

다양한 필터링 기법을 이용한 디지털 영상 처리 방법 분석

Analysis of Digital Image Processing Methods Using Various Filtering Techniques

유호영¹ · 이가람²

You, Ho Yeong · Lee, Garam

Abstract

This research utilizes Python language to experiment with and compare various image filtering techniques in order to improve image quality. In addition to the traditional smoothing and sharpening filtering techniques in the spatial domain, the study proposes algorithms for more advanced filtering techniques. The research also aims to analyze filtering in the frequency domain as well. This enhances the ability to improve image quality in various situations and can be applied to actual image processing. This type of research is essential in an era where image and video data are becoming increasingly important and is expected to become even more important in the future. This research opens new horizons for research and application in the field of computer vision and image processing.

Keywords : Spatial filter, Frequency filter, Salt and Pepper, Python

초 록

이 연구는 Python 언어를 활용하여 이미지 필터링 기법을 실험하고 비교 분석하여 이미지 품질을 개선하는 방법을 연구하며, 공간 영역의 기존 Smoothing과 Sharpening 필터링 기법과 더불어 더욱 향상된 기법의 알고리즘을 제시한다. 또한 공간 영역뿐만 아니라 주파수 영역에서의 필터링도 분석하고자 한다. 이를 통해 다양한 상황에서 이미지 품질을 개선할 수 있는 능력을 키울 수 있으며, 실제 이미지 처리에도 적용 가능하다. 이러한 연구는 이미지 및 영상 데이터가 증가하고 중요성이 대두되는 시대에 필수적인 요소로 간주되며, 미래에는 더욱 중요성이 증대될 것으로 예상된다. 이를 통해 컴퓨터 비전 및 이미지 처리 분야에서의 연구와 응용에 새로운 지평을 열 수 있을 것으로 기대된다.

핵심어 : 공간 영역, 주파수 영역, 필터링, 이미지 품질, 파이썬

Github link : <https://github.com/leeolivine/PBML-assignment/>

1 에너지자원공학과 4학년

2 지구자원시스템공학과 3학년

1. 서 론

최근 몇 년 동안 디지털 이미지와 영상처리 기술은 급속도로 발전하고 있으며, 이러한 발전은 다양한 산업 분야에서 영상 품질을 향상하기 위한 수요 증가와 관련이 있다. 디지털 이미지와 영상 데이터는 일상생활에서 점점 중요한 역할을 차지하게 되었고, 고품질의 영상을 생성하고 개선하는 것은 많은 응용 분야에서 중요한 연구 주제가 되어왔다. 이러한 맥락에서 이미지 영상 필터링은 컴퓨터 비전 및 이미지 처리 분야의 핵심적인 과제로 주목받고 있다. 이미지 영상 필터링은 영상에 존재하는 잡음을 제거하거나, 선명도를 개선하는 데 사용되는 처리 기법을 포함한다. 이 논문에서는 Python을 이용하여 다양한 이미지 필터링 기법을 실험하고 비교 분석하는 작업을 수행한다. 먼저, 영상에 Salt & Pepper Noise를 추가하여 잡음이 있는 이미지를 생성한다. 그런 다음, 원본 영상과 잡음이 추가된 영상의 히스토그램을 그려 두 영상의 분포를 비교한다.

본 연구에서는 공간영역에서 두 가지 범주의 필터링 기법을 적용하고 분석한다. 첫 번째 범주는 Smoothing 필터로, Mean(Moving average), Gaussian, Median 등의 기법을 포함한다. 이러한 필터는 잡음을 제거하여 영상의 전반적인 품질을 개선하는 데 사용된다. 두 번째 범주는 Sharpening 필터로, Laplacian, Sobel 등의 기법을 포함한다. 이러한 필터는 영상의 선명도를 향상하고 경계를 찾기 위해 사용된다.

또한 이 연구에서는 주파수 영역에서 영상을 처리하는 방법을 적용한다. 푸리에 변환을 이용하여 영상을 주파수 영역으로 변환한 후, low-pass 필터와 high-pass 필터 등을 각각 적용한다. 이를 통해 필터링 효과와 주파수 영역에서의 영상처리 방법을 분석하고, 이러한 기법이 영상 품질에 어떤 영향을 미치는지 평가한다. 주파수 영역에서의 필터링은 공간영역에서의 필터링과는 다른 관점에서 영상 품질을 향상하는 방법을 제공한다.

이를 토대로 공간영역에서의 Smoothing 필터와 주파수 영역에서의 Low pass 필터를 비교하는 등 공간영역의 필터링과 주파수 영역의 필터링을 비교 분석하는 과정을 거칠 것이다. 이를 통해 어떤 영역에서 영상처리가 우리의 목적에 맞게 잘 수행되는지 알아보려고 한다. 본 연구를 통해 이미지 필터링 기법의 원리와 적용 방법에 대한 깊은 이해를 얻을 것이다. 이를 바탕으로 다양한 상황에서 영상 품질을 개선할 수 있는 능력을 키울 수 있을 것으로 기대된다. 또한, 이 연구에서 제시한 방법들은 실제 이미지 처리 등에도 적용할 수 있어, 이 분야의 연구와 기술 발전에 이바지할 수 있을 것으로 예상된다.

2. 연구 방법

2.1 원본 데이터

본 논문에서 사용한 원본 데이터는 2023년 3월 30일 ‘파이썬기반기계학습’ 시간 서울 어린이대공원에서 야외 수업 시 찍은 사진이다. 사진에 나와 있는 건물은 서울 어린이대공원의 전망대인 ‘팔각당’이다. Pixel의 수는 2907 x 2907이며 2.8MB 크기의 사진이다. 여러 사진 중에서 이 팔각당의 사진을 원본 데이터로 설정한 이유는 찍은 사진 중에서 경계에 관한 판단과 결과 비교가 쉬울 것으로 예측되었기 때문이다. 또한, 사진 내에 색이 다양해 흑백으로 처리할 때도 여러 스케일에 각각의 값들이 분포해 있을 거라 생각해 영상처리를 할 때 다양한 상황이 나올 수 있을 것으로 기대되어 선택하였다.

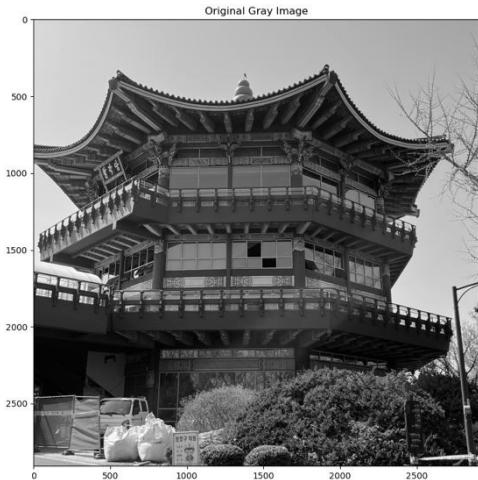


Fig 1. Raw Data

2.2 사용 라이브러리

2.2.1 OpenCV

본 연구에서 주로 다루는 라이브러리는 OpenCV이다. OpenCV는 Open Source Computer Vision의 약자로 영상처리 시 사용할 수 있는 open source library이다. 컴퓨터가 마치 사람의 눈처럼 영상을 인식할 수 있게 처리해주는 역할을 수행하며, 공장에서 제품을 검사하거나 의료 영상처리 및 보정과 판단, CCTV 영상, 로보틱스 등 다양한 분야에서 활용되고 있다. 또한, OpenCV는 실시간 처리에 가장 중점을 두도록 설계가 되었기에 빠른 속도와 효율성의 강점이다.

OpenCV의 기본 구조는 다음과 같다.

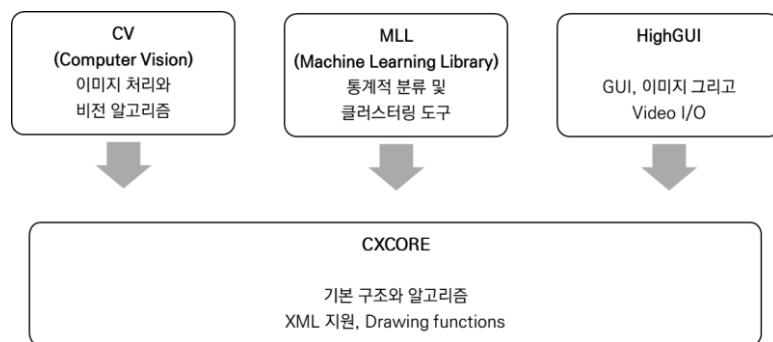


Fig 2. OpenCV의 기본구조

2.2.2 NumPy

NumPy는 Python에서 수치 계산을 위한 라이브러리이다. 다차원 배열 객체와 이러한 배열을 다루는 다양한 함수와 method를 제공하여 수치 계산을 보다 효율적으로 처리할 수 있도록 도와준다.

NumPy의 핵심 기능은 ndarray 객체이다. ndarray는 다차원 배열(N-dimensional array)의 약어로, 동일한 자료형을 가지는 원소들이 격자판 형태로 구성된 배열이다. 이러한 다차원 배열을 이용하여 벡터, 행렬, 그리고 다양한 수학적인 객체를 표현할 수 있다.

또한, 빠르고 효율적인 수치 계산을 위한 다양한 함수와 메서드를 제공한다. 이러한 함수와 method는 선형대수, 통계, 푸리에 변환, 난수 생성 등 다양한 분야에서 활용된다. 더불어 다른 파이썬 패키지와 함께 사용할 수 있는 유용한 도구이다. 예를 들어, NumPy 배열을 다루는 기능을 제공하는 pandas, 그리고 시각화 라이브러리인 matplotlib과 함께 사용하면 데이터 분석 및 시각화에 매우 유용하다. 게다가 오픈소스 라이브러리로, 파이썬 커뮤니티에서 널리 사용되고 있다. 많은 수의 과학, 엔지니어링, 경제학, 금융 등의 분야에서 NumPy를 활용하여 수치 계산 문제를 해결하고 있다.

이러한 이유들로 NumPy 라이브러리를 사용하였다.

2.2.3 Matplotlib

Matplotlib는 Python에서 데이터 시각화를 위한 라이브러리이다. 다양한 그래프, 플롯, 차트 등을 생성할 수 있도록 해주는 다양한 함수와 클래스를 제공한다. 또한, 다양한 형식의 그래프를 지원한다. 예를 들어, 라인 플롯, 바 차트, 산점도, 등고선 플롯, 3D 플롯 등 다양한 형식의 그래프를 생성할 수 있다. 이러한 그래프를 생성하는 데 사용되는 함수와 클래스는 Matplotlib 패키지의 pyplot 서브 패키지에 포함되어 있다. 그래프의 다양한 요소를 사용자가 직접 설정할 수 있는 다양한 옵션들도 또한 제공한다. 예를 들어, 축 레이블, 축 범위, 그래프 제목, 선 스타일, 색상 등의 다양한 요소를 직접 설정할 수 있다. 더불어, Matplotlib는 여러 그래프를 하나의 그림에 표시하거나, 서로 다른 서브플롯에 그래프를 표시하는 다양한 레이아웃 기능도 제공한다.

Matplotlib은 NumPy와 동일하게 파이썬 커뮤니티에서 널리 사용되는 오픈소스 라이브러리이다. 데이터 시각화를 위한 다른 라이브러리와 함께 사용되어, 데이터 분석 및 시각화 분야에서 광범위하게 활용되고 있다. 그렇기에 Matplotlib을 사용하였다.

2.3 사용 소프트웨어

본 연구에서 사용할 소프트웨어는 Visual Studio Code(이하 VS Code)이다. VS Code는 Microsoft에서 개발한 오픈소스 코드 편집기이다. VS Code는 다양한 프로그래밍 언어와 플랫폼에 대한 지원을 제공하며, 다양한 확장 기능을 통해 사용자의 필요에 따라 기능을 추가할 수 있다. 여러 장점들이 존재해 이 소프트웨어를 사용하였다.

첫 번째, VS Code의 주요 특징 중 하나는 강력한 코드 편집 기능이다. IntelliSense라는 기능을 통해 코드 작성 중 자동 완성, 문법 강조, 코드 추적 등의 기능을 제공한다. 또한, 디버깅, 테스트, 버전 관리, 작업 관리 등 다양한 기능을 통합하여 제공한다.

두 번째, 사용자가 필요에 따라 다양한 확장 기능을 설치하여 기능을 추가할 수 있다. 이러한 확장 기능은 VS Code에서 제공하는 기본 기능을 보완하며, 다양한 언어나 프레임워크, 빌드 도구, 테스트 프레임워크, 디버깅 도구 등을 지원한다.

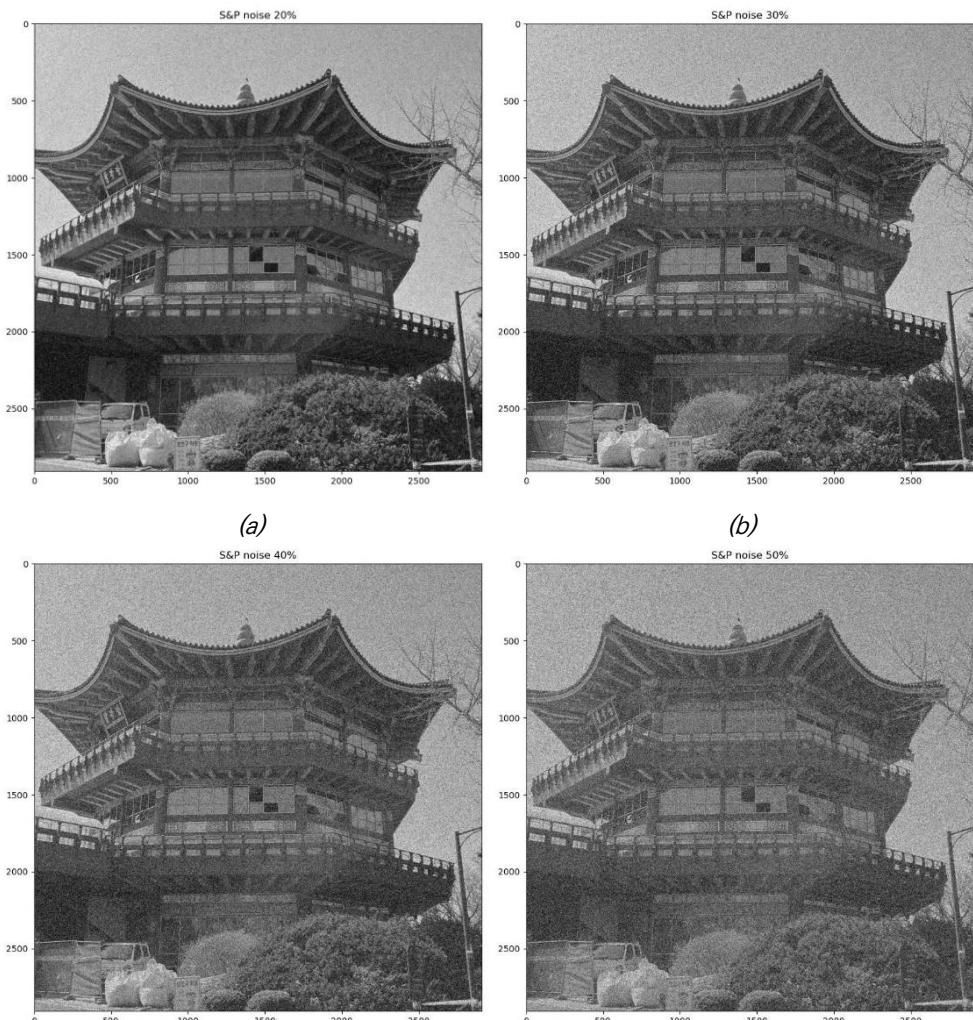
세 번째, 크로스 플랫폼 지원이다. Windows, macOS, Linux 등 다양한 운영체제에서 동일한 사용자 경험을 제공한다. 또한, VS Code는 가볍고 빠르며, 커뮤니티에서 다양한 플러그인을 지속적으로 개발하고 있어, 다양한 사용자 취향에 맞는 환경을 구성할 수 있다.

