# INT3404E 20 - Image Processing: Homework 2

Lê Vũ Minh - 21020649

## 1 Composing requirements.txt

We have identified the following libraries that must be installed to properly work with this class, which are:

- `numpy`
- `matplotlib`
- `opencv-python`
- `scikit-image`

These are also the contents of the `requirements.txt` file.

## 2 Image Filtering

### 2.1 Implementing padding and filter functions

Implement the following functions in the supplied code file: *padding_img*, *mean_filter*, *median_filter*

1. Padding image:

```python
def padding_img(img, filter_size=3):
    """
        Conveniently, np has a padding functions np.pad() that we can utilize immediately.
        The "edge" mode essentially works as Replicate padding.
    """
    padding = int(filter_size / 2)
    return np.pad(img, (padding, padding), mode='edge')
```

2. Mean filter:

```python
def mean_filter(img, filter_size=3):
    # Retrieves the image and filter dimensions
    height, width = img.shape[:2]
    padding = int(filter_size / 2)

    # Intermediate variables for modifying the image
    smooth_img = np.zeros((height, width), dtype=np.uint8)
    pad_img = padding_img(img, filter_size)

    # The mean filter itself
    for i in range(padding, height - padding):
        for j in range(padding, width - padding):
            block = pad_img[i - padding : i + padding + 1, j - padding : j + padding + 1]
            mean = np.mean(block)
            smooth_img[i - padding, j - padding] = int(mean)

    return smooth_img
```

3. Median filter:

```python
def median_filter(img, filter_size=3):
    height, width = img.shape[:2]
    smooth_img = np.zeros((height, width), dtype=np.uint8)
```

```
    padding = int(filter_size / 2)
    pad_img = padding_img(img, filter_size)

    for i in range(padding, height - padding):
        for j in range(padding, width - padding):
            block = pad_img[i - padding : i + padding + 1, j - padding : j + padding + 1]
            mean = np.median(block)
            smooth_img[i - padding, j - padding] = int(mean)

    return smooth_img
```

Following the application of the filters, we can observe the noisy image before and after applying them, as follows:

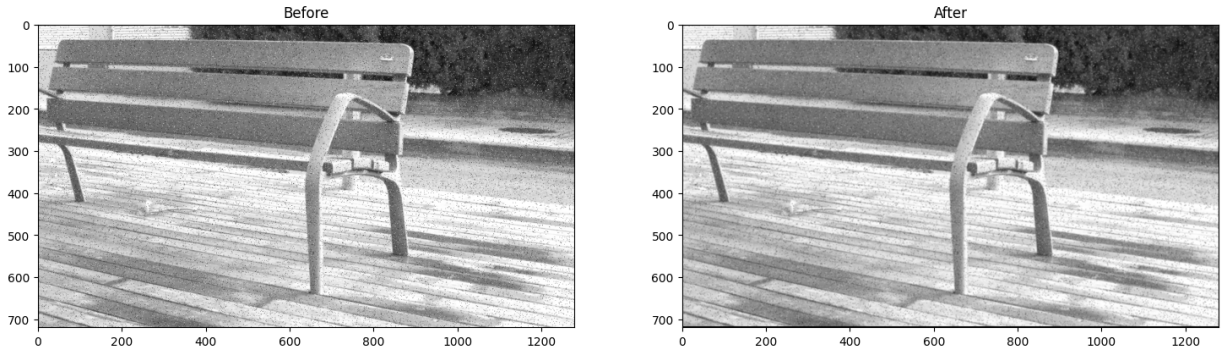


Figure 1: Mean filter result

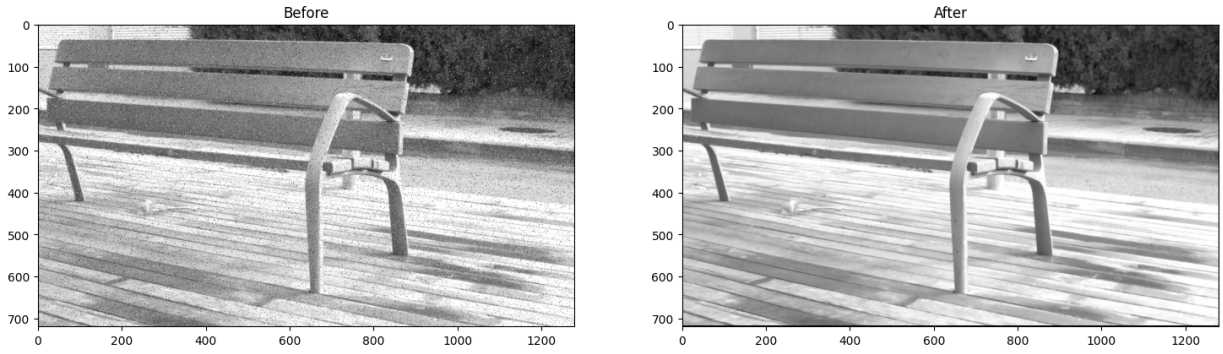


Figure 2: Median filter result

## 2.2 Implementing the PSNR metric

Implement the Peak Signal-to-Noise Ratio (PSNR) metric

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

where MAX is the maximum possible pixel value (typically 255 for 8-bit images), and MSE is the Mean Square Error between the two images.

```python
def psnr(gt_img, smooth_img):
    gt_img = gt_img.astype(np.float32)
    smooth_img = smooth_img.astype(np.float32)

    print(gt_img.shape, smooth_img.shape)

    mse = np.mean(np.square(gt_img - smooth_img))
    max_val = 255.0

    psnr = 10 * np.log10((max_val ** 2) / mse)

    return psnr
```

### 2.3 Evaluating which filter to choose

Considering the PSNR metrics, which filters should we choose between the mean and median filters for provided images?
According to MathWorks, the Peak Signal-to-Noise Ratio (PSNR) metric is used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image. The mean-square error (MSE) and the peak signal-to-noise ratio (PSNR) are used to compare image compression quality.
Performing the calculations on the ground-truth image and the filtered images, we get the following results:

- **Mean filter:** 17.600983862715204
- **Median filter:** 17.21038943522489

The mean filter scored higher on the PSNR metric, meaning that if we are to base our selection on this metric, the mean filter is to be preferred over the median filter.

