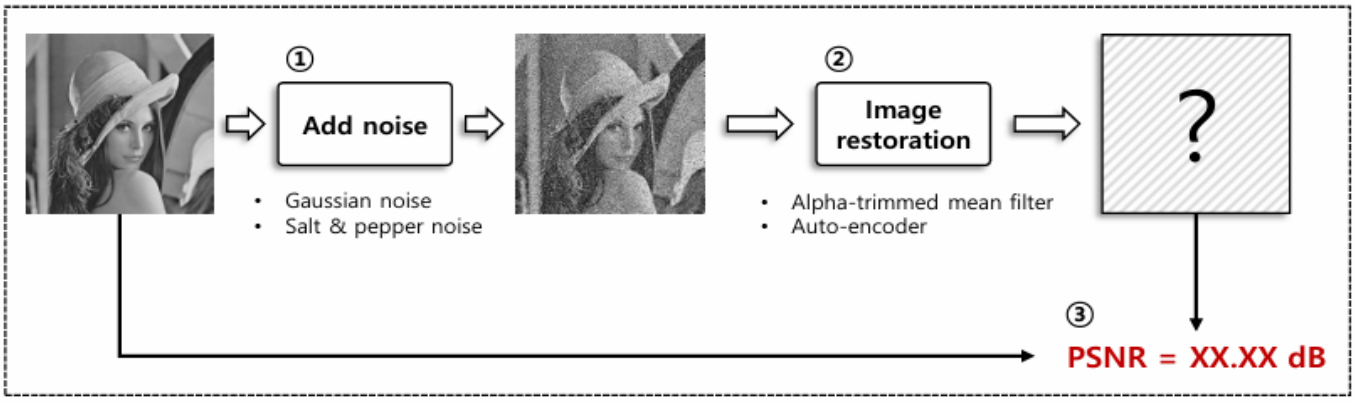
Computer Vision Assignment Report

Assignment 2 – Image Restoration

2019742071 – 이민규

1. 과제 개요

1. 잡음을 생성하여 입력 이미지에 추가
   – Gaussian noise
   – Salt & pepper noise
2. 잡음이 추가된 이미지에 대해 잡음 제거 알고리즘 수행
   – Alpha-trimmed mean filter
   – Auto-encoder
3. PNSR을 측정하여 잡음 제거 성능 비교



1. : 잡음을 생성하려 입력 이미지에 추가

- Gaussian noise (평균 0, 표준편차 10)

- Salt & Pepper noise (노이즈 생성비율 20%)

2. : 잡음이 추가된 이미지에 대해 잡음 제거 알고리즘 수행

- Alpha-trimmed mean filter (3 X 3 필터, Alpha 0.2, 5 X 5 필터, Alpha 0.4)

- Auto-encoder (32채널, 64채널)