마지막으로, VS Code는 무료이며, 오픈소스이다. 이는 사용자가 무료로 다운로드하고 사용할 수 있으며, 커뮤니티에서 함께 개발하고 개선할 수 있다는 것을 의미한다.
이러한 이유들로 VS Code 소프트웨어를 선택하게 되었다.

3. 연구 과정

3.1 Add Salt & pepper noise

다양한 필터링 기법들을 연구하기 위해 원본 RGB 영상에 Salt and Pepper(이하 S&P) 잡음을 추가해 연구를 진행했다. S&P 잡음은 일반적으로 신호 전파 과정에서 영상에 나타나는 희고 검은 점들이다. 이러한 점들은 영상 품질을 떨어트리며 영상 분석에 어려움을 야기시킨다. 따라서 본 연구에서는 다양한 S&P 잡음을 최소화시키는 기법들을 비교하고 성능을 검증하려 한다. 연구에 앞서 원본 데이터에 S&P 잡음을 넣는 작업을 실행하였다. Black point를 0으로 White point를 255로 두고 numpy의 random 함수를 이용해 랜덤으로 이미지 내에 S&P 잡음을 심었다. 잡음 밀도(P)는 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%로 20%부터 시작해서 10%씩 증가시켜 90%까지 잡음 밀도를 높이는 이미지를 제작하였다.



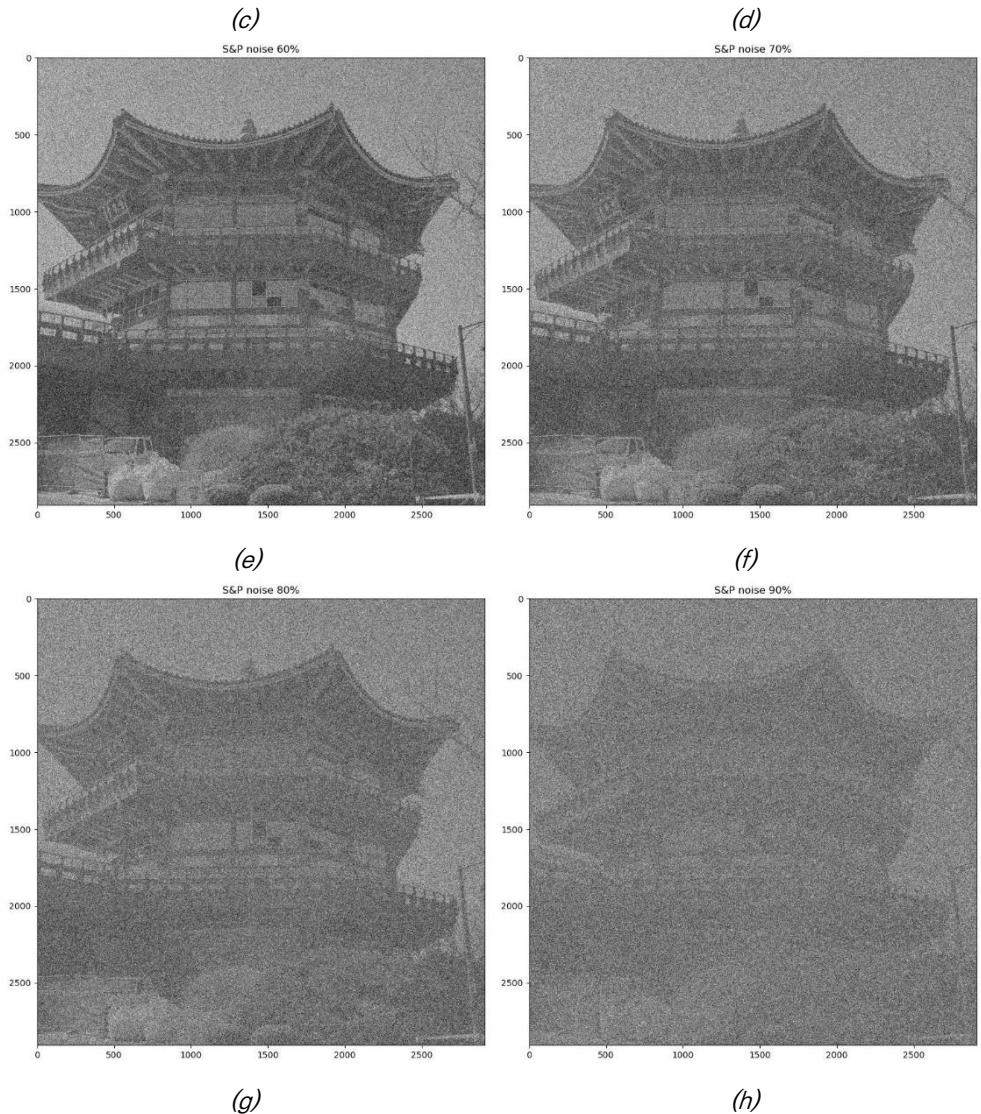
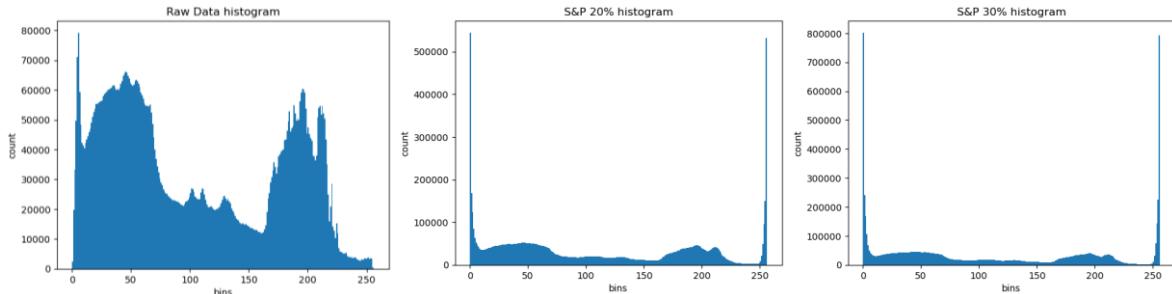


Fig 3. Salt and Pepper Noise (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

3.2 Histogram

원본 이미지와 S&P를 추가한 이미지를 비교하기 위해 matplotlib.pyplot을 이용해 histogram을 도시하였다. 그렇게 총 9개의 histogram을 그려볼 수 있었다.



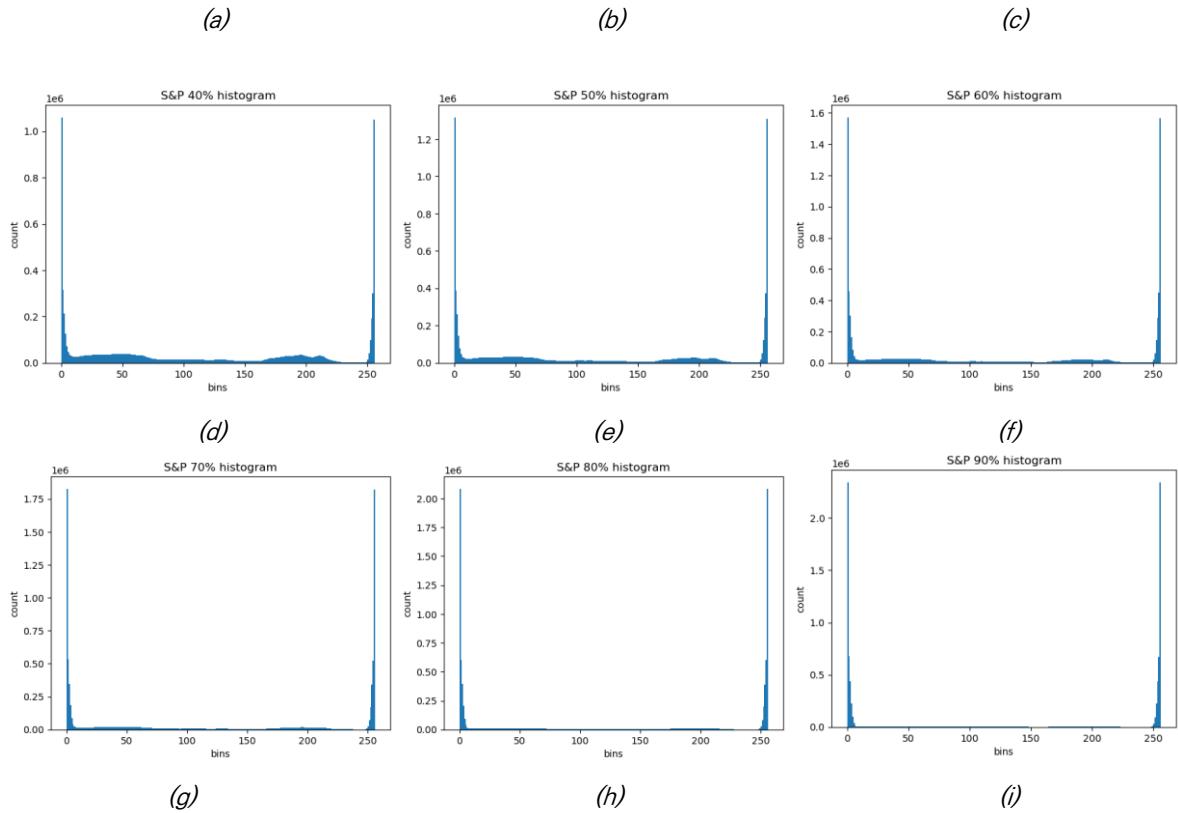


Fig 4. Histogram (a) Raw Data, (b) $P=20\%$, (c) $P=30\%$, (d) $P=40\%$, (e) $P=50\%$, (f) $P=60\%$, (g) $P=70\%$, (h) $P=80\%$, (i) $P=90\%$

Histogram의 x 축은 픽셀 값의 구간이며, y축은 그 값에 해당하는 픽셀의 개수이다. 잡음을 추가해준 영상을 살펴보면 0, 255 영역에서 뾰족하게 나온 것을 확인할 수 있다. S&P 잡음이 90%까지 추가된 마지막 영상에서는 0과 255의 히스토그램의 빈도수가 매우 크게 나타나는 것을 파악할 수 있다. 특정 값이 매우 커 사이 구간에 대한 값을 구별하기 힘들지만 (a)와 닮은 분포를 띠고 있을 거라고 추정할 수 있다.

3.3 Spatial filtering

공간 필터링(Spatial filtering)은 이미지 처리 과정에서 픽셀의 위치에 따라 값을 변경하여 이미지를 처리한다. 구체적으로는 필터를 원본 이미지에 적용하여 새로운 이미지를 만든다. 이때 필터는 이미지의 각 픽셀을 중심으로 한 $m \times n$ 행렬 형태이다. 필터 내의 중심 값을 제외한 나머지 같은 중심 픽셀의 새 값을 계산하는 데 사용된다. 공간 필터링은 이미지 처리에서 중요한 기술로, 가장자리나 질감처럼 이미지의 특징을 강화하거나 잡음과 같은 원치 않는 요소들을 제거하는 데 사용되어 영상 품질을 향상시키는데 필수적이다. 필터의 종류에는 여러 가지가 있지만, 원하는 작업 및 처리하는 이미지의 특성에 따라 달라진다.

Smoothing filtering에서는 Mean, Gaussian, Median, Bilateral, PA(proposed algorithm) 필터를 적용해볼 예정이고, Sharpening filtering에서는 Laplacian, Sobel, Unsharp Masking (USM), Canny를 적용해볼 것이다.

3.3.1 Smoothing

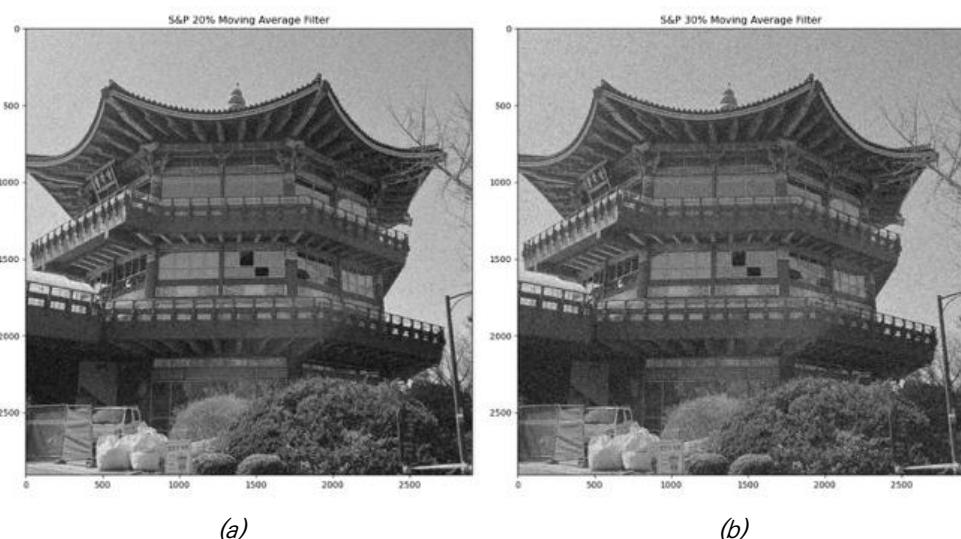
Smoothing filtering은 3×3 의 필터에 편차를 주어 원래 화소에 좀 더 가깝게 설정한다. 이를 이용해 잡음이 있는 영상에서 잡음을 제거하는 영상처리를 할 수 있다. 본 연구에서는 smoothing filtering을 잡음이 존재하는 영상에만 적용하려고 한다. 원본 영상에 적용하지 않는 이유는 다음과 같다. Smoothing filtering 자체가 잡음을 제거하기 위해 픽셀값을 변경하게 되면서 경계를 흐릿하게 하는데, 이는 오히려 원본 영상에서 우리가 얻고자 하는 정보를 지운다. 그렇기에 원본 영상에서는 경계가 흐리게 되면서 선명도가 떨어져 원본 영상이 담고 있는 정보가 손실되나, 잡음이 있는 영상에 적용한 앞으로 나올 이미지와 같이 잡음이 제거되어 본래 원하고자 하는 정보를 얻는 영상처리가 진행되었음을 알 수 있다.

1) Mean (Moving average)

이 필터는 이미지의 잡음을 제거하거나, 이미지에서 강조하려는 성분을 강화하기 위해 사용된다. 이 필터는 이미지의 각 픽셀 주변에 윈도우(window) 또는 커널(kernel)이라고 하는 고정 크기의 박스를 적용한다. 이 때, 윈도우 내의 모든 픽셀 값을 더한 후, 윈도우 크기로 나누어서 중심 픽셀의 값을 새로운 값으로 대체한다. 이동 평균 필터는 간단하면서도 효과적인 필터링 방법 중 하나이다. 이 필터는 이미지의 부드러운 효과를 줄 수 있어서 이미지를 흐리게 만들 수도 있다.

