INT3404E 20 - Image Processing: Homeworks week 2

Duong Bao Long - 21021514

1 Homework 3.1: Image Filtering

padding_img

```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.

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

    Return:
        padded_img: cv2 image: the padding image with replicated edge padding
    """

    height, width = img.shape[:2]
    pad_top = pad_bottom = filter_size // 2
    pad_left = pad_right = filter_size // 2

    padded_img = np.zeros(
        (height + pad_top + pad_bottom, width + pad_left + pad_right), dtype=img.dtype
    )
    top_pad = img[0]
    bottom_pad = img[-1]
    left_pad = np.repeat(img[:, 0], pad_left).reshape(height, pad_left)
    right_pad = np.repeat(img[:, -1], pad_right).reshape(height, pad_right)
    padded_img[:pad_top, pad_left:-pad_right] = top_pad
    padded_img[-pad_bottom:, pad_left:-pad_right] = bottom_pad
    padded_img[pad_top:-pad_bottom, :pad_left] = left_pad
    padded_img[pad_top:-pad_bottom, -pad_right:] = right_pad
    padded_img[:pad_top, :pad_left] = img[0, 0]  # Top-left corner
    padded_img[:pad_top, width + pad_left :] = img[0, -1]  # Top-right corner
    padded_img[height + pad_top :, :pad_left] = img[-1, 0]  # Bottom-left corner
    padded_img[height + pad_top :, width + pad_left :] = img[-1, -1]
    padded_img[pad_top : height + pad_top, pad_left : width + pad_left] = img

    return padded_img
```

mean_filter

```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.
    """
    # Need to implement here
    # Pad the image to maintain size after filtering
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros(img.shape, dtype=img.dtype)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            window_top = y
            window_bottom = window_top + filter_size
            window_left = x
            window_right = window_left + filter_size
            window = padded_img[window_top:window_bottom, window_left:window_right]
            mean_value = np.mean(window, axis=(0, 1))
            smoothed_img[y, x] = mean_value

    return smoothed_img
```

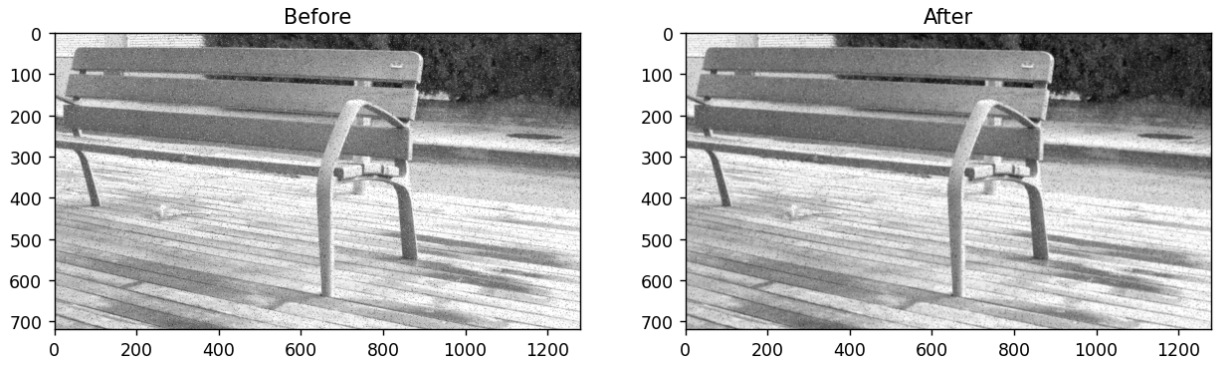

Figure 1: Mean filter

meadian_filter

```
def median_filter(img, filter_size=3):
    """
    Applies a median filter to an image using replicate padding.

    Args:
        img (np.ndarray): The original image as a NumPy array.
        filter_size (int, optional): The size of the square filter. Defaults to 3.

    Returns:
        np.ndarray: The smoothed image with the median filter.
    """

    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros(img.shape, dtype=img.dtype)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            window_top = y
            window_bottom = window_top + filter_size
            window_left = x
            window_right = window_left + filter_size
            window = padded_img[window_top:window_bottom, window_left:window_right]
            window_flat = window.flatten()
            median_value = np.median(window_flat)
            smoothed_img[y, x] = median_value
```

```
    return smoothed_img
```