3. : PSNR을 측정하여 잡음 제거 성능 비교

입력 이미지 : 256 X 256 Grayscale image, 3000 Train set + 100 Test set

2. 과제 수행 방법

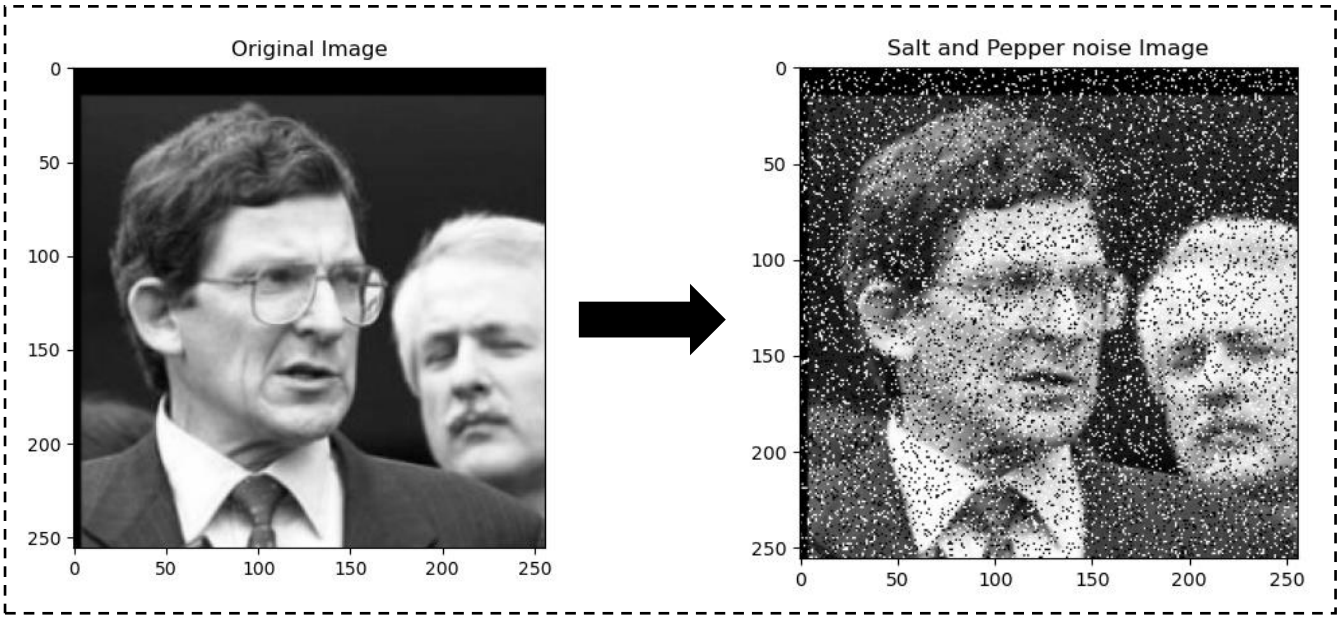
1. : 노이즈 추가

1-1: salt and pepper noise

```python
# salt and pepper noise를 추가하는 함수 정의
def snp_noise_maker_v3_optimized(gray_img, noise_ratio=0.2):
    noisy_img = np.copy(gray_img)
    mask = np.random.rand(*gray_img.shape) < noise_ratio
    # 비율에 따라 랜덤으로 노이즈를 생성할 좌표 list를 생성하여 이를 mask로 활용
    noise = np.random.choice([0, 255], size=gray_img.shape, p=[0.5, 0.5])
    # salt와 pepper의 비율은 1:1로 설정
    noisy_img[mask] = noise[mask]
    return noisy_img
```

< 그림1. 노이즈 추가함수 snp_noise_maker_v3_optimized >

노이즈 추가함수는 rand함수를 이용해 이미지 크기만큼의 마스크를 생성하고, noise ratio에 따라 노이즈를 생성할 위치를 정한다. Salt와 pepper의 비율은 1 : 1이다.



