4주차 주간 보고

11조

20191446 박형욱

20192074 임지영

프로젝트 진행 상황

W3 (11/7 – 13)

- Median Filter 최적화

- Gaussian Filter 최적화

W4 (11/14 – 20)

- Bilateral Filter 구현

- Bilateral Filter 최적화

W5 (11/21 – 27)

- Non-local Means Filter 구현

Salt and Pepper Noise
Median Filter

ALGORITHM 5.11 Perform median filtering on an image

MEDIANFILTER (I, w)

Input: grayscale image I, width w of square window for computing median
Output: median-filtered image

1 $\tilde{w} \leftarrow \lfloor (w-1)/2 \rfloor$ ➤ Determine half-width.
2 **for** $y \leftarrow 0$ **to** $height - 1$ **do**
3      $\mathcal{W} \leftarrow \{(-\tilde{w}, -\tilde{w}), \ldots, (\tilde{w}, \tilde{w})\}$ ➤ 2D window of pixel coordinates.
4      $h \leftarrow \text{HISTOGRAM}(I(\mathcal{W}))$ ➤ Compute graylevel histogram over window.
5      $med \leftarrow \text{MEDIAN}(h)$ ➤ Compute the median of the pixels.
6      $n_m \leftarrow \sum_{i=0}^{med} h[i]$ ➤ Number of pixels whose gray level is less than or equal to med.
7   **for** $x \leftarrow 0$ **to** $width - 1$ **do**
8         $I'(x, y) \leftarrow med$ ➤ Set output pixel to the median value.
9   **for** $y' \leftarrow -\tilde{w}$ **to** $\tilde{w}$ **do**
10           $v \leftarrow I(x - \tilde{w}, y + y')$ ➤ Update histogram by removing pixels along left edge.
11           $h[v] \leftarrow_- 1$;   **if** $v \le med$ **then** $n_m \leftarrow_- 1$
12           $v \leftarrow I(x + \tilde{w} + 1, y + y')$ ➤ Update histogram by adding pixels along right edge.
13           $h[v] \leftarrow_+ 1$;   **if** $v \le med$ **then** $n_m \leftarrow_+ 1$
14   **while** $n_m < \lfloor w * w/2 \rfloor$ **do**   ➤ Update the median.
15           $med \leftarrow_+ 1$; $n_m \leftarrow_+ h[med]$
16   **while** $n_m > \lfloor w * w/2 \rfloor$ **do**
17           $n_m \leftarrow_- h[med]$; $med \leftarrow_- 1$
18 **return** I'

이미지의 각 픽셀에 대해 주변 픽셀의 중앙값을 계산하여 노이즈를 줄이는 기법

Salt and Pepper Noise ————
Median Filter 구현

```python
def median_filter(img, filter_size=(3, 3), stride=1):
    img_shape = np.shape(img)
    result_shape = tuple(np.int64((np.array(img_shape[:2]) - np.array(filter_size)) / stride) + 1) + (img_shape[2],)
    result = np.zeros(result_shape)

    for h in range(0, result_shape[0], stride):
        for w in range(0, result_shape[1], stride):
            for c in range(img_shape[2]):
                tmp = img[h:h + filter_size[0], w:w + filter_size[1], c].ravel()
                tmp = np.sort(tmp)
                result[h, w, c] = tmp[int(len(tmp) / 2)]

    return result
```