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (\text{식 1.1})$$

윈도우의 크기를 크게 하면 이미지가 더욱 부드러워지지만, 이미지의 세부 정보가 손실될 수 있다. 이동 평균 필터는 OpenCV의 cv2.boxFilter() 함수를 사용하여 구현할 수 있다. 이 함수는 커널의 크기와 윈도우의 크기를 지정할 수 있다. 그러나 이 함수는 이미지의 각 픽셀에 대해 커널을 적용하기 때문에, 연산 속도가 느려질 수 있다. 따라서, 이동 평균 필터를 구현할 때는 이미지의 해상도와 커널의 크기를 고려하여 적절한 값을 선택해야 한다. 아래의 그림은 S&P 잡음에 따른 3×3 커널 필터링을 한 예시이다.



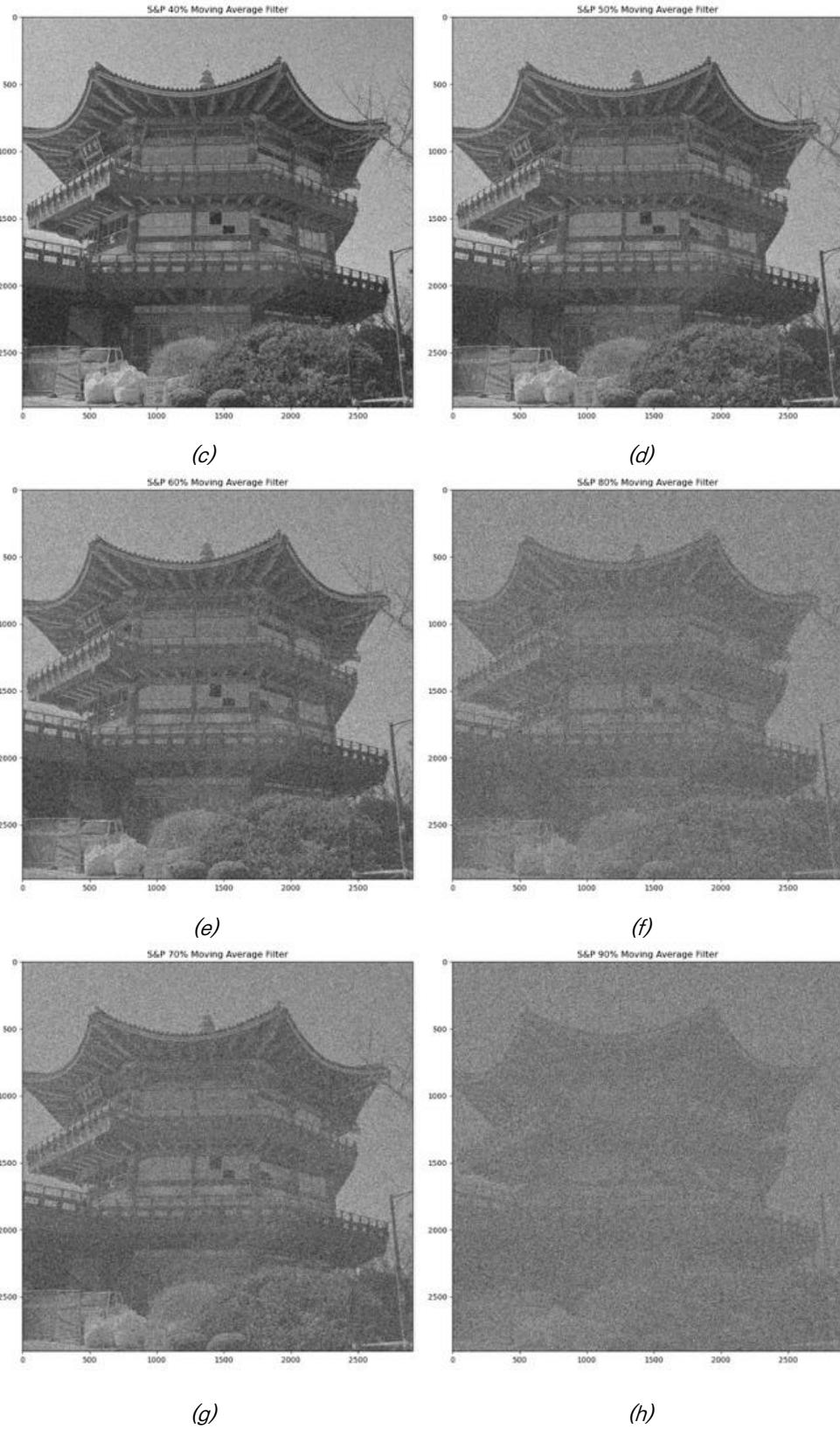
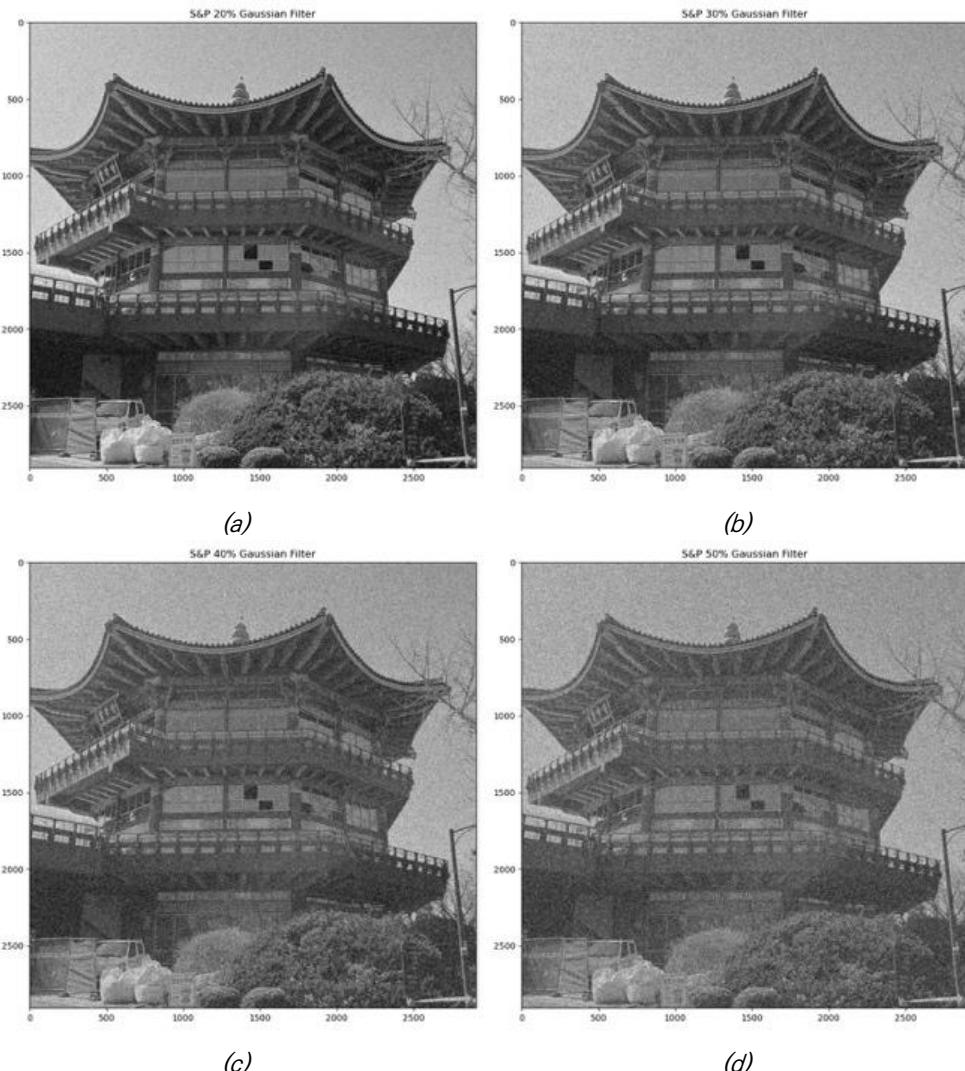


Fig 5. Mean Filter (Moving Average Filter) (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$,
(g) $P=80\%$, (h) $P=90\%$

2) Gaussian

가우시안 필터링(Gaussian filtering) 방법은 이동 평균 필터와 비슷하지만, 박스 필터 대신 가우시안 커널을 사용한다. 가우시안 커널은 가우시안 함수를 이용하여 만들어진 수학적인 필터이다. 이 필터는 확률 분포 중 하나인 가우시안 분포를 따르는 값들을 가진다. 이 필터를 적용하면 마찬가지로 이미지의 잡음을 제거하거나, 이미지를 부드럽게 만드는 데 사용된다. 커널의 크기와 표준 편차 값을 조절하여, 다양한 종류의 가우시안 커널을 만들 수 있다.

가우시안 커널은 이미지의 각 픽셀 주변에 적용되며, 커널의 값은 주변 픽셀의 가중 평균으로 계산된다. 이 가중치는 가우시안 함수를 이용하여 계산된다. 따라서, 커널 내의 중심 픽셀은 다른 픽셀보다 더 큰 가중치를 가지며, 노이즈나 영상의 특징적인 부분들을 부드럽게 처리한다. 본 논문에서는 cv2.GaussianBlur() 함수를 사용하여 구현할 것이다. 이 함수는 커널의 크기와 표준 편차(standard deviation) 값을 지정할 수 있다. 표준 편차 값이 작을수록, 이미지의 잡음을 제거하는 데 효과적이다.



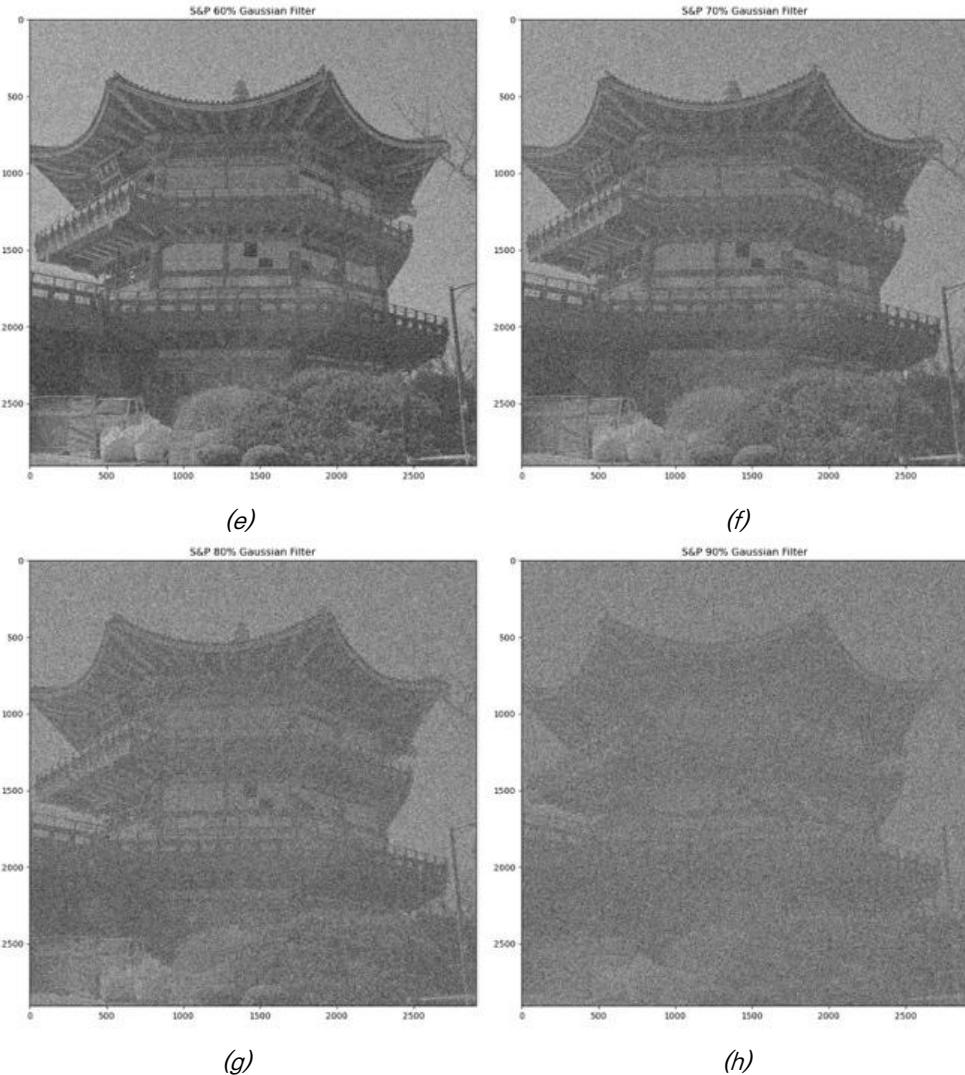
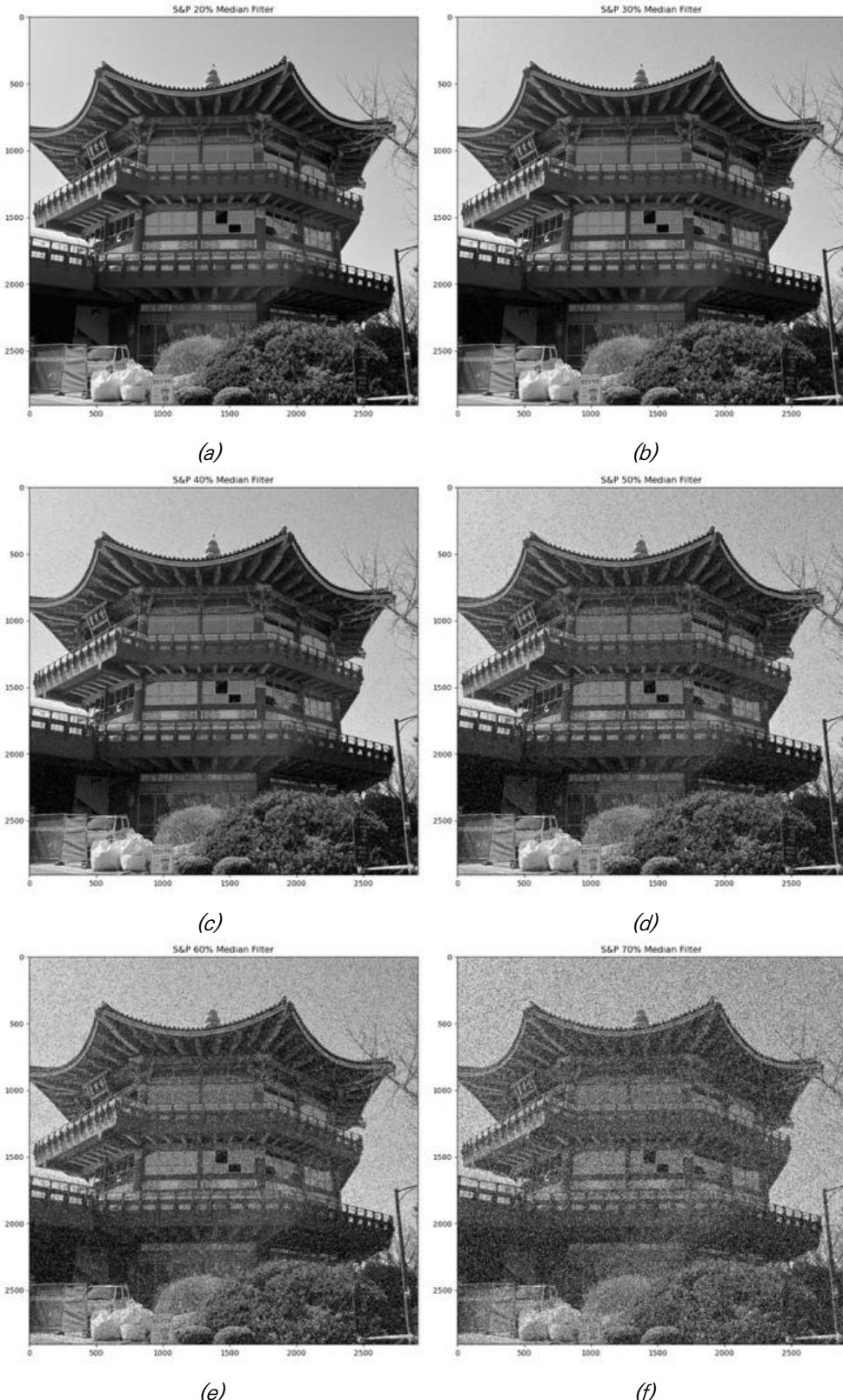


Fig 6. Gaussian Filter (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

3) Median

이 기법은 커널 영역 내의 모든 픽셀 값의 중앙값을 계산하여 중심 픽셀 값을 해당 중앙값으로 대체한다. 중앙값 필터링은 다른 필터와 달리, 이미지 내의 픽셀 값들의 중앙값을 사용하여 필터링을 수행한다. 특히 이 기법은 S&P 잡음을 제거하는데 매우 효과적인데, 그 이유는 이미지 내에 있는 잡음이나 이상값(outlier) 들이 다른 픽셀 값들과 매우 다른데, 이 값을 평균 값이나 다른 값을 사용하는 필터링 방법으로 제거하기 어렵기 때문이다. 본 연구에서는 cv2.medianBlur()를 사용하여 이미지의 S&P 잡음을 다른 필터링 기법들에 비해 효과적으로 제거하는 것을 보여줄 것이다. 이 커널의 특징 중 하나는 커널의 크기가 홀수여야 한다는 것이다. 또한 작은 크기의 커널을 사용할 경우 이미지의 세부 정보가 손실될 수도 있다.