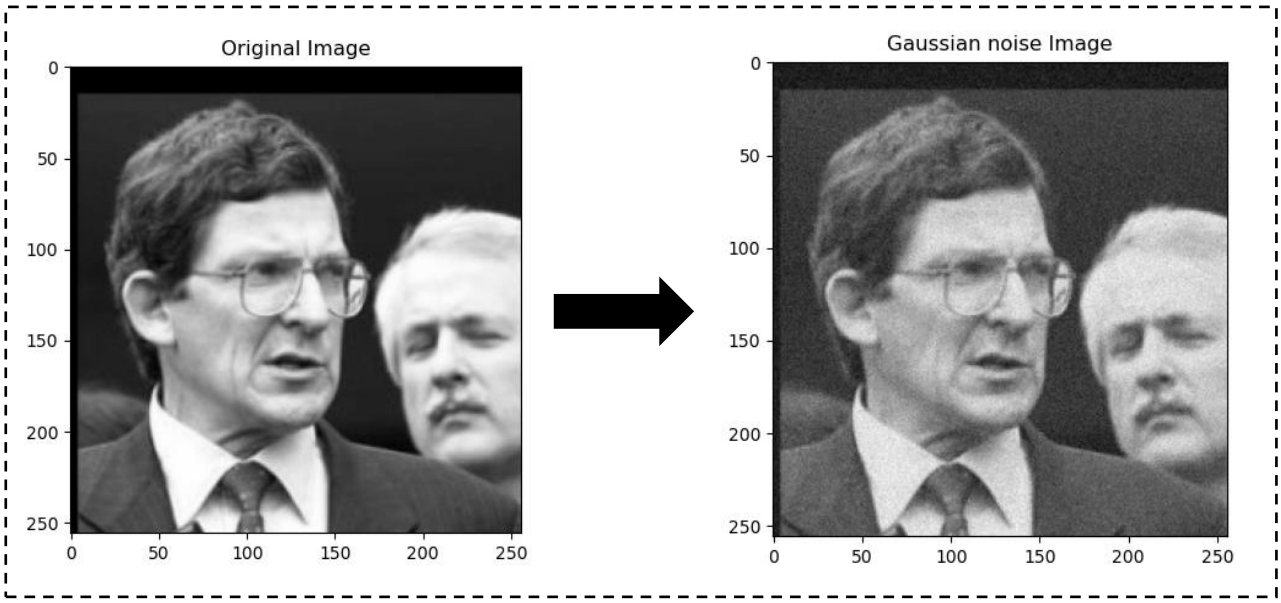
< 그림2. Salt and Pepper noise 추가 결과 >

1-2: Gaussian noise

```python
#가우시안 노이즈를 추가하는 함수 정의
def Gaussian_noise_maker_optimized(gray_img, sigma=10):
    noise = np.random.normal(0, sigma, gray_img.shape)
    # numpy 내장함수 random.normal은 가우시안 노이즈와 동일한 역할 수행
    imgdata_Gaussian_noise = gray_img + noise
    return imgdata_Gaussian_noise
```

< 그림3. 노이즈 추가함수 Gaussian_noise_maker_optimized >

Numpy 내장함수중, random.normal은 정규분포를 따르는 임의의 실수를 생성한다. 지정한 파라미터로 가우시안 잡음을 생성가능하다.