각 영상 픽셀에 대해 Median Filter 적용

Salt and Pepper Noise ───
Median Filter 최적화

1. 커널 사이즈 및 stride에 따른 PSNR 값 비교

```
1   stride 1, filter size : 3    37   stride 1, filter size : 5
2   k size : 3,  sigma = 1       38   k size : 5,  sigma = 1
3   PSNR : 26.865229883265556    39   PSNR : 26.938089508391954
4   stride 1, filter size : 3    40   stride 1, filter size : 5
5   k size : 3,  sigma = 4       41   k size : 5,  sigma = 4
6   PSNR : 26.914981703598002    42   PSNR : 26.96127622960564
7   stride 1, filter size : 3    43   stride 1, filter size : 5
8   k size : 3,  sigma = 7       44   k size : 5,  sigma = 7
9   PSNR : 26.916010196318577    45   PSNR : 26.9569790978835
10  stride 1, filter size : 3    46   stride 1, filter size : 5
11  k size : 5,  sigma = 1       47   k size : 7,  sigma = 1
12  PSNR : 27.092299276174035    48   PSNR : 26.947103701589366
13  stride 1, filter size : 3    49   stride 1, filter size : 5
14  k size : 5,  sigma = 4       50   k size : 7,  sigma = 4
15  PSNR : 27.161295722165132    51   PSNR : 26.909635207122978
16  stride 1, filter size : 3    52   stride 1, filter size : 5
17  k size : 5,  sigma = 7       53   k size : 7,  sigma = 7
18  PSNR : 27.148359033237472    54   PSNR : 26.88270836092078
19  stride 1, filter size : 3    55   stride 1, filter size : 7
20  k size : 7,  sigma = 1       56   k size : 3,  sigma = 1
21  PSNR : 27.12090264185008     57   PSNR : 26.645330842325325
22  stride 1, filter size : 3    58   stride 1, filter size : 7
23  k size : 7,  sigma = 4       59   k size : 3,  sigma = 4
24  PSNR : 27.10715949657424     60   PSNR : 26.619319798832745
25  stride 1, filter size : 3    61   stride 1, filter size : 7
26  k size : 7,  sigma = 7       62   k size : 3,  sigma = 7
27  PSNR : 27.054817893227217    63   PSNR : 26.61796068027901
28  stride 1, filter size : 5    64   stride 1, filter size : 7
29  k size : 3,  sigma = 1       65   k size : 5,  sigma = 1
30  PSNR : 26.860851454047612    66   PSNR : 26.670853406495745
31  stride 1, filter size : 5    67   stride 1, filter size : 7
32  k size : 3,  sigma = 4       68   k size : 5,  sigma = 4
33  PSNR : 26.8587488536939      69   PSNR : 26.625964355886303
34  stride 1, filter size : 5    70   stride 1, filter size : 7
35  k size : 3,  sigma = 7       71   k size : 5,  sigma = 7
36  PSNR : 26.85823743806598     72   PSNR : 26.620557366465725
```

커널 사이즈: 3, stride: 1일 때 가장 높은 PSNR 값 도출

2. 필터 적용 횟수에 따른 PSNR 값 비교

```
median filtering : 1

PSNR : 27.161295722165132


median filtering : 2

PSNR : 26.927550862087138


median filtering : 3

PSNR : 26.552685933516592


median filtering : 4

PSNR : 26.13068250021685


median filtering : 5

PSNR : 25.701710413533124
```

필터 한 번만 적용했을 때 가장 높은 PSNR 값 도출

White Gaussian Noise
Gaussian Filter

ALGORITHM 5.4 Create a 1D Gaussian kernel

$\text{CREATEGAUSSIANKERNEL}(\sigma)$

Input: floating-point standard deviation σ
Output: 1D Gaussian kernel (as an array with w elements)

1   $\tilde{w} \leftarrow \text{GETKERNELHALFWIDTH}(\sigma)$ ➤ Determine a reasonable halfwidth $\tilde{w}$, using, e.g., Algorithm 5.5.
2   $w \leftarrow 2\tilde{w} + 1$ ➤ Compute the (odd) width w from the halfwidth $\tilde{w}$.
3   $norm \leftarrow 0$ ➤ Initialize the normalization factor to zero.
4   **for** $i \leftarrow 0$ **to** $w - 1$ **do**   ➤ Construct the w-element kernel by sampling
5      $gauss[i] \leftarrow \exp(-(i - \tilde{w}) * (i - \tilde{w})/(2 * \sigma * \sigma))$ the continuous Gaussian function,
6      $norm \leftarrow_+ gauss[i]$ while keeping track of the normalization factor.
7   **for** $i \leftarrow 0$ **to** $w - 1$ **do**   ➤ Apply the normalization factor
8      $gauss[i] \leftarrow_/ norm$ to ensure that $\sum_{i=0}^{w-1} gauss[i] = 0$.
9   **return** $gauss$