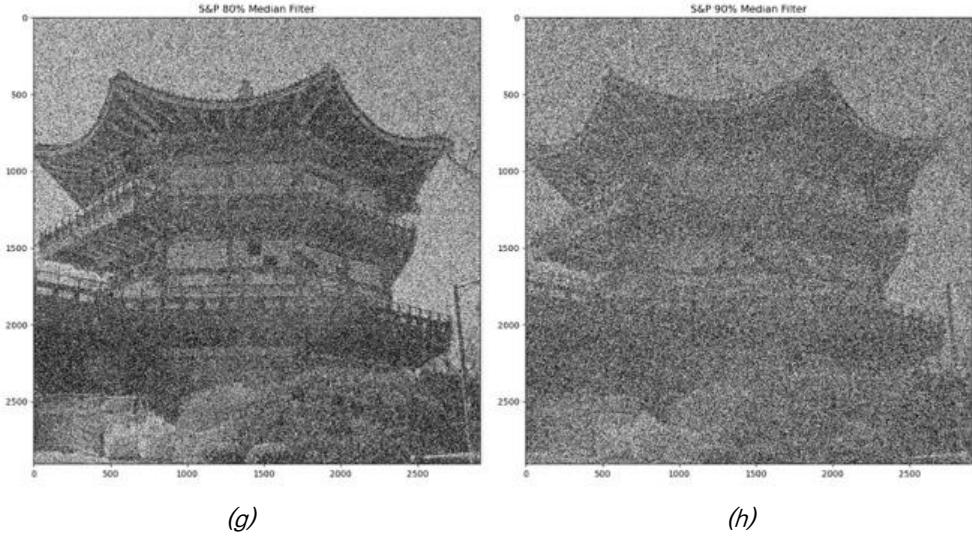
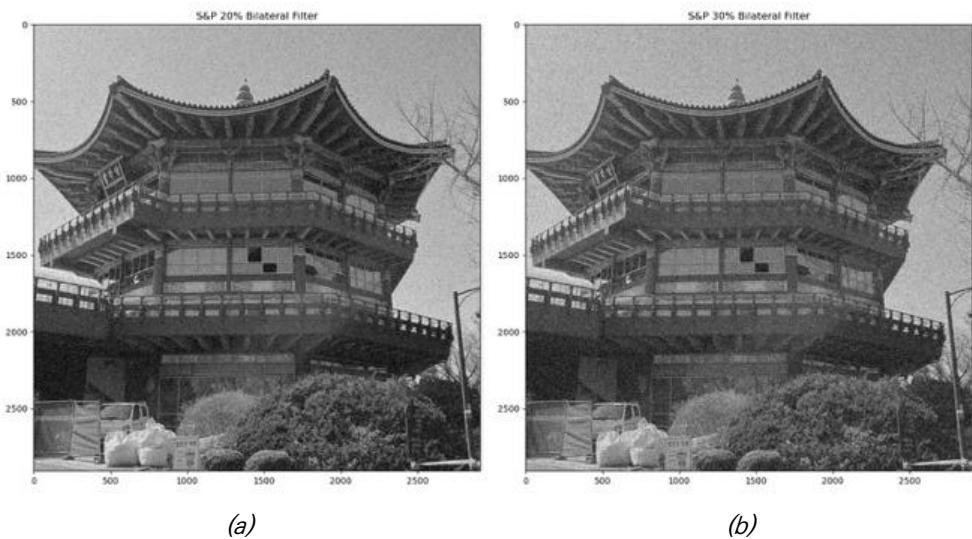


Fig 7. Median Filter (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

4) Bilateral

가우시안 필터링과 마찬가지로, Bilateral 필터링은 공간 상에서 가우시안 필터를 적용하지만, 픽셀 간의 차이에 대한 가우시안 필터도 함께 적용한다. 이를 통해, 중심 픽셀과 비슷한 픽셀만을 대상으로 필터링을 수행하여, 가장자리를 보존하면서 이미지의 잡음을 제거할 수 있다. 즉, Bilateral 필터링은 이미지에서 픽셀 간의 공간적 거리와 강도 차이를 모두 고려하여 필터링을 수행한다. 이를 통해, 주변 픽셀의 밝기와 가장자리 모두 고려하여 이미지의 잡음을 제거할 수 있다. 따라서, 가장자리에 대한 특징 정보가 매우 중요한 영상에서, 이 필터링은 다른 필터링 방법보다 더욱 효과적이다. 본 연구에서는 cv2.bilateralFilter() 함수를 이용하여 원본 영상의 가장자리를 보존하면서 이미지의 잡음을 제거하는 것을 보여줄 것이다. 그러나 다른 필터보다 연산 속도가 느리기 때문에, 빠른 필터링이 필요한 경우에는 다른 필터를 사용하는 것이 좋다.



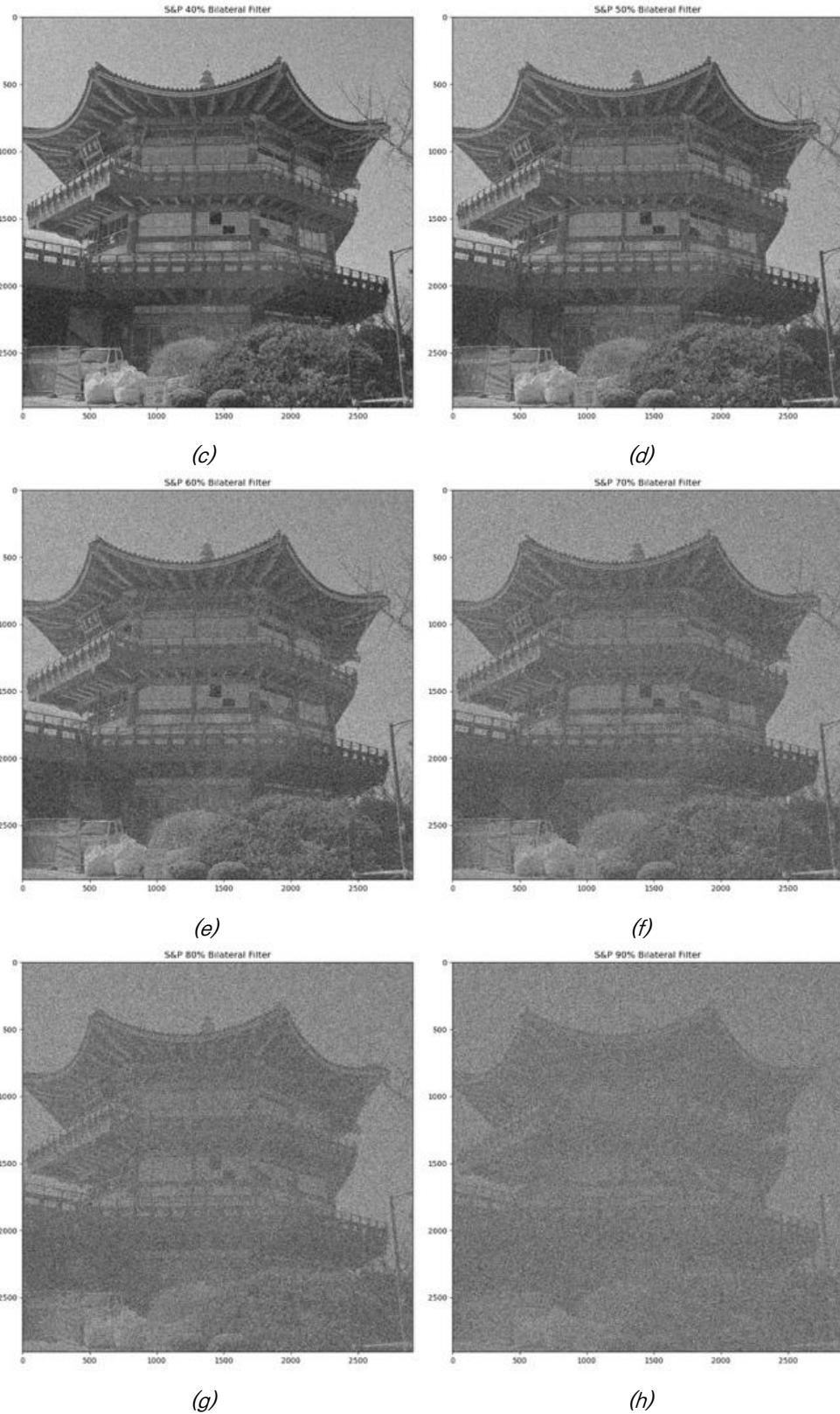


Fig 8. Bilateral Filter (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

5) Proposed Algorithm

본 연구에서는 기존에 사용되던 보편적인 S&P 잡음 제거 기법 이외에 새로운 알고리즘을 제시하겠다. 기본적인 방법은 다음과 같다. 중심 픽셀이 S&P 잡음이 아닌 경우 유지하고, 잡음이 맞는 경우 잡음 중심 픽셀 주변 잡음 밀도에 따라서 커널 크기를 다르게 한다.

$$x(i,j) = \begin{cases} n & \text{prob } P \\ I & \text{prob } 1 - P \end{cases} \quad (\text{식 2.1})$$

위의 식에서 P의 확률은 S&P 잡음의 밀도를 나타낸다. S&P 잡음의 경우 0과 255의 값을 가지므로, 이 값을 제외한 나머지 값을 갖는 원 영상의 밀도는 1-P이다. 잡음의 처리 과정은 다음과 같다.

Step 1: 중심 픽셀값이 비잡음인 경우, 원 영상의 픽셀값을 넣는다.

$$O(i,j) = I \quad (\text{식 2.2})$$

Step 2: 중심 픽셀값이 잡음인 경우, 크로스 커널을 만들고 비잡음 픽셀의 개수에 따라 출력을 다르게 한다.

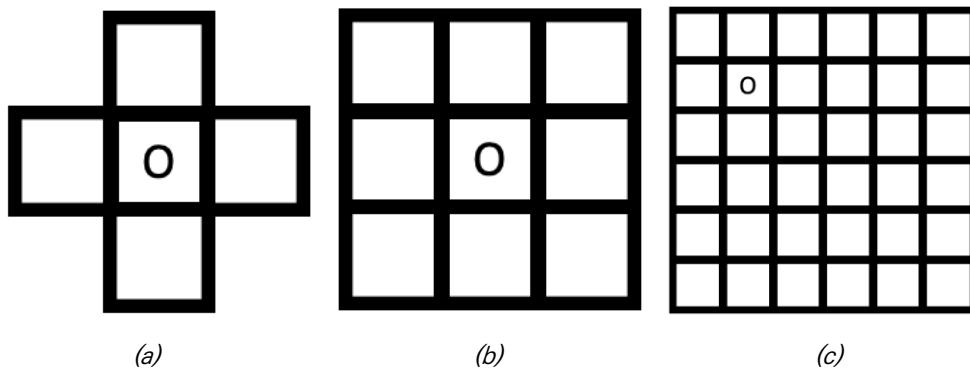


Fig 9. PA Filter (a) cross kernel, (b) 3x3 kernel, (c) 6x6 kernel

Step 3: 크로스 커널 내에 비잡음 픽셀이 2개 이상일 경우, 픽셀들에 대한 NM1 집합을 만들고, 다음 (식 2.3)을 구한다. 만약 비잡음 픽셀 개수가 1개일 경우, 비잡음 픽셀 값으로 대체한다. (식 2.4)

$$O(i,j) = \frac{1}{2}\{\max(\text{NM1}) + \min(\text{NM1})\} \quad (\text{식 2.3})$$

$$O(i,j) = \text{NM1} \quad (\text{식 2.4})$$

Step 4: 크로스 커널 내에 비잡음 픽셀이 없는 경우, 3 x 3 커널을 생성한다. 이때 비잡음 픽셀들의 집합을 NM2라 하고, 개수를 k라 한다. 만약 k가 홀수일 경우 (식 2.5), 짝수일 경우 (식 2.6)을 구한다.

$$O(i,j) = \text{Median}(\text{NM2}) \quad (\text{식 2.5})$$

$$O(i,j) = \frac{1}{2} \left\{ NM2 \left(\frac{k}{2} \right) + NM2 \left(\frac{k}{2} + 1 \right) \right\} \quad (\text{식 2.6})$$

Step 5: 3×3 커널 내에 비잡음 픽셀이 없는 경우 6×6 커널을 생성한다. 커널 내의 잡음 밀도가 50% 이하인 경우 거리에 대한 가중치를 매긴 이후 출력값을 정한다.

$$D(p,q) = \sqrt{(N - \text{abs}(p))^2 + (N - \text{abs}(q))^2} \quad (\text{식 2.7})$$

$$W(p,q) = \{D(p,q) + 1\}^3 \quad (\text{식 2.8})$$

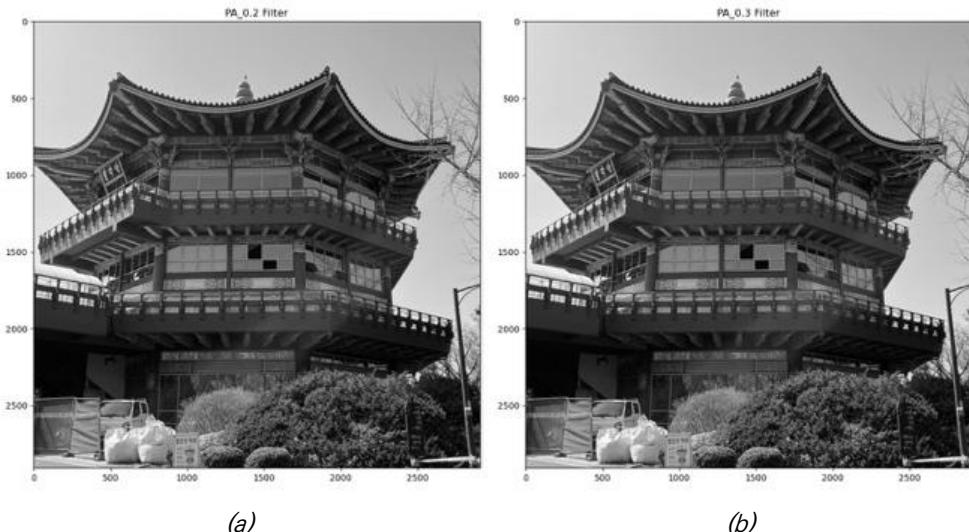
$$F(p,q) = \begin{cases} 1 & \text{Not noise} \\ 0 & \text{0 or 255} \end{cases} \quad (\text{식 2.9})$$

$$O(i,j) = \frac{\sum F(p,q)*W(p,q)*\text{Pixel Value}}{\sum F(p,q)*W(p,q)} \quad (\text{식 2.10})$$

만약 커널 내의 잡음 밀도가 50% 이상인 경우 Fig 9. (c)를 각각 4개의 구역을 나눈 뒤 Step 4를 수행한다.