< 그림4. Gaussian noise 추가결과 >

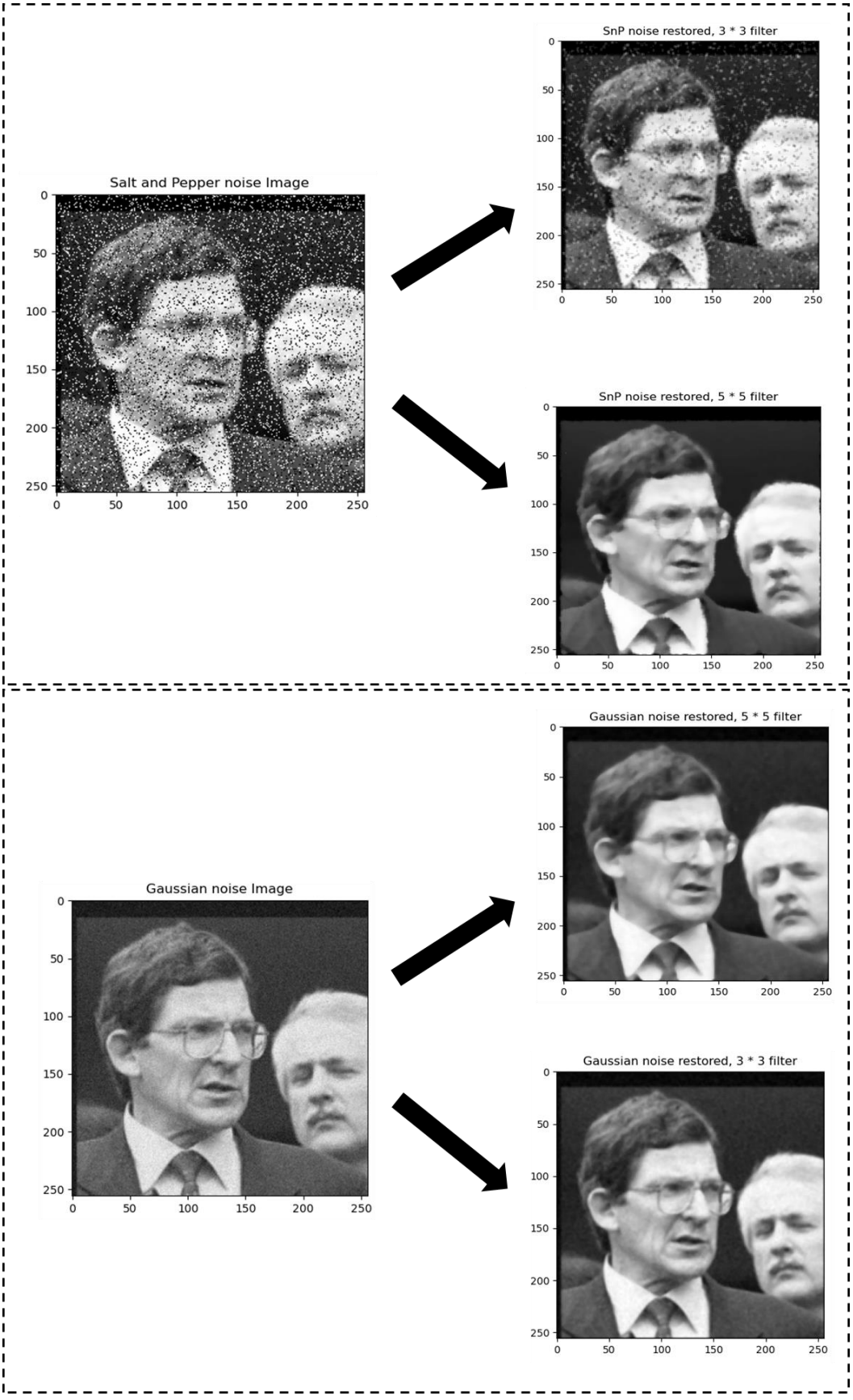
2. : 복원

2-1: Alpha-Trimmed-Mean-Filter 적용

```python
# alpha trimmed mean filter를 적용하는 함수 정의
def alpha_trimmed_mean_filter(image, filter_size, alpha):
    # window에서 몇개의 픽셀을 제거할 것인가?
    trim_amount = int((filter_size * filter_size) * alpha)

    # 사용하는 필터의 크기에 따라 zero padding을 적용
    pad_size = filter_size // 2
    padded_image = np.pad(image, pad_size, mode='constant', constant_values=0)

    filtered_image = np.zeros_like(image)
    height, width = image.shape

    for i in range(height):
        for j in range(width):
            # 각 픽셀을 중심으로 필터사이즈 만큼의 이웃픽셀을 추출하여 1-dimension으로 변경.
            window = padded_image[i:i + filter_size, j:j + filter_size].flatten()
            # 뽑아낸 배열을 크기순서대로 정렬
            window_sorted = np.sort(window)
            # 좌우 극단값을 제거하고 남은 평균을 지정한 픽셀의 값으로 결정
            trimmed_window = window_sorted[trim_amount:-trim_amount]
            filtered_image[i, j] = np.mean(trimmed_window)

    return filtered_image
```

< 그림5. alpha_trimmed_mean_filter 함수정의 >

선택한 좌표로부터 주어진 크기의 window 내의 모든 픽셀값을 뽑아내 순서대로 정렬한다. 기존 중간값 필터는 이중 중간의 값을 취하여 새 픽셀의 값으로 결정하겠지만 alpha trimmed mean filter는 alpha값에 따라 중간값 주변의 값을 평균으로 하여 새로운 픽셀값을 정하게된다. 이와 같은 방식은 salt and pepper 잡음과 같이 커다란 이상치를 무시할 수 있게 한다.



< 그림6. alpha_trimmed_mean_filter 적용결과 >

2-2: Auto Encoder 적용

```
input = layers.Input(shape=(256, 256, 1))

# Encoder
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(input)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decoder
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)

autoencoder_32c = Model(input, x)
autoencoder_32c.compile(optimizer="adam", loss="binary_crossentropy")
autoencoder_32c.summary()
```