이미지 중심과의 거리에 따라 가중치를 다르게 하여 영상 내 노이즈를 줄이는 기법

White Gaussian Noise
Gaussian Filter 구현

```python
def gaussian_kernel(k_size, sigma):
    size = k_size//2
    y, x = np.ogrid[-size:size+1, -size:size+1]
    filter = 1/(2*np.pi * (sigma**2)) * np.exp(-1 *(x**2 + y**2)/(2*(sigma**2)))
    sum = filter.sum()
    filter /= sum
    return filter


def padding(img, k_size):
    pad_size = k_size//2
    h, w, ch = img.shape

    res = np.zeros((h + (2*pad_size), w+(2*pad_size), ch), dtype=np.float)

    if pad_size == 0:
        res = img.copy()
    else:
        res[pad_size:-pad_size, pad_size:-pad_size] = img.copy()
    return res
```

```python
def gaussian_filtering(img, k_size=5,sigma=4):
    h, w, ch = img.shape
    filter = gaussian_kernel(k_size, sigma)
    pad_img = padding(img,k_size)
    filtered_img = np.zeros((h, w, ch), dtype=np.float32)

    for ch in range(0, ch):
        for i in range(h):
            for j in range(w):
                filtered_img[i, j, ch] = np.sum(filter * pad_img[i:i+k_size, j:j+k_size, ch])

    return filtered_img
```

White Gaussian Noise ———
Gaussian Filter 최적화

1. 커널 사이즈 및 표준 편차에 따른 PSNR 값 비교


```
1   stride 1, filter size : 3      37  stride 1, filter size : 5
2   k size : 3,  sigma = 1         38  k size : 5,  sigma = 1
3   PSNR : 26.865229883265556      39  PSNR : 26.938089508391954
4   stride 1, filter size : 3      40  stride 1, filter size : 5
5   k size : 3,  sigma = 4         41  k size : 5,  sigma = 4
6   PSNR : 26.914981703598002      42  PSNR : 26.96127622960564
7   stride 1, filter size : 3      43  stride 1, filter size : 5
8   k size : 3,  sigma = 7         44  k size : 5,  sigma = 7
9   PSNR : 26.916010196318577      45  PSNR : 26.9569790978835
10  stride 1, filter size : 3      46  stride 1, filter size : 5
11  k size : 5,  sigma = 1         47  k size : 7,  sigma = 1
12  PSNR : 27.092299276174035      48  PSNR : 26.947103701589366
13  stride 1, filter size : 3      49  stride 1, filter size : 5
14  k size : 5,  sigma = 4         50  k size : 7,  sigma = 4
15  PSNR : 27.161295722165132      51  PSNR : 26.909635207122978
16  stride 1, filter size : 3      52  stride 1, filter size : 5
17  k size : 5,  sigma = 7         53  k size : 7,  sigma = 7
18  PSNR : 27.148359033237472      54  PSNR : 26.88270836092078
19  stride 1, filter size : 3      55  stride 1, filter size : 7
20  k size : 7,  sigma = 1         56  k size : 3,  sigma = 1
21  PSNR : 27.12090264185008       57  PSNR : 26.645330842325325
22  stride 1, filter size : 3      58  stride 1, filter size : 7
23  k size : 7,  sigma = 4         59  k size : 3,  sigma = 4
24  PSNR : 27.10715949657424       60  PSNR : 26.619319798832745
25  stride 1, filter size : 3      61  stride 1, filter size : 7
26  k size : 7,  sigma = 7         62  k size : 3,  sigma = 7
27  PSNR : 27.054817893227217      63  PSNR : 26.61796068027901
28  stride 1, filter size : 5      64  stride 1, filter size : 7
29  k size : 3,  sigma = 1         65  k size : 5,  sigma = 1
30  PSNR : 26.860851454047612      66  PSNR : 26.670853406495745
31  stride 1, filter size : 5      67  stride 1, filter size : 7
32  k size : 3,  sigma = 4         68  k size : 5,  sigma = 4
33  PSNR : 26.8587488536939        69  PSNR : 26.625964355886303
34  stride 1, filter size : 5      70  stride 1, filter size : 7
35  k size : 3,  sigma = 7         71  k size : 5,  sigma = 7
36  PSNR : 26.85823743806598       72  PSNR : 26.620557366465725
```