## 3 Fourier Transform

### 3.1 1D Fourier Transform

Implement a function named DFT_slow to perform the Discrete Fourier Transform (DFT) on a one-dimensional signal:

```python
def DFT_slow(data):
    N = len(data)
    DFT = np.zeros(N, dtype=np.complex128)

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

    return DFT
```

### 3.2 2D Fourier Transform

Implement the function *DFT_2D* within the provided code. The procedure to simulate a 2D Fourier Transform is as follows:

1. Conducting a Fourier Transform on each row of the input 2D signal. This step transforms the signal along the horizontal axis.
2. Perform a Fourier Transform on each column of the previously obtained result.

The code is as follows:

```python
def DFT_2D(gray_img):
    height, width = gray_img.shape
    row_fft = np.zeros_like(gray_img, dtype=np.complex_)
    row_col_fft = np.zeros_like(gray_img, dtype=np.complex_)

    for i in range(height):
        row_fft[i, :] = np.fft.fft(gray_img[i, :])
    for j in range(width):
        row_col_fft[:, j] = np.fft.fft(row_fft[:, j])
    return row_fft, row_col_fft
```

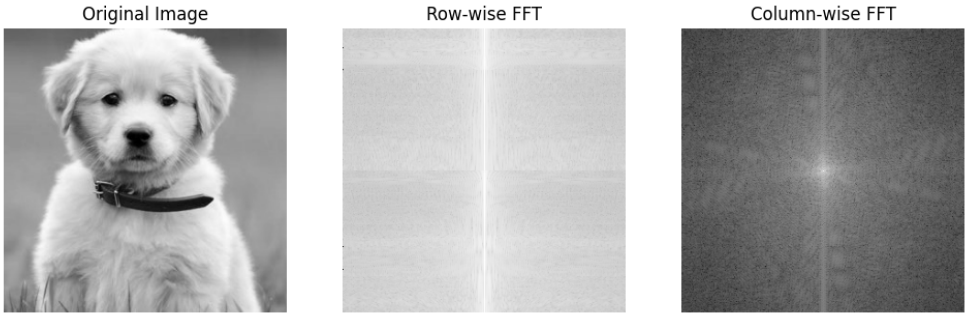


Figure 3: Output of the 2D Fourier Transform

## 3.3 Frequency Removal Procedure

Implement the *filter_frequency* function:

```python
def filter_frequency(orig_img, mask):
    f_img = np.fft.fft2(orig_img)
    f_img_shifted = np.fft.fftshift(f_img)
    f_img_filtered = f_img_shifted * mask
    f_img_filtered_shifted = np.fft.ifftshift(f_img_filtered)
    img = np.abs(np.fft.ifft2(f_img_filtered_shifted))

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

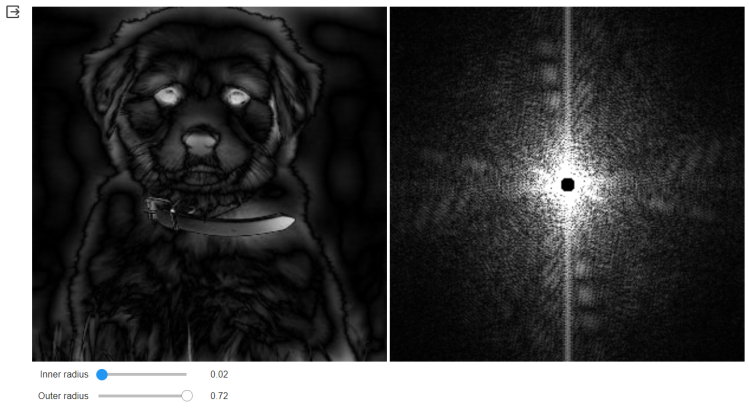


Figure 4: Output of the Frequency Removal exercise

### 3.4 Creating a Hybrid Image

Implement the *create_hybrid_img* function:

```
def create_hybrid_img(img1, img2, r):
    img1_fft_shifted = np.fft.fftshift(np.fft.fft2(img1))
    img2_fft_shifted = np.fft.fftshift(np.fft.fft2(img2))

    mask1 = np.zeros_like(img1_fft_shifted)
    rows, cols = mask1.shape
    center_row, center_col = int(rows / 2), int(cols / 2)
    y, x = np.ogrid[-center_row:rows - center_row, -center_col:cols - center_col]
    mask1[x*x + y*y <= r*r] = 1

    mask2 = (1 - mask1)
    hybrid_fft_shifted = img1_fft_shifted * mask1 + img2_fft_shifted * mask2
    hybrid_fft = np.fft.ifftshift(hybrid_fft_shifted)

    return np.abs(np.fft.ifft2(hybrid_fft))
```

Figure 5: Output of the Hybrid Image function