Step 6: 6×6 커널 내의 비잡음 픽셀 개수가 0인 경우 다음 식을 수행한다.

$$O(i,j) = \frac{1}{4} \{ O(i-1,j-1) + O(i-1,j) + O(i-1,j+1) + O(i,j-1) \} \quad (\text{식 2.11})$$



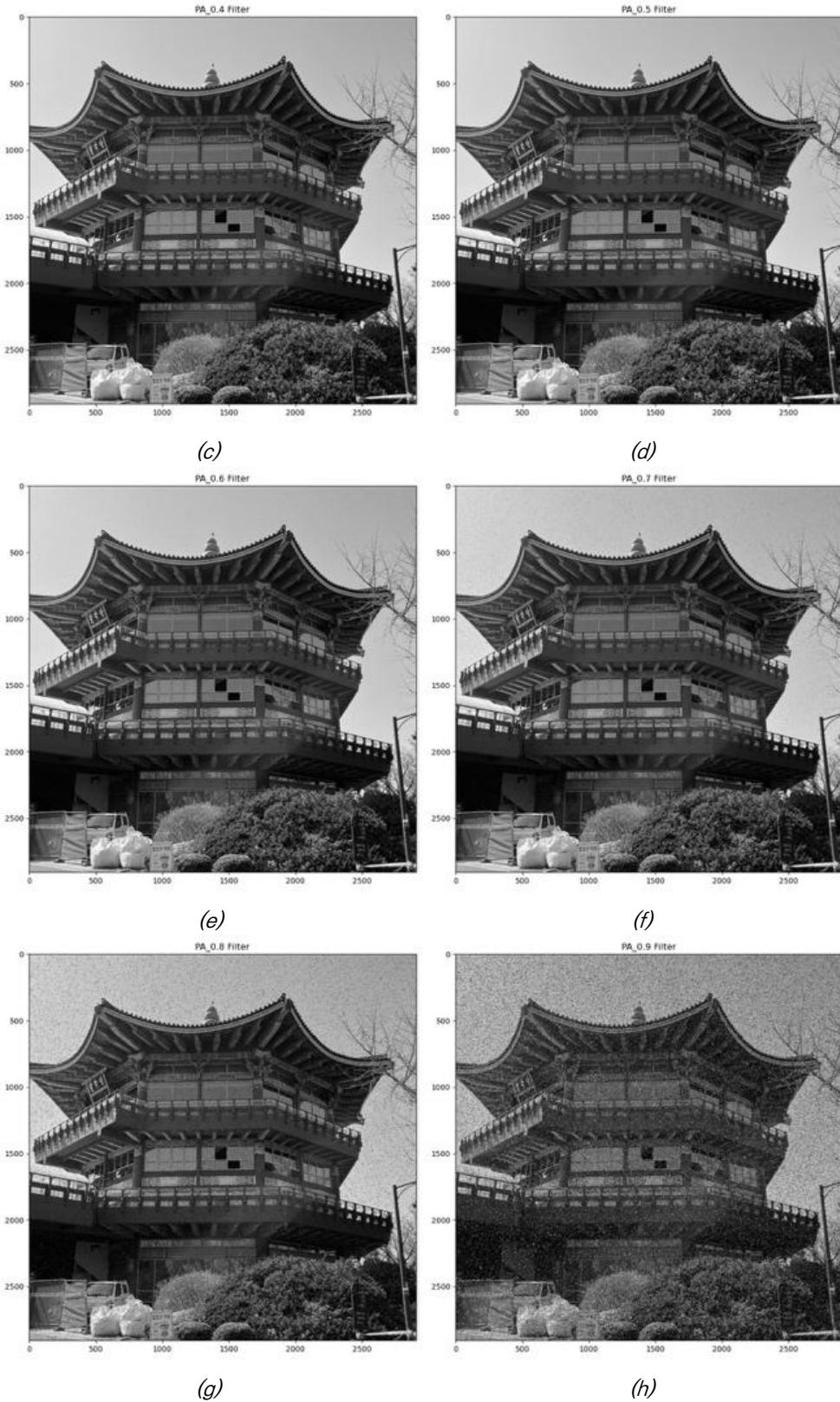


Fig 10. Proposed Algorithm Filter (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

3.3.2 Sharpening

Sharpening filtering은 영상처리에서 이미지를 더욱 선명하게 만드는 기법 중 하나이다. 이 기법은 공간영역 필터링 기법 중 하나이며, 이미지의 각 픽셀 값을 조정하여 선명도를 높이는 방식으로 작동한다.

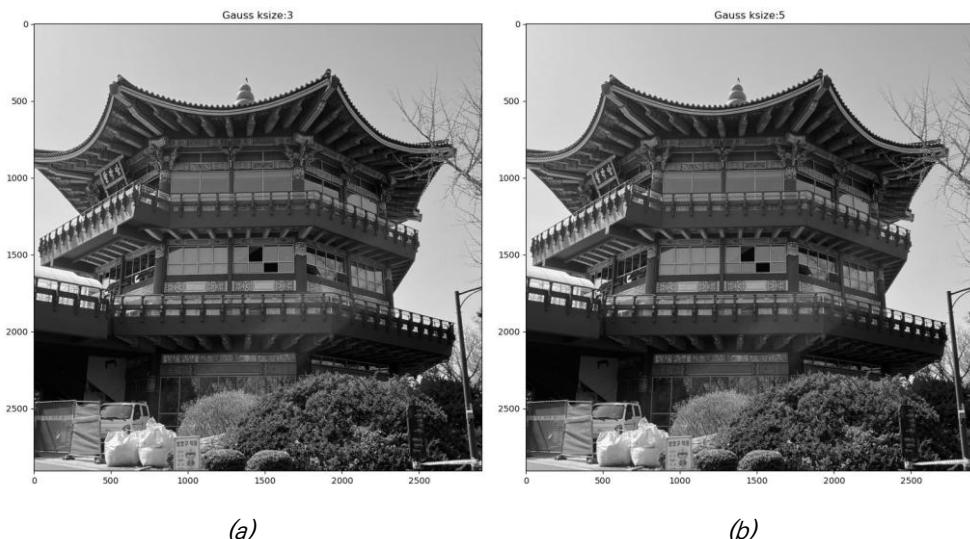
일반적으로 sharpening filtering은 먼저 이미지의 원본과 유사한 형태의 필터링 마스크를 생성한다. 그런 다음, 이미지에 이 마스크를 적용하여 이미지의 각 픽셀 값을 수정한다.

예를 들어, "Unsharp Masking (USM)" 필터는 이미지의 원본과 유사한 마스크를 만들고, 원본 이미지에서 이 마스크를 빼서 이미지를 선명하게 만든다. 이 마스크는 중심에 밝은 값이 있고 주변에는 음수 값이 있어, 중심의 픽셀 값을 강조하고 주변 픽셀 값을 약화시키는 효과가 있다.

다른 예로는 "Laplacian of Gaussian" 필터가 있다. 이 필터는 먼저 이미지에 가우시안 필터를 적용하여 노이즈를 제거한다. 그런 다음, 라플라시안 필터를 적용하여 이미지의 경계를 검출하고 강조한다.

이러한 sharpening filtering 기법은 이미지의 세부 정보를 강조하고 선명도를 높여 이미지를 더욱 선명하게 만든다. 그러나 이러한 필터링은 이미지 노이즈를 더욱 강조할 수도 있으므로, 필터링 전에 이미지를 미리 처리하여 노이즈를 제거하는 것이 좋다.

위에서 설명한 대로 노이즈가 있는 영상을 sharpening 시키는 것은 이미지의 품질을 오히려 떨어트리는 작업이기 때문에 Sharpening filtering에 들어가기 앞서 원본 데이터에 Gaussian Filter를 이용해 경계선을 흐릿하게 만들었다. 이렇게 해준 이유는 원본 데이터에 대한 sharpening보다는 품질이 떨어지는 영상을 sharpening해 원본영상에서 추출한 경계와 sharped 영상에서 추출한 경계를 비교하는 것이 더 의미가 있을 것이라고 생각해, 연구 대상 이미지를 Gaussian Blur를 거친 영상 4개를 선택하였다. 각각 kernel size를 3x3, 5x5, 7x7, 9x9로 sigma는 0으로 설정하였다.



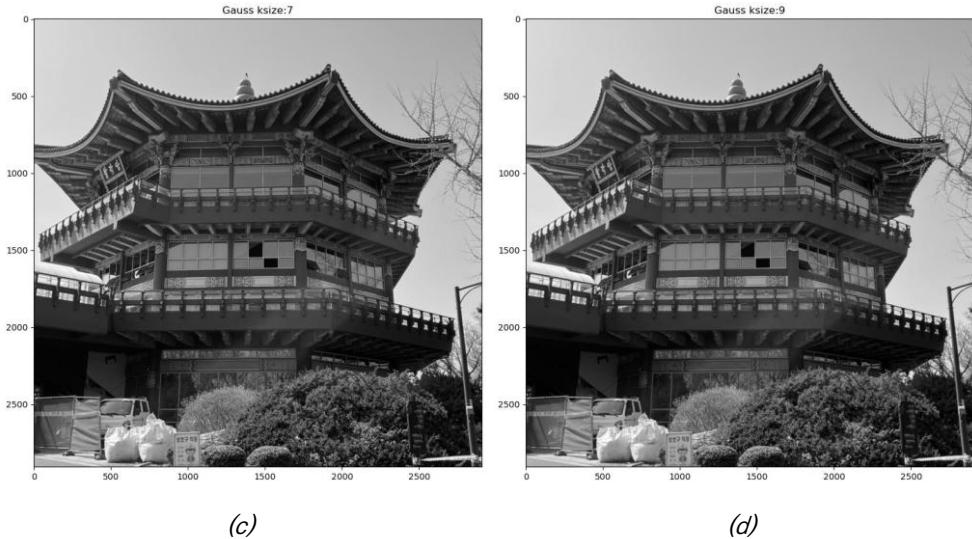


Fig 11. Gaussian Blur (a) k size=3, (b) k size=5, (c) k size=7 (d) k size=9

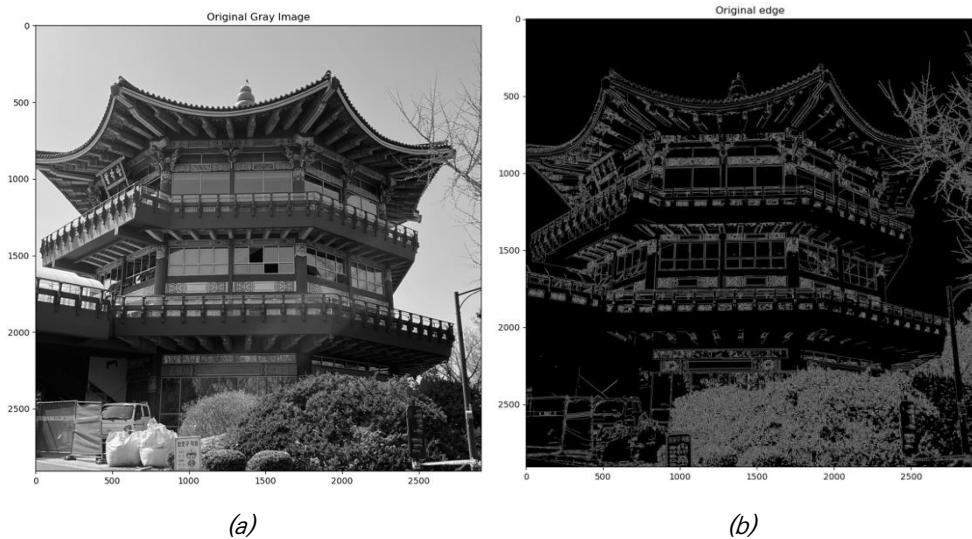


Fig 12. Raw Data and Edge of Raw Data (a) Raw Data, (b) Edge of Raw data

1) Laplacian

Laplacian 필터는 영상에서 미분을 이용하여 경계를 검출하는 필터 중 하나이다. 2차 미분 연산자를 사용하여 입력 영상의 밝기 값 변화량을 계산한다. 이때, Laplacian 연산자를 사용하는데, 이는 x와 y축 방향에서의 2차 미분 값을 더한 값이다. 따라서 Laplacian 필터는 경계의 굽곡을 고려하며, 경계가 굽곡되어 있을 때도 이를 추출할 수 있다. 영상 내에서 경계를 찾는 필터는 크게 2가지가 존재하는데 바로 등방성과 비등방성 필터이다. 본 연구에서는 가장 대표적인 등방성 필터인 Laplacian 필터를 사용할 것이다.

cv2.Laplacian을 이용해 앞서 만든 영상에 적용하고 이것을 cv.canny를 이용해 경계를 추출하였다.