커널 사이즈: 5, 표준 편차: 4일 때 가장 높은 PSNR 값 도출

2. 필터 적용 횟수에 따른 PSNR 값 비교


```
gaussian filtering : 1
PSNR : 27.161295722165132

gaussian filtering : 2
PSNR : 27.03794673695009

gaussian filtering : 3
PSNR : 26.7766655818144

gaussian filtering : 4
PSNR : 26.532586568433622

gaussian filtering : 5
PSNR : 26.30939786619438
```
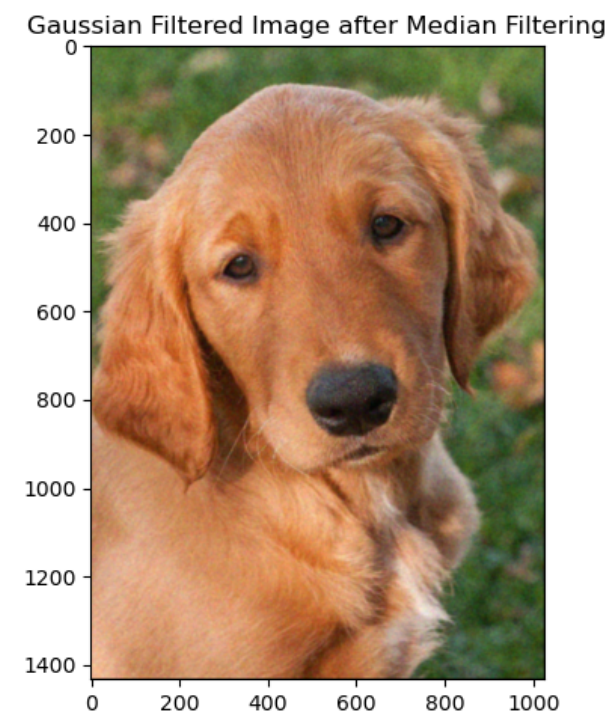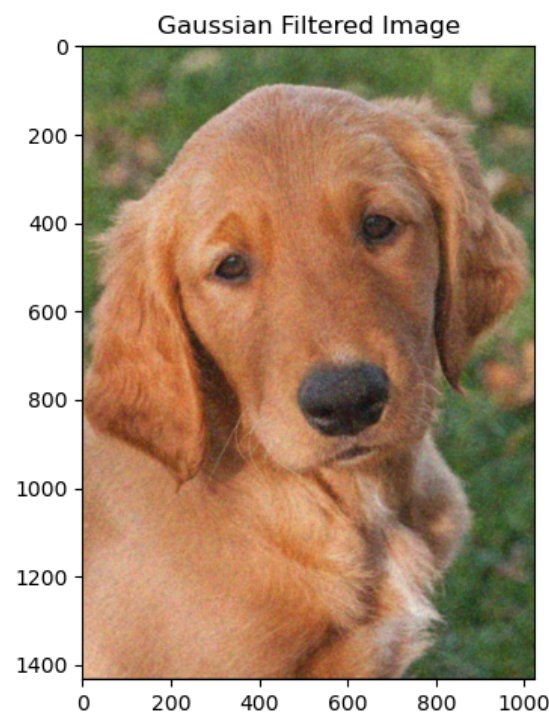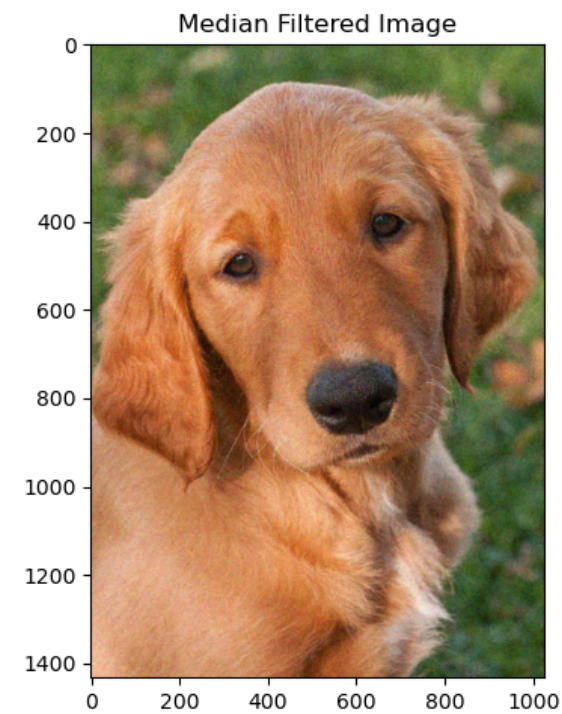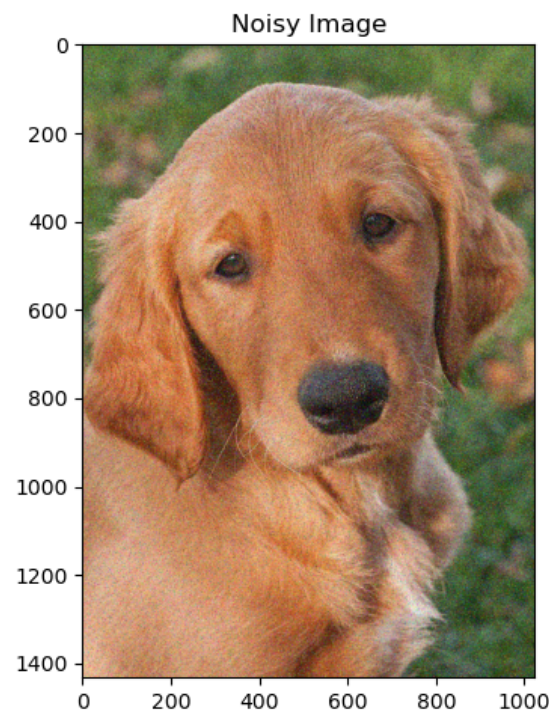
필터 한 번만 적용했을 때 가장 높은 PSNR 값 도출