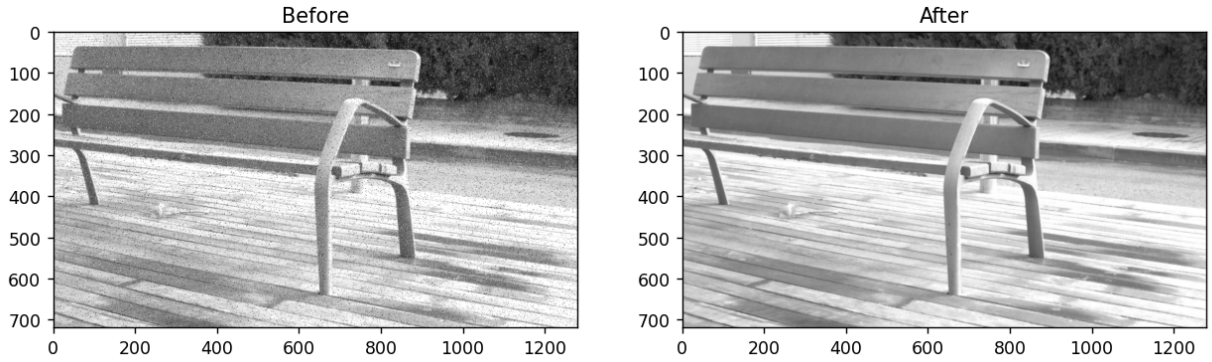

Figure 2: Median filter

PSNR score

```
def psnr(gt_img, smooth_img):
    """
    Calculates the PSNR (Peak Signal-to-Noise Ratio) between two images.

    Args:
        gt_img (np.ndarray): The ground truth image as a NumPy array.
        smooth_img (np.ndarray): The smoothed image as a NumPy array.

    Returns:
        float: The PSNR score (in dB).
    """
    gt_img = gt_img.astype(np.float64)
    smooth_img = smooth_img.astype(np.float64)
    mse = np.mean((gt_img - smooth_img) ** 2)
    if mse == 0:
        return 100.0  # Arbitrarily set PSNR to 100 for perfect match
    max_pixel = 255.0  # Adjust this if using images with different bit depth
    psnr = 10 * np.log10(max_pixel**2 / mse)

    return psnr
```

```
PSNR score of mean filter:  18.2940945653814
PSNR score of median filter:  17.835212311092135
```

Figure 3: PSNR score

2 Homework 3.2: Fourier Transform

2.1 HW 3.2.1: 1D Fourier Transform

1D Fourier Transfomr

```
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
    len_data = len(data)
    array = np.zeros(len_data, dtype=np.complex128)

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

    return array
```

2.2 HW 3.2.2: 2D Fourier Transform

2D Fourier Transform

```
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
    """
    # You need to implement the DFT here
    height, width = gray_img.shape

    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
```

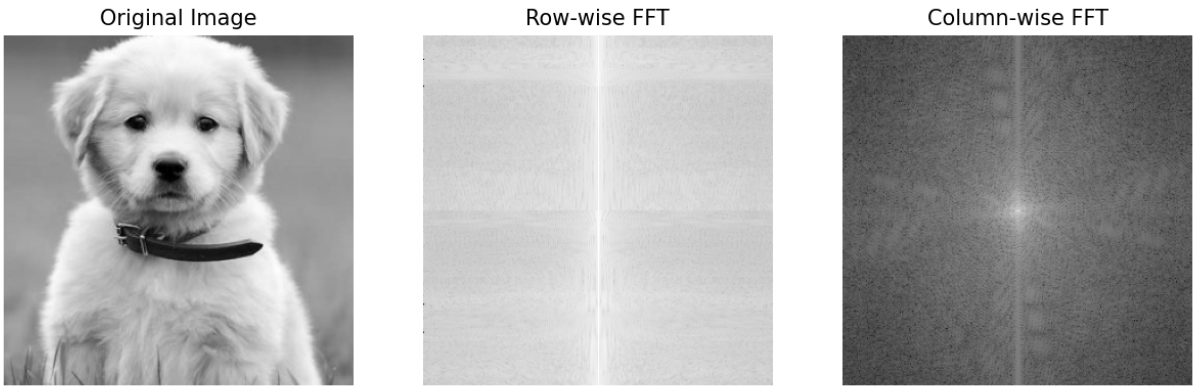

Figure 4: 1D and 2D Fourier Transform

2.3 HW 3.2.3: Frequency Removal Procedure

Filter Frequency

```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
    """
    # You need to implement this function
    f_img = np.fft.fft2(orig_img)

    f_img_shifted = np.fft.fftshift(f_img)

    f_img_filtered_shifted = f_img_shifted * mask

    f_img_filtered = np.fft.ifftshift(f_img_filtered_shifted)

    img = np.abs(np.fft.ifft2(f_img_filtered))

    return np.abs(f_img_filtered_shifted), img
```