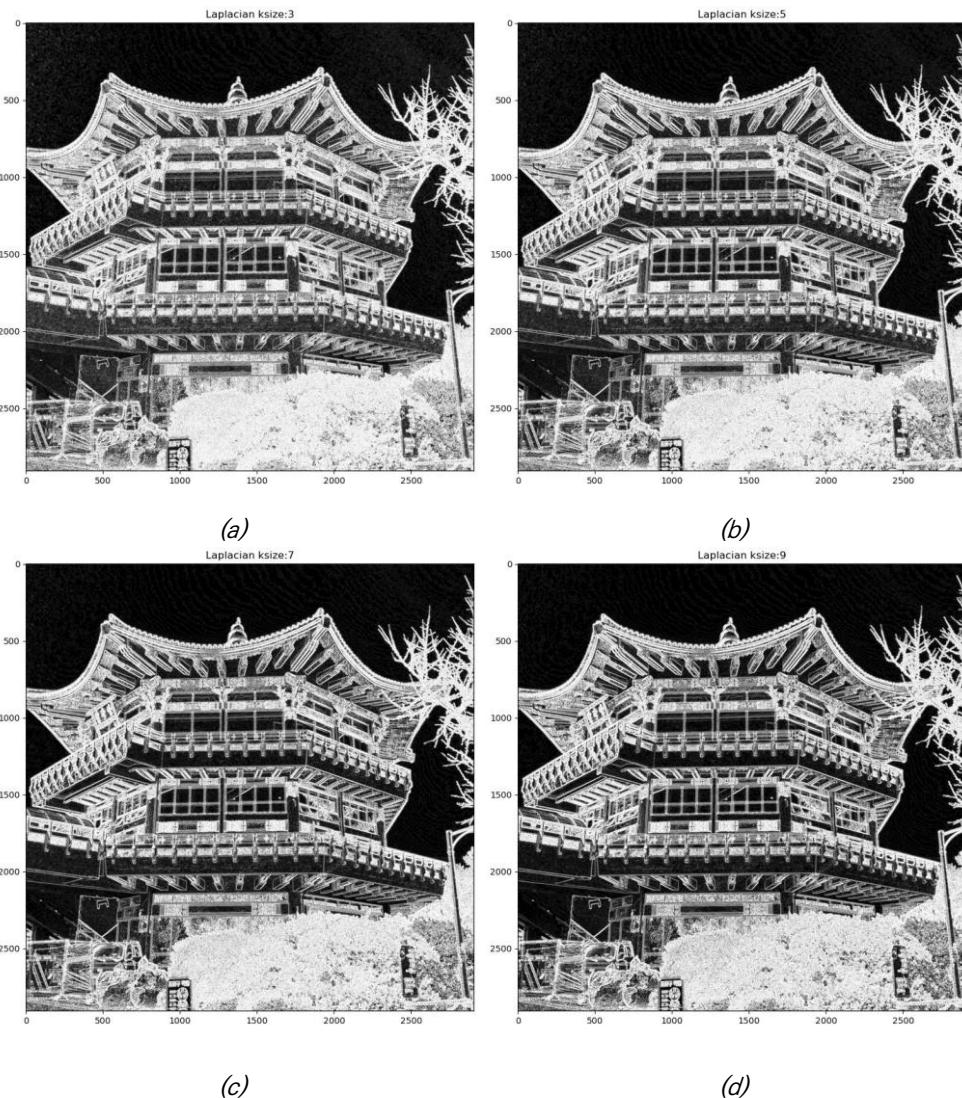
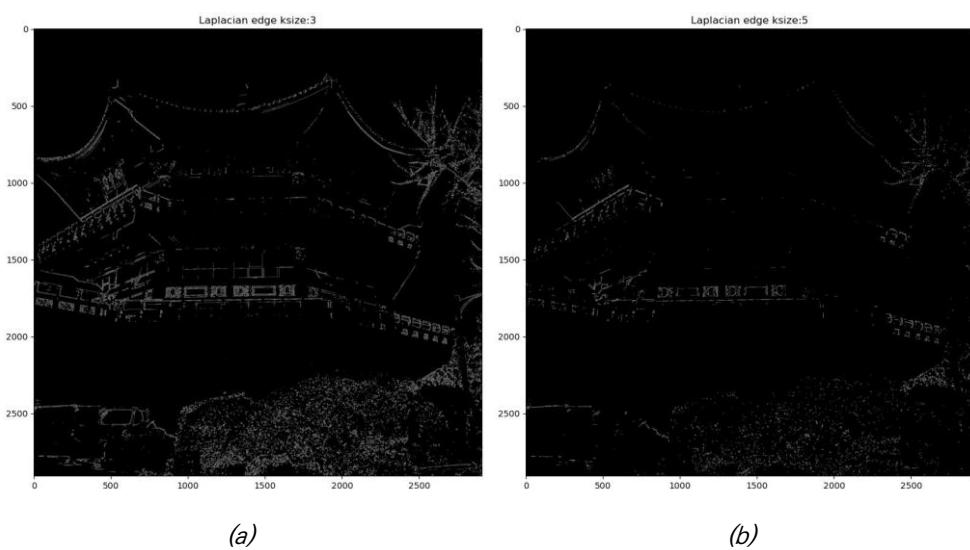


Fig 13. Laplacian Filter



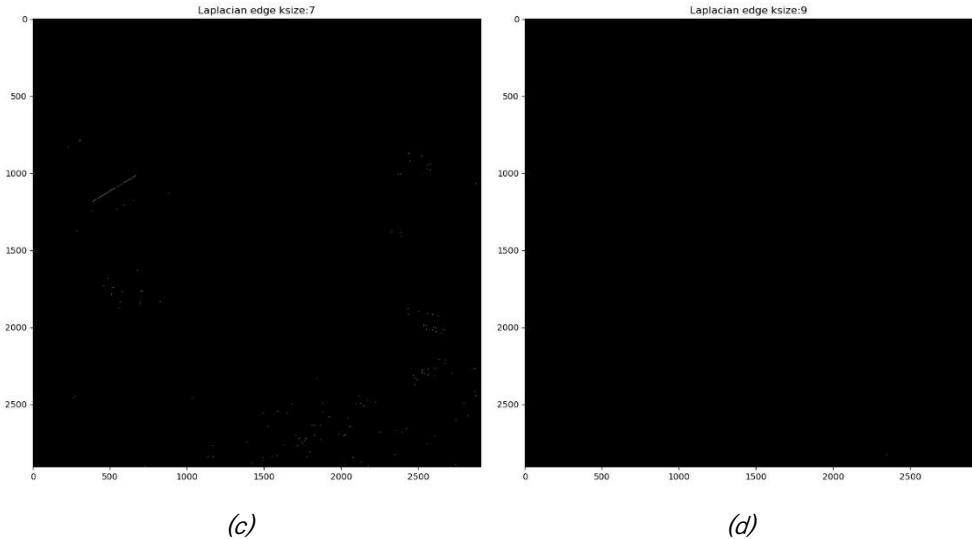
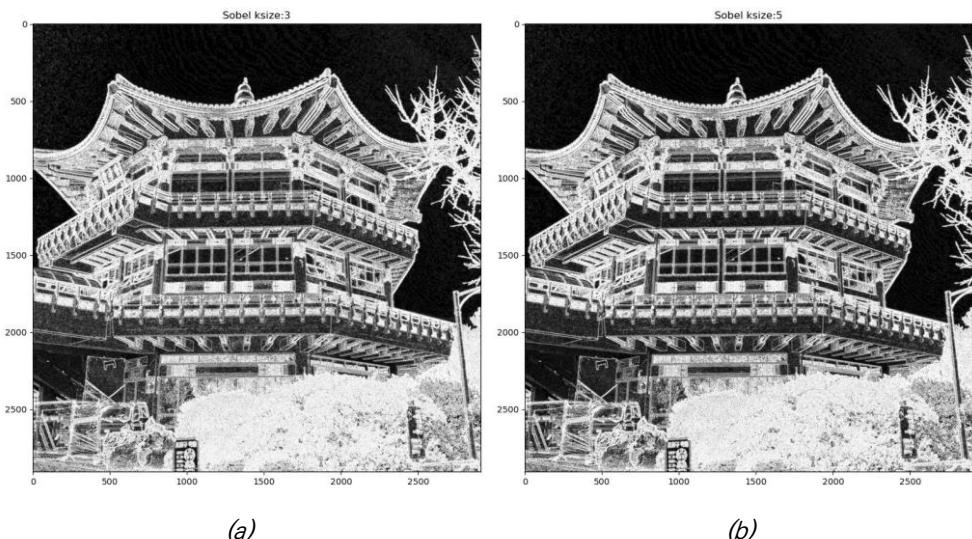


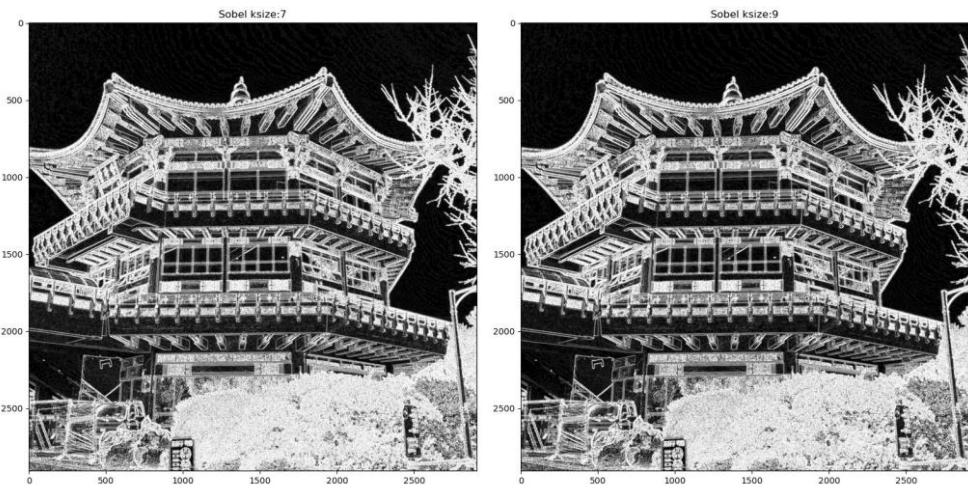
Fig 14. Edge of Laplacian Filter

2) Sobel

Sobel 필터는 마찬가지로 경계 추출을 위해 미분 연산자를 사용하는 필터이다. 입력 영상에서 x와 y방향으로 미분을 수행한 후, 이를 이용하여 그레디언트(gradient) 값을 계산한다. 그레디언트 값은 x와 y방향으로의 변화량의 벡터값을 합한 것으로, 경계의 방향과 크기를 결정한다. Sobel 필터는 x와 y방향의 미분 연산자를 각각 적용하여 그레디언트 값을 계산한다. 이때, 각 방향에 대해서 중앙 차분을 사용하면, 잡음 대비 강한 경계 추출이 가능해진다.

Laplacian과 마찬가지로 cv.canny를 이용해 경계를 추출해줬다.

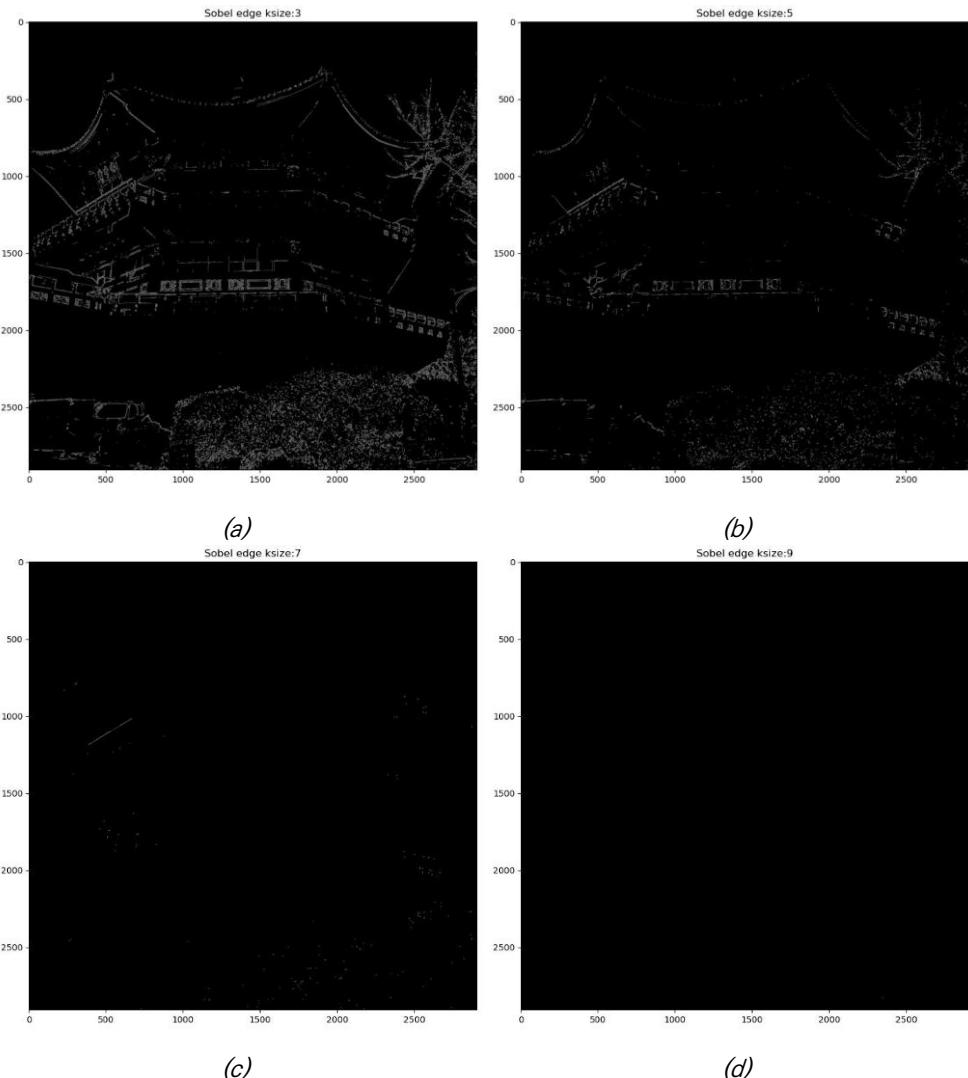




(c)

(d)

Fig 15. Sobel Filter



(a)

(b)

(c)

(d)

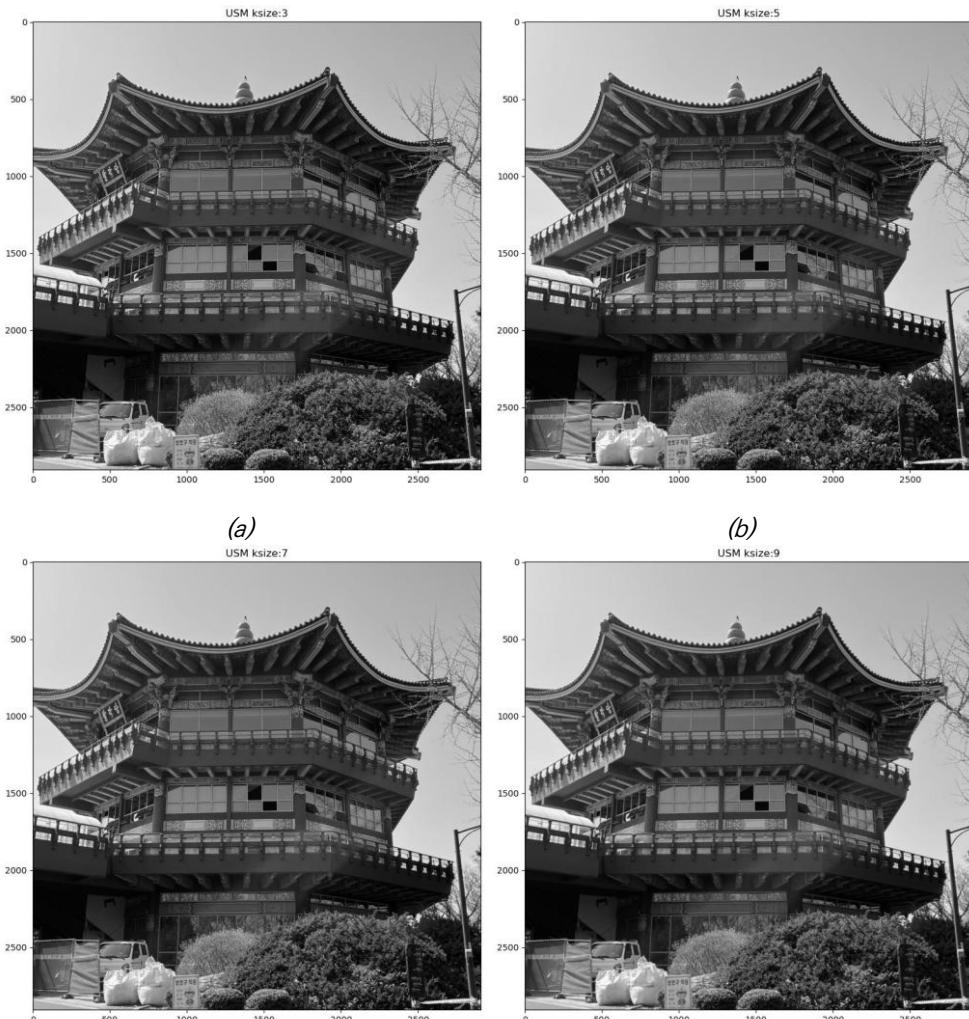
Fig 16. Edge of Sobel Filter

3) Unsharp Masking (USM)

USM 필터링 기법과 기존 Laplacian 필터의 차이점은 바로 미분 사용 여부이다. 위의 필터링 방법에서는 미분을 사용했지만, 사실 사용하지 않고도 이미지의 경계를 추출할 수 있다. USM은 영상의 선명도를 향상시키기 위한 필터이다. 블러링 필터를 적용하여 원본 영상에서 블러된 영상을 뺀 후, 원본 영상에 추가하는 방식으로 진행한다. 이때, 블러링 필터로는 Gaussian 필터를 사용하는 경우가 많아 본 연구에서도 Gaussian 필터를 적용했다. 추가적으로 앞선 방법들처럼 canny를 이용해 경계를 추출해줬다. 처리 방법은 다음과 같다.