< 그림7. Auto Encoder 모델 >

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 1)]     0

 conv2d_3 (Conv2D)           (None, 256, 256, 32)      320

 max_pooling2d_2 (MaxPoolin  (None, 128, 128, 32)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 128, 128, 32)      9248

 max_pooling2d_3 (MaxPoolin  (None, 64, 64, 32)        0
 g2D)

 conv2d_transpose_2 (Conv2D  (None, 128, 128, 32)      9248
 Transpose)

 conv2d_transpose_3 (Conv2D  (None, 256, 256, 32)      9248
 Transpose)

 conv2d_5 (Conv2D)           (None, 256, 256, 1)       289

=================================================================
Total params: 28353 (110.75 KB)
Trainable params: 28353 (110.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

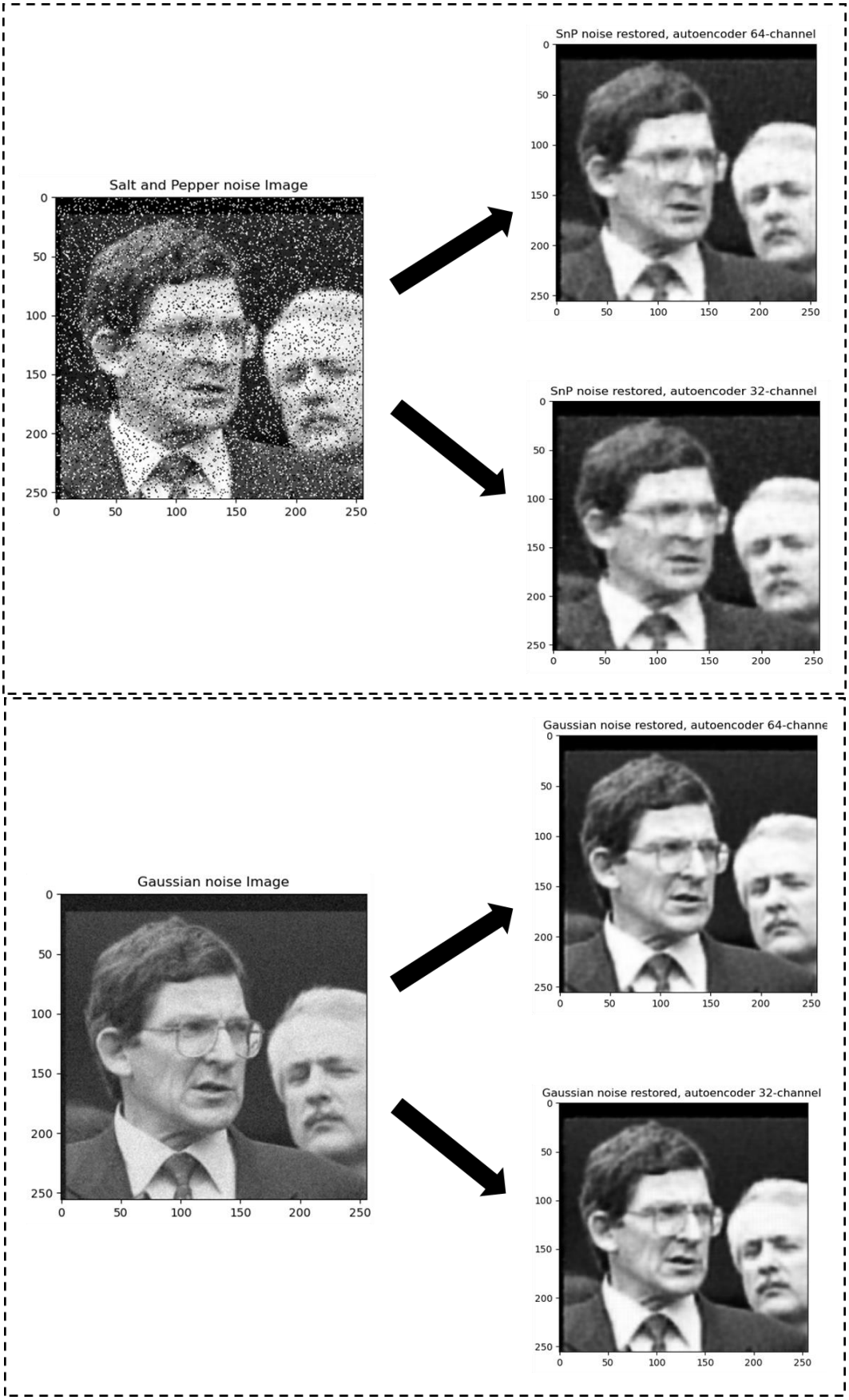
< 그림8. Auto Encoder 모델 Summary >

Auto Encoder는 입력데이터의 특징을 추출하고, 이를 토대로 원본을 재구성 하는 것을 목표로 하는 신경망이다. 모델의 구조를 확인하면 이미지의 특징을 파악하는 CNN모델로 이루어져 있으며 점차 그 차원이 작아진다. 차원이 작아진다는 것은 축소된 이미지의 한 픽셀이 저장하는 원본이미지에서의 데이터의 크기가 증가한다는 것을 의미한다. 또한 사용하는 필터의 개수에 따라서 여러 개의 특징을 추출할 수 있다. 특징은 저차원적인 가로선, 세로선, 코너등의 특징부터 고차원적인 특정한 중요요소의 위치가 있는데 이때, 어떤 특징을 추출할지는 모델의 학습과정에서 결정된다. 추출한 특징을 기반으로 원본이미지를 다시 복원하도록 훈련함으로서 오토인코더는 노이즈가 포함된 이미지를 원본으로 복원할 수 있게된다.

- **모델 추가 제원 :** Epoch 10, ADAM optimizer 사용

32채널 모델 평균 학습시간 약 10초 per 1 Epoch (CPU 사용시 60초)

64채널 모델 평균 학습시간 약 25초 per 1 Epoch (CPU 사용시 90초)



< 그림9. Auto Encoder 적용결과 >

3. : 통계적특성 도출 및 결과출력

```python
# PSNR을 계산하는 함수 정의
def psnr(img1, img2):
    mse = np.mean( (img1 - img2) ** 2 ) #MSE 구하는 코드