Image Denoising
Filter 적용 순서에 따른 PSNR 비교

Noisy Image

Median Filtered Image

Gaussian Filtered Image

Gaussian Filtered Image after Median Filtering

이미지에 Median 필터 적용한 뒤 Gaussian 필터를 적용했을 때, 더 높은 PSNR 값 도출

노이즈 적용: 14.56
Median Filter: 25.84
Gaussian Filter: 24.69
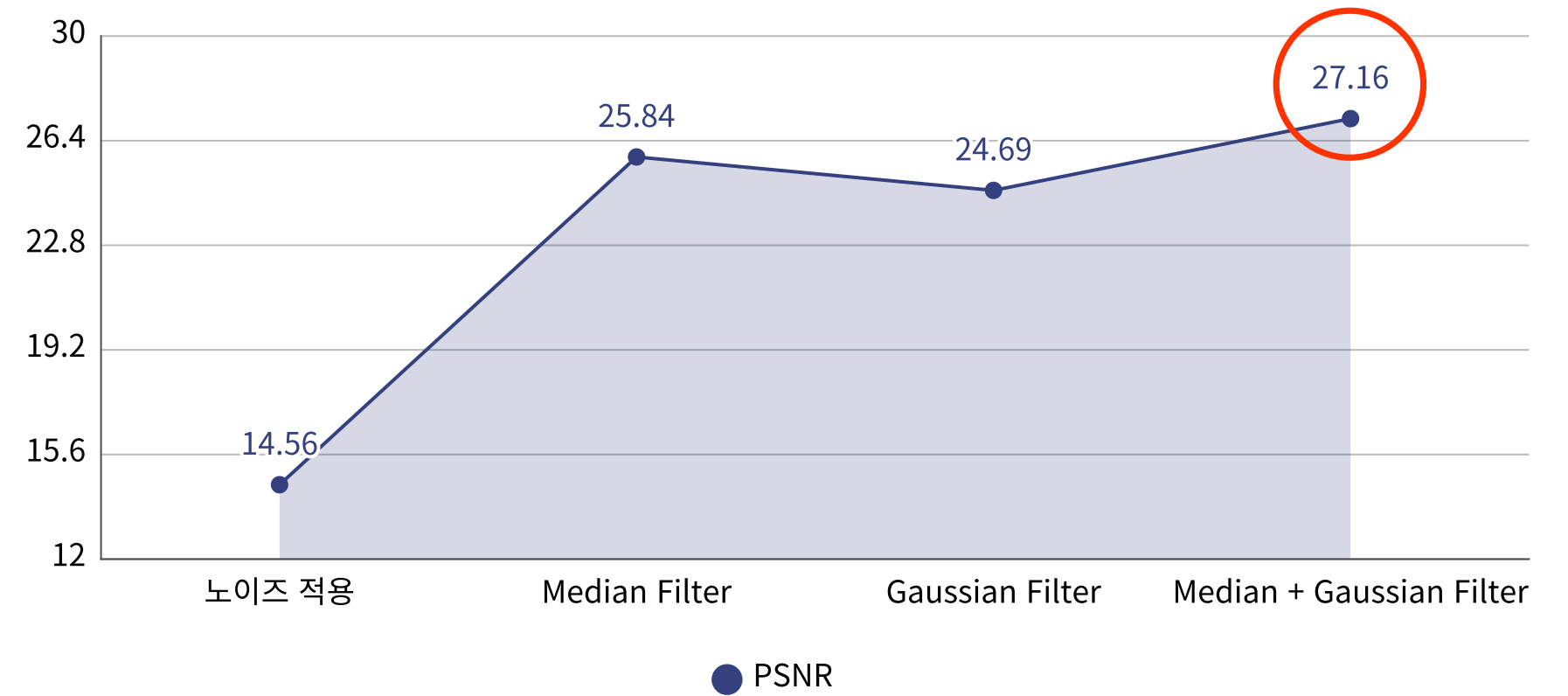Median + Gaussian Filter: 27.16