1. 입력 이미지 Gaussian Blur 처리
2. Gaussian Blur 과 입력 이미지와의 차이를 통한 Mask 추출
3. Mask 영상과 입력 이미지의 합

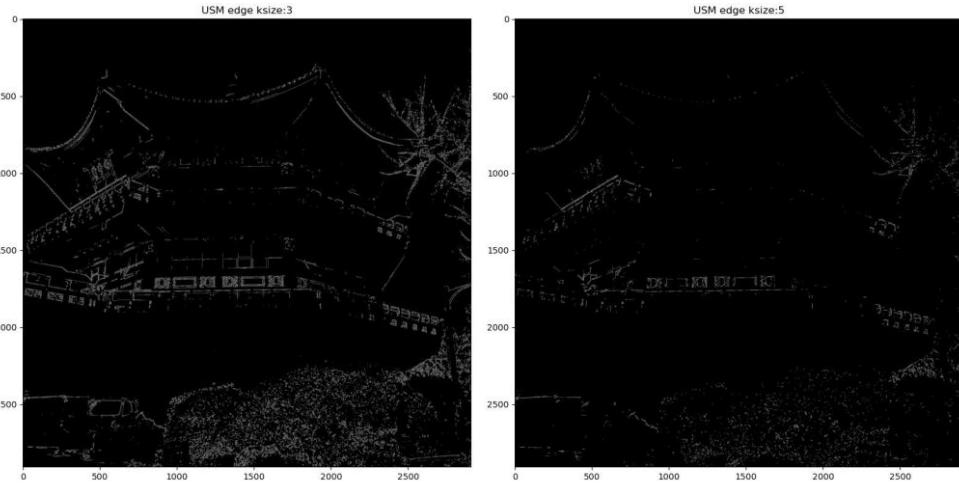
$$g(x, y) = f(x, y) + k * g_{mask}(x, y) \quad (\text{where } k = 1) \quad (\text{식 3.1})$$



(c)

(d)

Fig 17. USM Filter

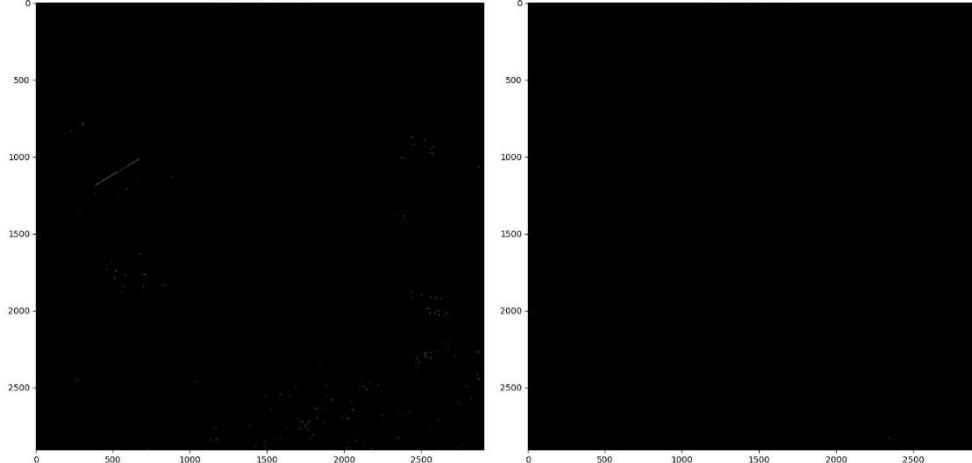


(a)

(b)

USM edge ksize:7

USM edge ksize:9



(c)

(d)

Fig 18. Edge of USM Filter

4) Canny

Canny 필터는 경계 추출을 위한 최적의 방법 중 하나로 알려져 있다. 입력 영상에서 경계를 검출하기 위해 4 단계의 과정을 거친다. 먼저, 가우시안 필터를 적용하여 노이즈를 제거한다. 그 다음으로, Sobel 필터를 사용하여 그레디언트 값을 계산하고, 경계의 방향과 크기를 결정한다. 그 다음, Non-maximum suppression을 적용하여 지울 경계 부분을 결정한다. 이때, 가장 큰 그레디언트 값을 가지는 경계만 남기고, 다른 경계는 억제한다. 마지막 과정으로는 임계값을 설정한 뒤 과정을 수행하여 경계를 검출한다. 이진화 과정은 두 개의 임계값 (threshold)을 사용하는데, 그 중 상위 임계값 이상의 그레디언트 값을 가지는 픽셀을 경계로 판단한다. Canny 필터는 잡음에 강하고, 경계가 끊어지거나 굴곡되어 있을 때도 추출할 수 있으며, 정확한 경계 검출이 필요한 영상 처리에 많이 사용된다. Canny filter 같은 경우는 이미 경계가 확실하게 나와있기 때문에 이대로 바로 분석에

들어갔다.

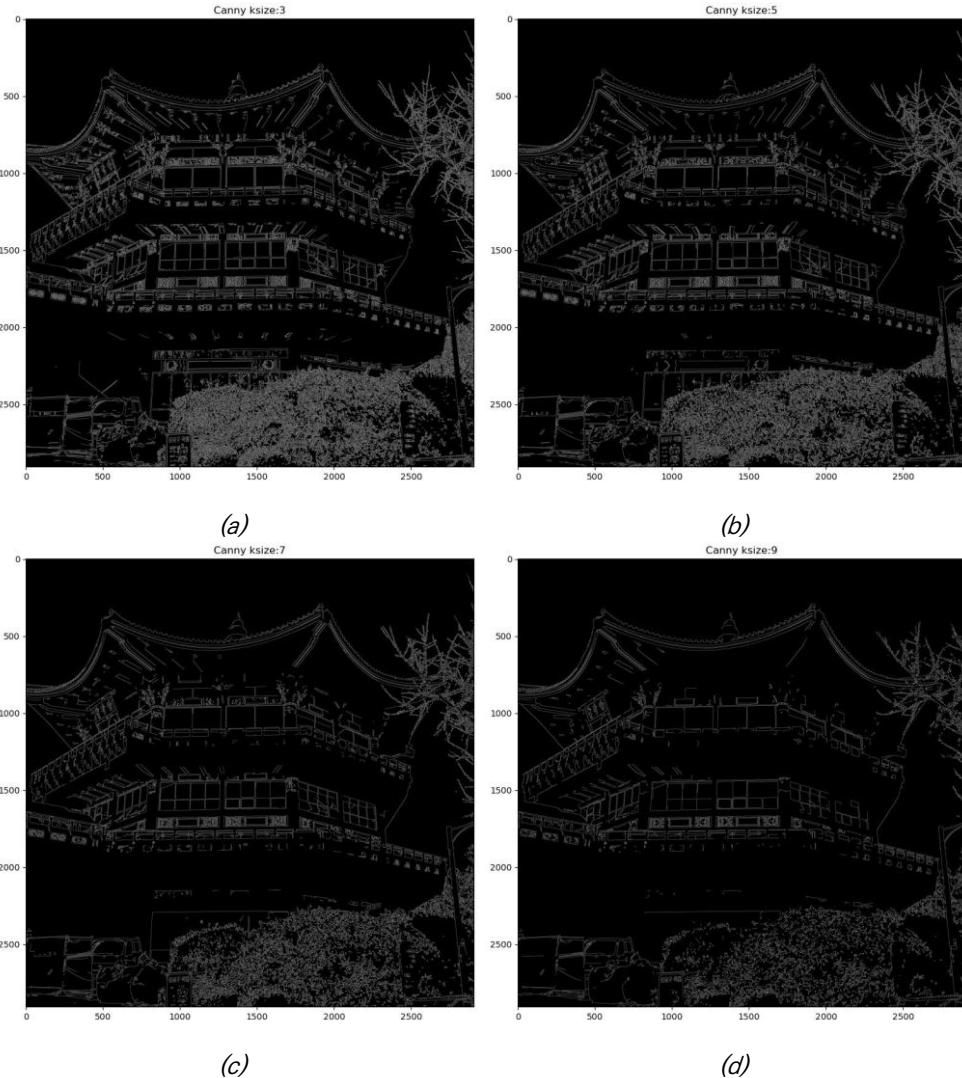


Fig 19. Canny Filter

3.4 Frequency domain filtering

Frequency domain filtering은 image processing에서 사용되는 주요 방법 중 하나이다. 이 방법은 이미지를 Spatial domain에서 Frequency domain으로 변환하여 filtering을 수행하는 것으로, filtering은 고주파 노이즈나 선명도 등과 같은 문제를 해결하기 위해 사용된다.

Frequency domain filtering의 기본 아이디어는 이미지가 Spatial domain에서는 높은 주파수 구성 요소가 포함된 곳에 노이즈나 불필요한 정보가 들어 있다는 것이다. 따라서 이러한 노이즈나 불필요한 정보를 제거하기 위해 이미지를 Frequency domain으로 변환한 다음, 저주파 영역에 있는 필요한 정보만 유지하고 고주파 영역의 노이즈와 불필요한 정보를 제거하는 필터를 적용한다.

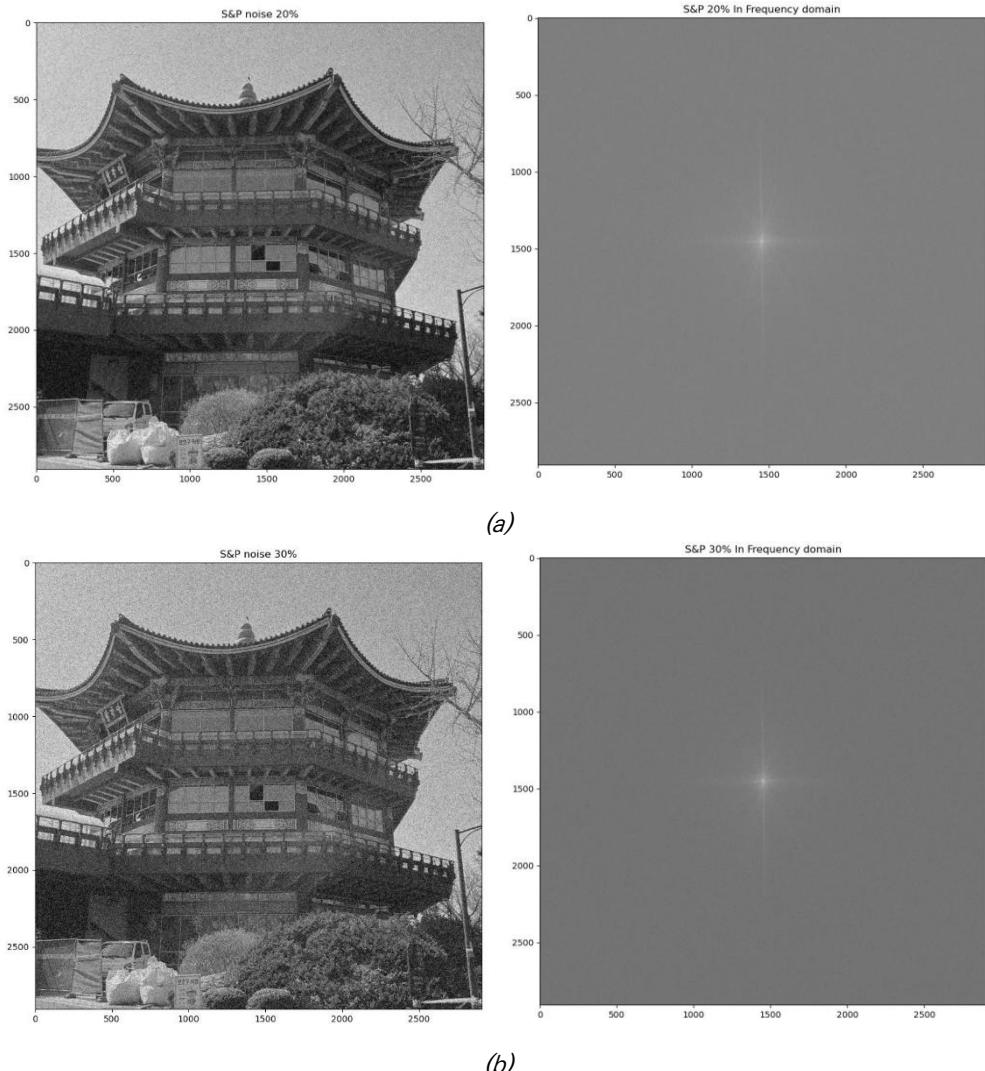
이렇게 주파수 영역에서 filtering을 수행하면, 이미지나 신호의 고주파 성분과 저주파 성분을 분리할 수 있다.

