiện lên ảnh cô gái là thao tác loại bỏ những vùng tần số thấp; hay là chỉ giữ những vùng tần số cao là các chi tiết cạnh. Còn ảnh chú chó thì ngược lại; và khi kết hợp thì ta sẽ có bức ảnh như trên.