● PSNR

Image Denoising ——————
Bilateral Filter

Gaussian Filter 적용 시,

영상 엣지 부근에서 픽셀 값을 평탄하게 만드는 단점을 보완하기 위해 Bilateral Filter 구현

ALGORITHM 5.13 Perform bilateral filtering on an image

$\text{BILATERALFILTER}(I, \sigma_s, \sigma_r, n_{iter})$

Input: grayscale image I, standard deviations σ_s and σ_r of Gaussian spatial and range kernels, number n_{iter} of iterations

Output: bilateral-filtered image

1	**for** $k \leftarrow 1$ **to** n_{iter} **do**	For each iteration,
2	**for** $(x, y) \in I$ **do**	and for each pixel in the image,
3	$val \leftarrow 0$	initialize the value to zero,
4	$norm \leftarrow 0$	and the normalization factor to zero.
5	**for** $(\delta_x, \delta_y) \in \mathcal{W}$ **do**	For each pixel in a $\pm 2.5\sigma_s$ window,
6	$d_s^2 \leftarrow \delta_x * \delta_x + \delta_y * \delta_y$	compute squared spatial distance,
7	$d_r \leftarrow I(x, y) - I(x + \delta_x, y + \delta_y)$	and range difference
8	$w \leftarrow \exp(-d_s^2/(2 * \sigma_s^2)) * \exp(-(d_r * d_r)/(2 * \sigma_r^2))$	to compute weight.
9	$val \leftarrow_+ w * I(x + \delta_x, y + \delta_y)$	Accumulate weighted sum
10	$norm \leftarrow_+ w$	and update normalization factor.
11	$I'(x, y) \leftarrow val/norm$	Set output to normalized weighted sum.
12	$I \leftarrow I'$	Copy entire output image to input for next iteration.
13	**return** I'	

Image Denoising
Bilateral Filter 구현

```python
def bilateral_filter(noisy_img, k_size=5, sigma_space=4, sigma_intensity=0.2):
    h, w, ch = noisy_img.shape
    bilateral_noisy_img = np.zeros((h, w, ch))
    spatial_filter = gaussian_kernel(k_size, sigma_space)
    for c in range(ch):
        for i in range(h):
            for j in range(w):
                intensity_center = noisy_img[i, j, c]
                weighted_sum = 0.0
                normalization_factor = 0.0
                for m in range(-k_size//2, k_size//2 + 1):
                    for n in range(-k_size//2, k_size//2 + 1):
                        i_neighbor = i + m
                        j_neighbor = j + n

                        if 0 <= i_neighbor < h and 0 <= j_neighbor < w:
                            intensity_neighbor = noisy_img[i_neighbor, j_neighbor, c]
                            weight_intensity = np.exp(-(intensity_center - intensity_neighbor)**2 / (2 * sigma_intensity**2))
                            weight_spatial = spatial_filter[m + k_size//2, n + k_size//2]
                            weight = weight_intensity * weight_spatial
                            weighted_sum += intensity_neighbor * weight
                            normalization_factor += weight

                bilateral_noisy_img[i, j, c] = weighted_sum / normalization_factor

    return bilateral_noisy_img
```
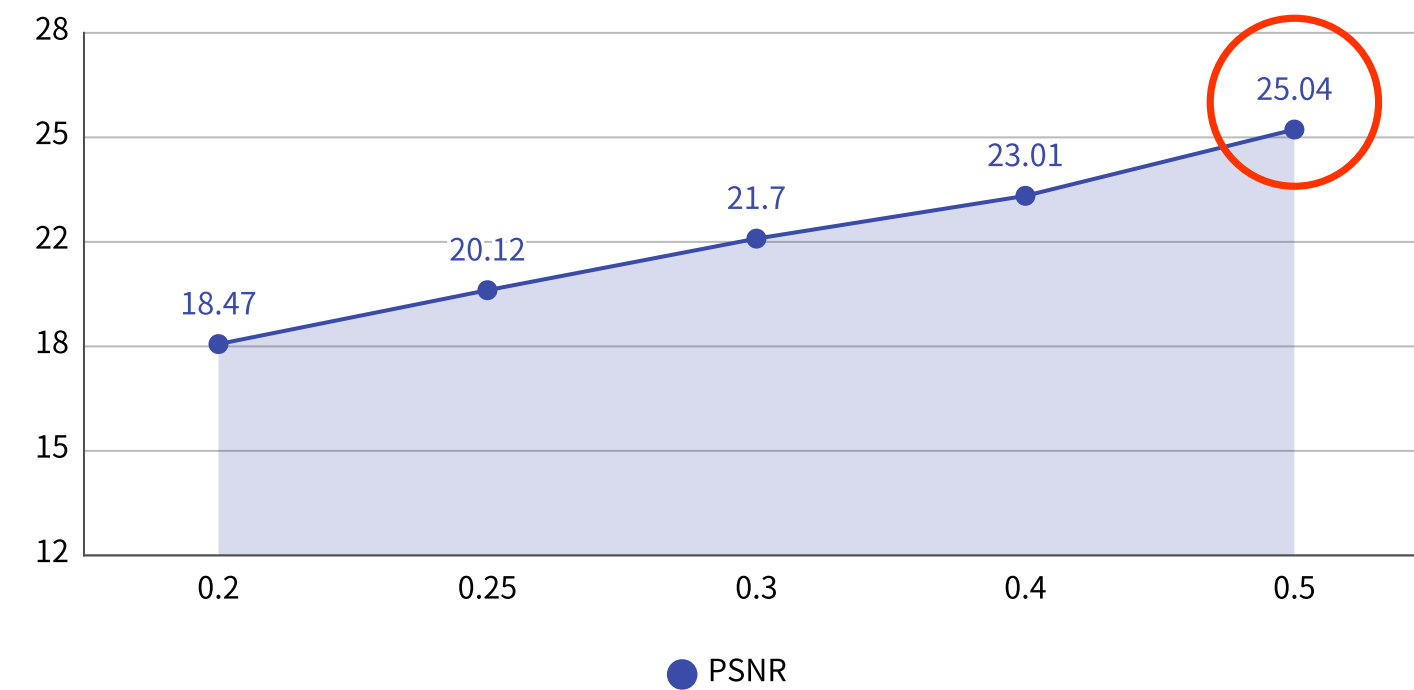
이미지 내 특정 픽셀과 주변 픽셀 간 강도 차이를 고려하여
가중치를 다르게 적용