    #MSE가 0이면 100으로 정의
    if mse == 0:
        return 100

    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

# 이미지 배열의 PSNR의 평균과 분산을 계산하는 함수 정의
def calc_psnr(original_images, restored_images):
    psnr_values = []

    for i in range(len(original_images)):
        psnr_value = psnr(original_images[i], restored_images[i])
        psnr_values.append(psnr_value)

    average_psnr = np.mean(psnr_values)
    variance_psnr = np.var(psnr_values)
    return average_psnr, variance_psnr
```

< 그림10. PSNR의 평균과 분산을 구하는 함수 정의 >

PSNR을 정의하려면 먼저 MSE를 정의해야한다. MSE는 각 픽셀별 값과 원본 픽셀별 값을 비교하여 제곱하고 평균을 내어 계산한다. MSE가 작을수록 원본과 비슷하다고 판단할 수 있다.

MSE 가 작을수록 PSNR은 높아진다. 따라서 PSNR이 높을수록 원본과 가깝다고 판단 할 수 있다.

Calc_psnr 함수는 100장의 복원데이터셋에 대하여 구한 psnr의 평균과 분산을 구한다.

추가적으로 복원한 이미지를 imshow 함수로 출력하였다.

SnP, Alpha trimmed filter, 3 X 3	(31.634320440096143, 0.24780205626222226)
SnP, Alpha trimmed filter, 5 X 5	(36.15841262944738, 1.94155206529383)
Gaussian, Alpha trimmed filter, 3 X 3	(32.32897592837245, 2.3330034867248015)
Gaussian, Alpha trimmed filter, 5 X 5	(32.945557667468776, 3.260892406429426)
SnP, Auto Encoder, 32 channels	(31.458176778315252, 0.5205141566768053)
SnP, Auto Encoder, 64 channels	(32.538785797749945, 0.6135773843762758)
Gaussian, Auto Encoder, 32 channels	(33.544488475158275, 1.1160949491837335)
Gaussian, Auto Encoder, 64 channels	(34.34213365059265, 1.0901227312288733)
SnP noise Image	(35.11938400001762, 0.3088809481882852)
Gaussian noise Image	(28.13189557164667, 0.0006621731265782096)

