INT3404E 20 - Image Processing: Homeworks 2

Tong Minh Tri - 21020249

1 Image Filtering

(a)

```python
def padding_img(img, filter_size=3):

    # Pad the image using replicate padding.
    height, width = img.shape
    half_filter_size = filter_size // 2
    padded_height = height + 2 * half_filter_size
    padded_width = width + 2 * half_filter_size

    padded_img = np.zeros((padded_height, padded_width), dtype=img.dtype)

    for i in range(height):
        for j in range(width):
            padded_img[i + half_filter_size, j + half_filter_size] = img[i, j]
    for i in range(half_filter_size):
        padded_img[i, half_filter_size:-half_filter_size] = img[0]
        padded_img[-i - 1, half_filter_size:-half_filter_size] = img[-1]
        padded_img[half_filter_size:-half_filter_size, i] = img[:, 0]
        padded_img[half_filter_size:-half_filter_size, -i - 1] = img[:, -1]

    return padded_img
```

Figure 1: Padding function

```python
def mean_filter(img, filter_size=3):

    padded_img = padding_img(img, filter_size)
    height, width = padded_img.shape

    smoothed_img = np.zeros_like(img)

    for i in range(height - filter_size + 1):
        for j in range(width - filter_size + 1):
            window = padded_img[i:i + filter_size, j:j + filter_size]
            smoothed_img[i, j] = np.mean(window, axis=(0, 1))

    return smoothed_img
```

Figure 2: Mean Filtering

```python
def median_filter(img, filter_size=3):

    padded_img = padding_img(img, filter_size)
    height, width = padded_img.shape

    smoothed_img = np.zeros_like(img)

    for i in range(height - filter_size + 1):
        for j in range(width - filter_size + 1):
            window = padded_img[i:i + filter_size, j:j + filter_size]
            smoothed_img[i, j] = np.median(window, axis=(0, 1))

    return smoothed_img
```

Figure 3: Median Filtering

(b)

```python
def psnr(gt_img, smooth_img):

    gt_img = gt_img.astype(np.float64)
    smooth_img = smooth_img.astype(np.float64)

    # Calculate the squared error (MSE)
    mse = np.mean((gt_img - smooth_img) ** 2)

    # Find the maximum possible pixel value for the given data type
    # max_pixel_value = np.iinfo(gt_img.dtype).max
    max_pixel_value = 255

    # Calculate PSNR using the formula: PSNR = 10 * log10((max_pixel_value^2) / MSE)
    psnr_score = 10 * np.log10((max_pixel_value ** 2) / mse)

    return psnr_score
```

Figure 4: PSNR

(c) Because the mean filter gives higher PSNR score so we will choose mean filter for provided images

```
PSNR score of mean filter:  18.294085709147602
PSNR score of median filter:  17.835212311092135
```

Figure 5: PSNR score

2 Fourier Transform

(a) 1D Fourier Transform

```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
    """
    # You need to implement the DFT here
    N = len(data)
    n = np.arange(N)
    k = n.reshape((N, 1))
    # Exponential term in the DFT formula
    exp_term = np.exp(-2j * np.pi * k * n / N)
    # Calculate DFT using the formula
    DFT = np.dot(exp_term, data)
    return DFT
```

Figure 6: DFT slow

(b) 2D Fourier Transform

```python
def DFT_2D(gray_img):

    H, W = gray_img.shape
    # Initialize arrays to store row-wise and column-wise FFTs
    row_fft = np.zeros((H, W), dtype=np.complex_)
    row_col_fft = np.zeros((H, W), dtype=np.complex_)

    # Compute row-wise FFT
    for i in range(H):
        row_fft[i, :] = np.fft.fft(gray_img[i, :])

    # Compute column-wise FFT
    for j in range(W):
        row_col_fft[:, j] = np.fft.fft(row_fft[:, j])

    return row_fft, row_col_fft
```

Figure 7: DFT 2D

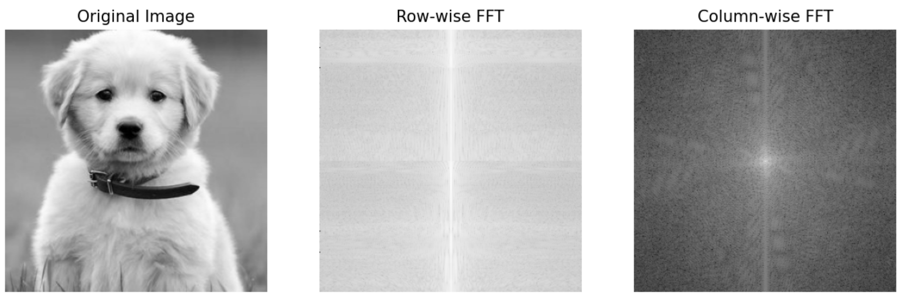

Figure 8: DFT 2D result

(c) Frequency Removal Procedure

```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
    """
    # Transform using fft2
    f_img = fft2(orig_img)

    # Shift frequency coefficients to center using fftshift
    f_img_shifted = fftshift(f_img)

    # Filter in frequency domain using the given mask
    f_img_filtered = f_img_shifted * mask

    # Shift frequency coefficients back using ifftshift
    f_img_filtered_shifted = ifftshift(f_img_filtered)

    # Invert transform using ifft2
    img = ifft2(f_img_filtered_shifted).real

    return f_img_filtered, img
```

Figure 9: Filter frequency

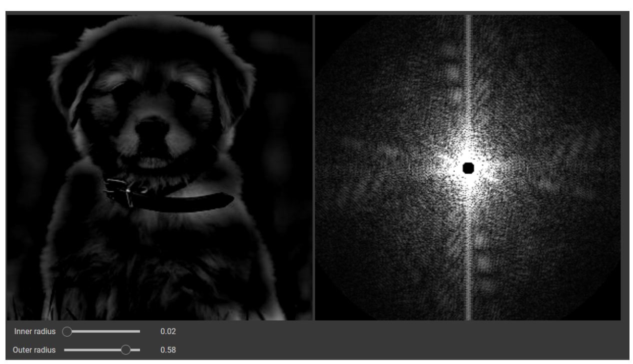

Figure 10: Filter frequency result

(d) Creating a Hybrid image

```python
def create_hybrid_img(img1, img2, r):

    # Perform Fourier Transform on both images
    img1_fft = fft2(img1)
    img2_fft = fft2(img2)

    # Shift frequency coefficients to center
    img1_fft_shifted = fftshift(img1_fft)
    img2_fft_shifted = fftshift(img2_fft)

    # Create a mask based on the given radius (r)
    rows, cols = img1.shape
    center_row, center_col = rows // 2, cols // 2
    mask = np.zeros((rows, cols))
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_row) ** 2 + (j - center_col) ** 2)
            if distance <= r:
                mask[i, j] = 1

    # Combine frequency of 2 images using the mask
    hybrid_img_fft = img1_fft_shifted * mask + img2_fft_shifted * (1 - mask)

    # Shift frequency coefficients back
    hybrid_img_fft_shifted = ifftshift(hybrid_img_fft)

    # Invert transform using inverse Fourier Transform
    hybrid_img = np.abs(ifft2(hybrid_img_fft_shifted))

    # Normalize the resulting image
    hybrid_img = cv2.normalize(hybrid_img, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

    return hybrid_img
```

Figure 11: Hybrid function

Figure 12: Hybrid image