Image Denoising

Bilateral Filter 최적화 I

- 표준 편차에 따른 PSNR 값 비교


```
sigma_intemsity=0.2 - PSNR: 18.465199371341477
sigma_intemsity=0.25 - PSNR: 20.11810428632533
sigma_intemsity=0.3 - PSNR: 21.69718677355458
sigma_intemsity=0.35 - PSNR: 23.00701856295732
sigma_intemsity=0.4 - PSNR: 23.966817433556663
sigma_intemsity=0.5 - PSNR: 25.038090386099
```

표준 편차 0.5일 때 가장 높은 PSNR 값 도출

Image Denoising
Bilateral Filter 최적화 II

- vectorization을 통한 실행 시간 단축

기존 코드: 기본 반복문 구조

```python
def bilateral_filter(noisy_img, k_size=5, sigma_space=4, sigma_intensity=0.2):
    h, w, ch = noisy_img.shape
    bilateral_noisy_img = np.zeros((h, w, ch))
    spatial_filter = gaussian_kernel(k_size, sigma_space)

    for c in range(ch):
        for i in range(h):
            for j in range(w):
                intensity_center = noisy_img[i, j, c]
                weighted_sum = 0.0
                normalization_factor = 0.0
                for m in range(-k_size//2, k_size//2 + 1):
                    for n in range(-k_size//2, k_size//2 + 1):
                        i_neighbor = i + m
                        j_neighbor = j + n

                        if 0 <= i_neighbor < h and 0 <= j_neighbor < w:
                            intensity_neighbor = noisy_img[i_neighbor, j_neighbor, c]
                            weight_intensity = np.exp(-(intensity_center - intensity_neighbor)**2 / (2 * sigma_intensity**2))
                            weight_spatial = spatial_filter[m + k_size//2, n + k_size//2]
                            weight = weight_intensity * weight_spatial
                            weighted_sum += intensity_neighbor * weight
                            normalization_factor += weight

                bilateral_noisy_img[i, j, c] = weighted_sum / normalization_factor

    return bilateral_noisy_img
```