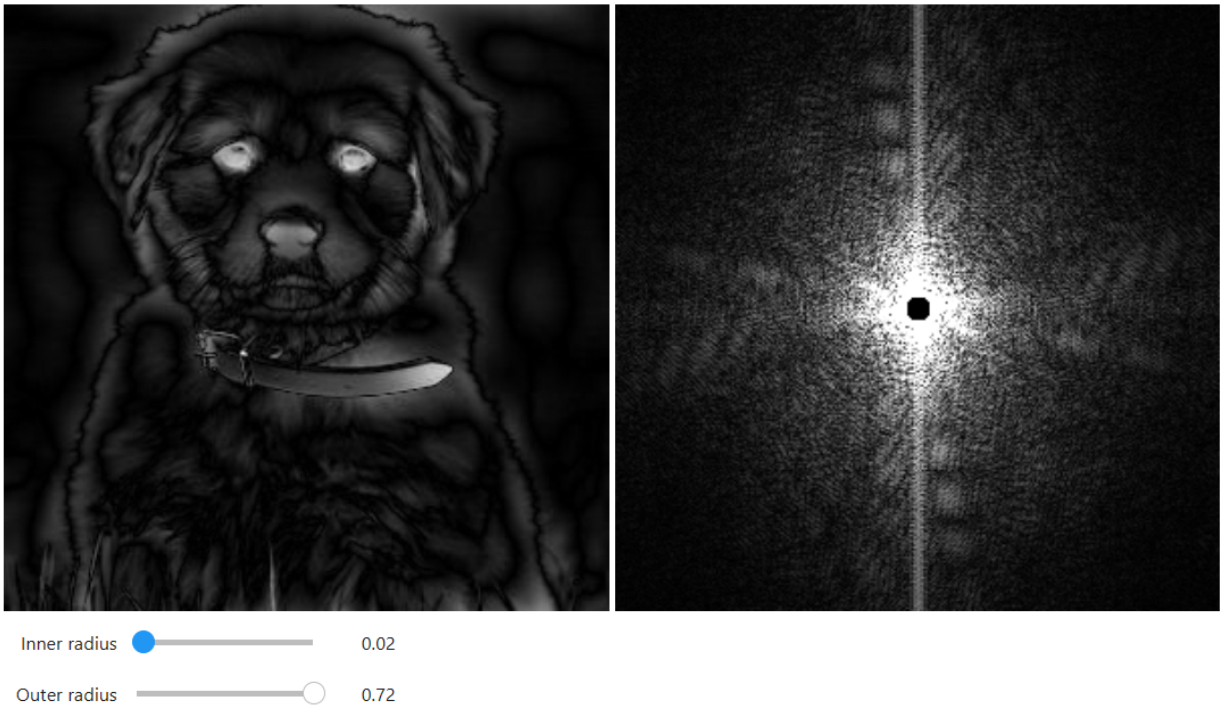

Figure 5: Frequency Removal Procedure

2.4 HW 3.2.4: Creating a Hybrid Image

Create Hybrid Image

```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.
    """
    f_img1 = np.fft.fftshift(np.fft.fft2(img1))
    f_img2 = np.fft.fftshift(np.fft.fft2(img2))

    rows, cols = img1.shape[:2]
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols), dtype=np.uint8)
    mask[crow - r : crow + r, ccol - r : ccol + r] = 1

    f_img1_filtered = f_img1 * mask
    f_img2_filtered = f_img2 * (1 - mask)

    f_hybrid = f_img1_filtered + f_img2_filtered

    hybrid_img = np.abs(np.fft.ifft2(np.fft.ifftshift(f_hybrid)))

    hybrid_img = np.clip(hybrid_img, 0, 255).astype(np.uint8)

    return hybrid_img
```

Figure 6: Hybrid Image