< 그림11. 실행결과 >

3. 결과분석

	Alpha-trimmed mean filtering		Autoencoder	
Parameter setting	Filter 3x3, α = 0.2	Filter 5x5, α = 0.4	32 channels	64 channels
Gaussian noise	**32.32 dB**	**32.94 dB**	**33.54 dB**	**34.34 dB**
Salt & pepper noise	**31.64 dB**	**36.15 dB**	**31.46 dB**	**32.54 dB**

비션형필터와 딥러닝방식의 복원방법을 모두 사용해 가우시안 노이즈와 salt and pepper 노이즈를 복원해 보았다. 육안상으로는 노이즈가 추가된 이미지에 비해서는 확실히 두 방법 모두 노이즈가 제거된 개선점을 확인할 수 있었다. 흥미로운 점은 Salt and Pepper noise 같은 경우는 PSNR의 평균을 계산했을 때, 35dB로 출력되었는데 필터나 오토인코더로 복원을 시도한 이미지의 PSNR값은 그에 미치지 못했다는 점이다.

실제로 PSNR은 정량적이지만 완벽한 지표는 아닌데, 이는 단지 이미지의 픽셀이 원본과 얼마나 비슷한지만 판단하며, PSNR이 높게 출력이 된다고 하더라도 사람이 주관적으로 판단하는 화질과는 그 차이가 있을 수 있다.

위 실행결과에서 출력된 이미지를 확인해보면 PSNR은 낮아졌지만 복원된 이미지 자체는 어느정도 노이즈가 제거되어 주관적 화질이 개선된 모습을 보여주고 있다.

수치적 에러가 아닌 인간의 시각적 화질차이를 평가하는 방법인 SSIM(Structural Similarity Index Map)을 사용하여 평가하면 다른 결과가 나왔을 가능성이 있을 것으로 예상한다.

* 개발환경 : Anaconda & VScode

conda version : 23.7.4

conda-build version : 3.26.1

python version : 3.11.5.final.0

```
Sat Apr 27 20:50:19 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 552.22                 Driver Version: 552.22         CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                     TCC/WDDM  | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce GTX 1050 Ti   WDDM  |   00000000:01:00.0  On |                  N/A |
| 45%   34C    P8             N/A /   75W |     699MiB /   4096MiB |      2%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

4. 고찰

노이즈가 추가된 이미지에 비선형필터와 오토인코더를 적용하여 복원을 시도하였다. 이미지는 다양한 이유로 손상될 수 있으며 그 이유를 완벽하게 수학적으로 모델링 하는 것은 불가능하다. 따라서 복원을 시도하는 사용자가 노이즈가 발생한 이유와 노이즈의 특성을 가정하고 확실한 기준을 가지고 복원을 진행한다. 그와 같은 기준을 가지고 복원에 사용할 방법을 선택한다고 하더라도 최적의 복원을 위한 파라미터를 적절하게 설정해 줄 필요성이 있음을 확인하였다.

컴퓨터비전 영역에서 인공지능의 발전은 매우 큰 영향을 미쳤을 것으로 생각한다. 이미지에서 어떠한 특징을 뽑는다고 하면 그 특징을 추출하기 위한 가정과 알고리즘을 기존에는 사람이 모두 선택하였다. 예를 들어 이미지에 미분을 적용하여 그래디언트를 찾아 임계값과 비교하는 경계선 검출의 예를 생각해보면 임계값의 결정은 온전히 사용자가 결정하는 수치였다. 인공지능의 모델의 강력한점은 인간의 직관과 경험에 의존하던 부분을 인공지능으로 대신할 수 있게 되었다는 점이라고 생각한다. 오토인코더에서 사용한 필터의 의미는 필터가 특징을 추출하여 피쳐맵을 만든다는 것이고 무슨 특징을 추출할지는 모델이 결정한다는 것이다. 이러한 딥러닝 모델의 특성이 이미지 특성을 추출하는 것에 있어서 가능성을 열어준다.

이번 구현에서는 PSNR을 사용하여 복원결과를 평가했지만 SSIM(Structural Similarity Index Map)을 사용하여 좀 더 사람의 시각에 가까운 평가지표를 사용해 평가하거나 다양한 파라미터를 사용해보며 복원을 시도하면 더 좋은 결과를 낼 수 있으리라 기대한다.