최적화 코드: Numpy 배열 연산 사용

```python
def fff_bilateral_filter(noisy_img, k_size=5, sigma_space=4, sigma_intensity=0.2):
    h, w, ch = noisy_img.shape
    bilateral_noisy_img = np.zeros((h, w, ch))

    spatial_filter = gaussian_kernel(k_size, sigma_space)

    for c in range(ch):
        intensity_center = noisy_img[:, :, c]
        weighted_sum = np.zeros_like(intensity_center)
        normalization_factor = np.zeros_like(intensity_center)

        for m in range(-k_size//2, k_size//2 + 1):
            for n in range(-k_size//2, k_size//2 + 1):
                i_neighbors = np.clip(np.arange(h) + m, 0, h - 1)
                j_neighbors = np.clip(np.arange(w) + n, 0, w - 1)
                intensity_neighbors = noisy_img[i_neighbors, :, c][:, j_neighbors]
                weight_intensity = np.exp(-(intensity_center - intensity_neighbors)**2 / (2 * sigma_intensity**2))
                weight_spatial = spatial_filter[m + k_size//2, n + k_size//2]
                weighted_sum += intensity_neighbors * weight_intensity * weight_spatial
                normalization_factor += weight_intensity * weight_spatial

        bilateral_noisy_img[:, :, c] = weighted_sum / normalization_factor
    return bilateral_noisy_img
```

Image Denoising ───────────

Bilateral Filter 최적화 II

- 커널 사이즈 및 표준 편차에 따른 PSNR 비교


```
kernel size=5, sigma_space=3, sigma_intemsity=0.1 - PSNR: 15.482560680105301
kernel size=5, sigma_space=3, sigma_intemsity=0.2 - PSNR: 18.392135237237245
kernel size=5, sigma_space=3, sigma_intemsity=0.3 - PSNR: 21.585433732029156
kernel size=5, sigma_space=3, sigma_intemsity=0.4 - PSNR: 23.871470681241483
kernel size=5, sigma_space=3, sigma_intemsity=0.5 - PSNR: 24.974357477097954
kernel size=5, sigma_space=3, sigma_intemsity=0.6 - PSNR: 25.425883603286312
kernel size=5, sigma_space=3, sigma_intemsity=0.7 - PSNR: 25.593061943578054
kernel size=5, sigma_space=3, sigma_intemsity=0.8 - PSNR: 25.642755359154297
kernel size=5, sigma_space=3, sigma_intemsity=0.9 - PSNR: 25.64498497109403
kernel size=5, sigma_space=3, sigma_intemsity=1.0 - PSNR: 25.628996811033005
kernel size=5, sigma_space=4, sigma_intemsity=0.1 - PSNR: 15.509620617675209
kernel size=5, sigma_space=4, sigma_intemsity=0.2 - PSNR: 18.46346220024416
kernel size=5, sigma_space=4, sigma_intemsity=0.3 - PSNR: 21.693411693786757
kernel size=5, sigma_space=4, sigma_intemsity=0.4 - PSNR: 23.961350463343383
kernel size=5, sigma_space=4, sigma_intemsity=0.5 - PSNR: 25.032026330667687
kernel size=5, sigma_space=4, sigma_intemsity=0.6 - PSNR: 25.458961405099252
kernel size=5, sigma_space=4, sigma_intemsity=0.7 - PSNR: 25.609732151321513
kernel size=5, sigma_space=4, sigma_intemsity=0.8 - PSNR: 25.64859932744764
kernel size=5, sigma_space=4, sigma_intemsity=0.9 - PSNR: 25.643513499767497
kernel size=5, sigma_space=4, sigma_intemsity=1.0 - PSNR: 25.622419298357514
kernel size=5, sigma_space=5, sigma_intemsity=0.1 - PSNR: 15.52199519684902
kernel size=5, sigma_space=5, sigma_intemsity=0.2 - PSNR: 18.495943815090857
kernel size=5, sigma_space=5, sigma_intemsity=0.3 - PSNR: 21.741335996969905
kernel size=5, sigma_space=5, sigma_intemsity=0.4 - PSNR: 23.999221820794677
kernel size=5, sigma_space=5, sigma_intemsity=0.5 - PSNR: 25.0544917753176
```

커널 사이즈: 5, 공간 표준편차: 4, 강도 표준편차: 0.8 일 때, 가장 높은 PSNR 도출

차주 계획

W4 (11/14 – 20)

- Bilateral Filter 구현

- Bilateral Filter 최적화

W5 (11/21 – 27)

- Non-local Means Filter 구현

W6 (11/28 – 12/4)

- asdf