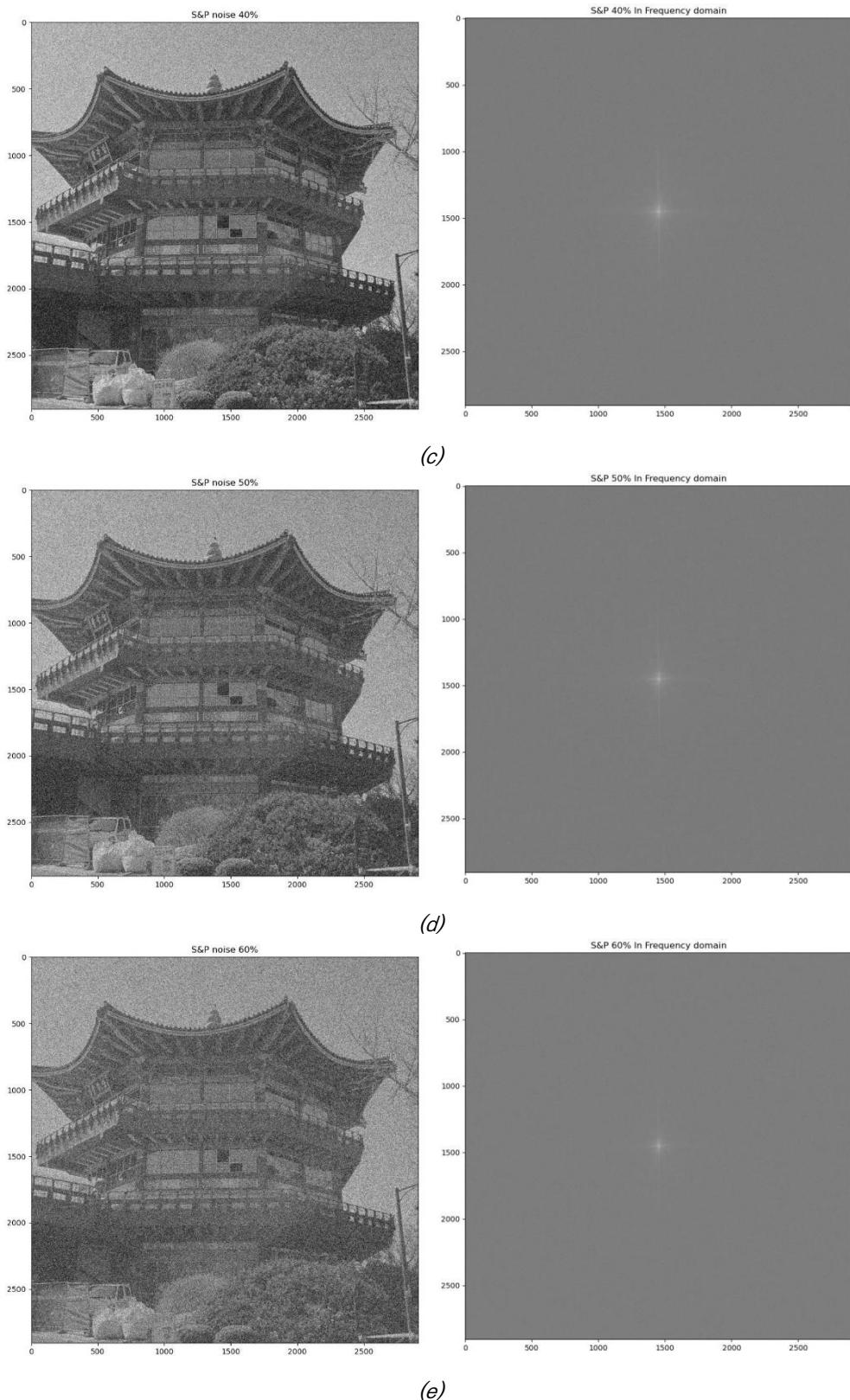
이렇게 분리된 고주파 성분과 저주파 성분을 각각 다른 방식으로 처리하여 이미지나 신호를 개선할 수 있다. 이 때 사용되는 filtering 기법에는 low-pass filter, high-pass filter, band-pass filter, band-stop filter 등이 있다.

Frequency domain filtering의 장점은 고속 Fourier transform(FFT) 알고리즘을 이용하여 빠르게 처리할 수 있으며, 주파수 대역에서 필요한 성분만을 선택하여 처리할 수 있다는 것이다. 또한, 필터링 과정에서의 노이즈나 왜곡이 최소화되므로 이미지나 신호의 품질이 개선된다. 또한, 이 방법은 이미지의 변환 및 필터링에 대한 일반적인 수학적 개념을 기반으로하기 때문에, 다른 분야에서도 쉽게 적용할 수 있다.

따라서, frequency domain filtering은 영상 처리나 신호 처리 분야에서 매우 중요한 기술로 자리 잡고 있으며, 다양한 분야에서 활용되고 있다.

Low-pass filter, Band-pass filter를 적용하기 앞서 S&P 잡음을 Fourier Transform을 수행해 Spatial domain을 Frequency domain으로 바꿔주었다.





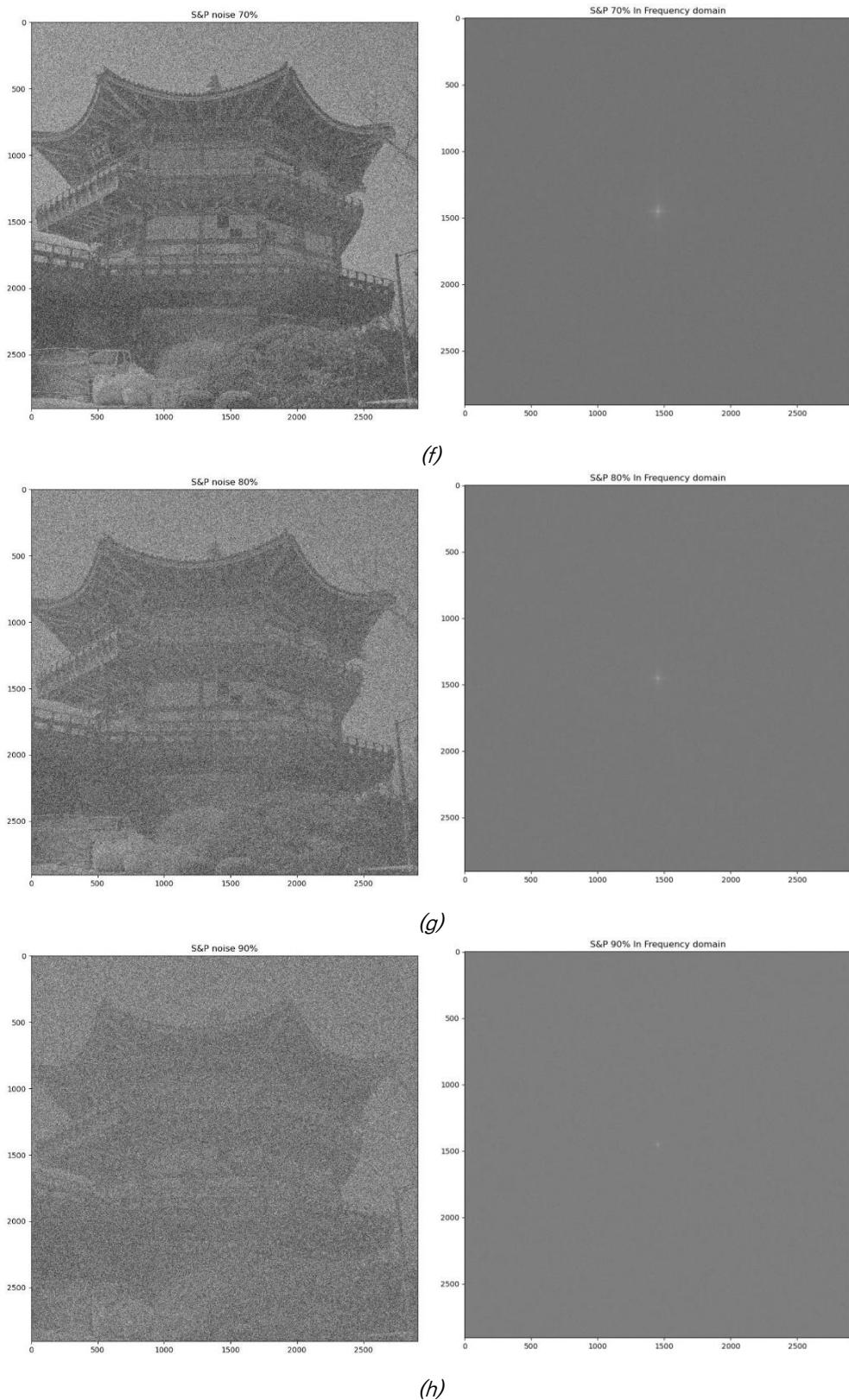


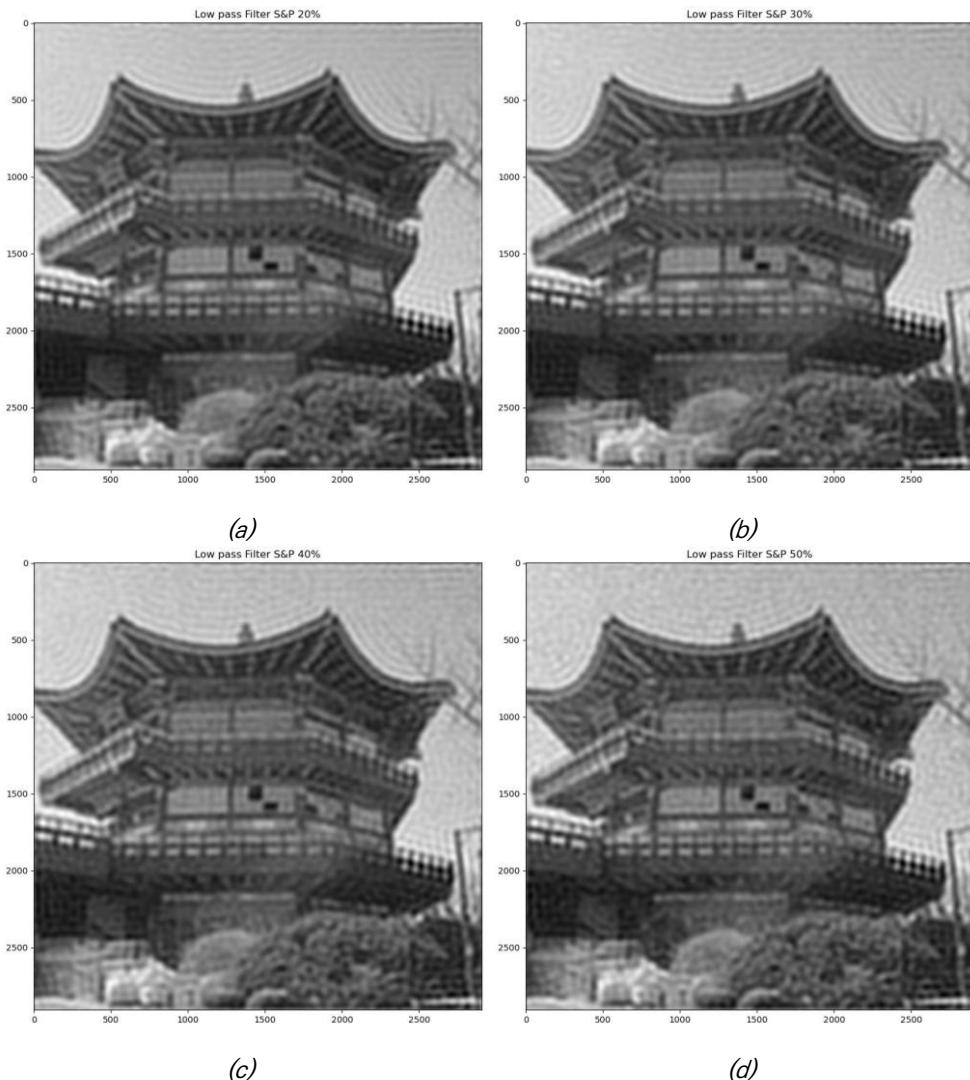
Fig 20. Salt and Pepper Noise Image In Frequency Domain (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

3.4.1 Low pass filter

주파수 영역에서의 low pass filter는 특정 주파수보다 낮은 주파수의 신호를 통과시키고, 높은 주파수의 신호를 차단하는 필터이다. 이러한 필터는 아날로그와 디지털 신호 처리에서 매우 일반적으로 사용되며, 신호에서 원치 않는 고주파 잡음을 제거하거나 필요한 주파수 성분을 추출하는 데 사용된다.

다른 유형의 low pass filter로는 대역폭을 정확하게 제어할 수 있는 Butterworth 필터, 저역통과 필터보다 더 급격하게 주파수를 차단할 수 있는 Chebyshev 필터, 주파수 응답이 정확하지만 계수가 복잡한 Elliptic 필터 등이 있다. 본 연구에서는 fourier transform을 실행하고 저주파수만 통과시키는 mask를 만들어 low pass filtering을 진행하였다. Mask의 크기는 사용한 라이브러리가 임의로 정한 값이다.

Low pass filter는 고주파 성분을 제거하여 smoothing 데에 효과적이기 때문에 앞서 S&P 잡음을 추가한 영상에서 잡음을 제거하기 위해 S&P 잡음 이미지를 사용해 Low pass filter를 적용해보려 한다.



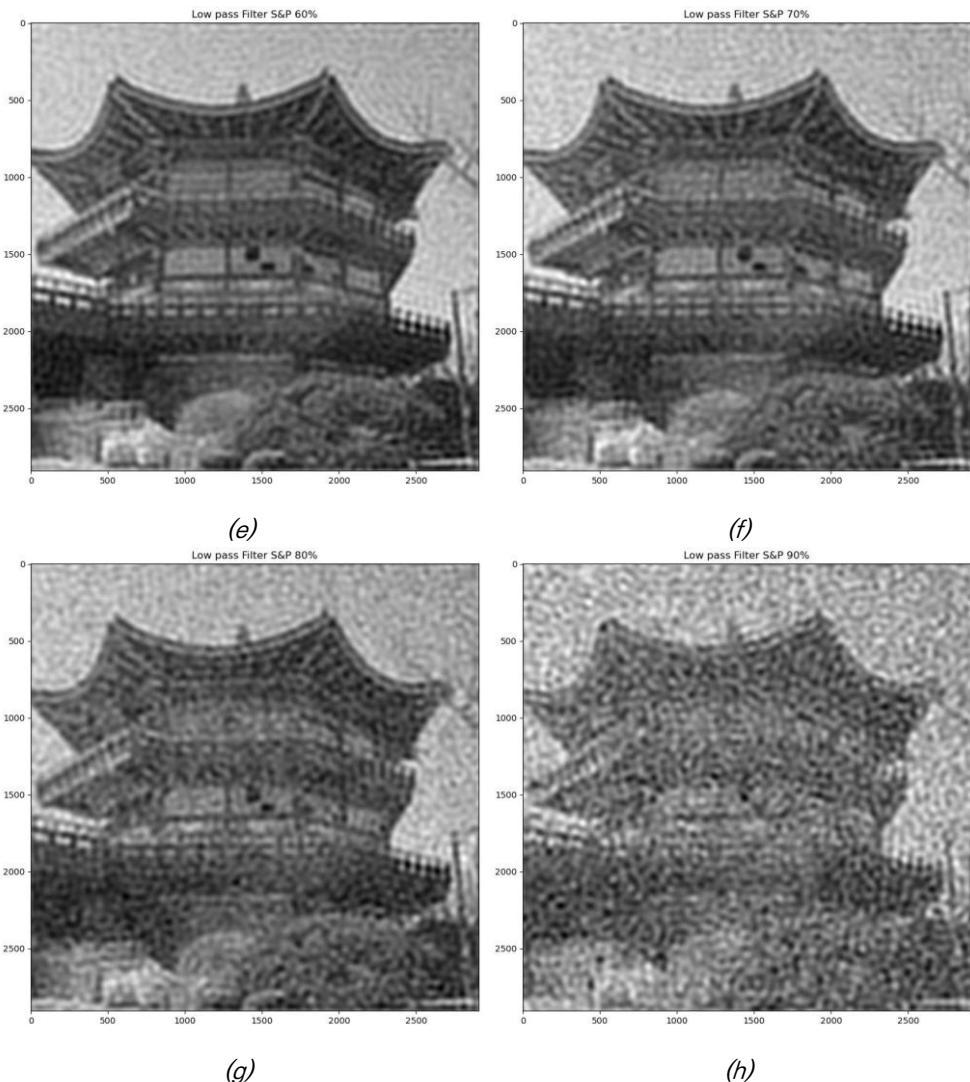


Fig 21. Low Pass Filter (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

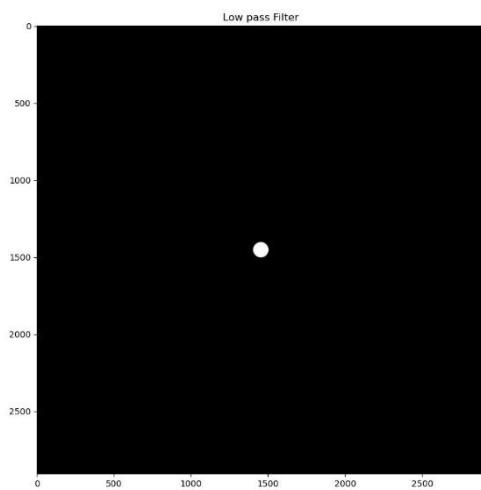


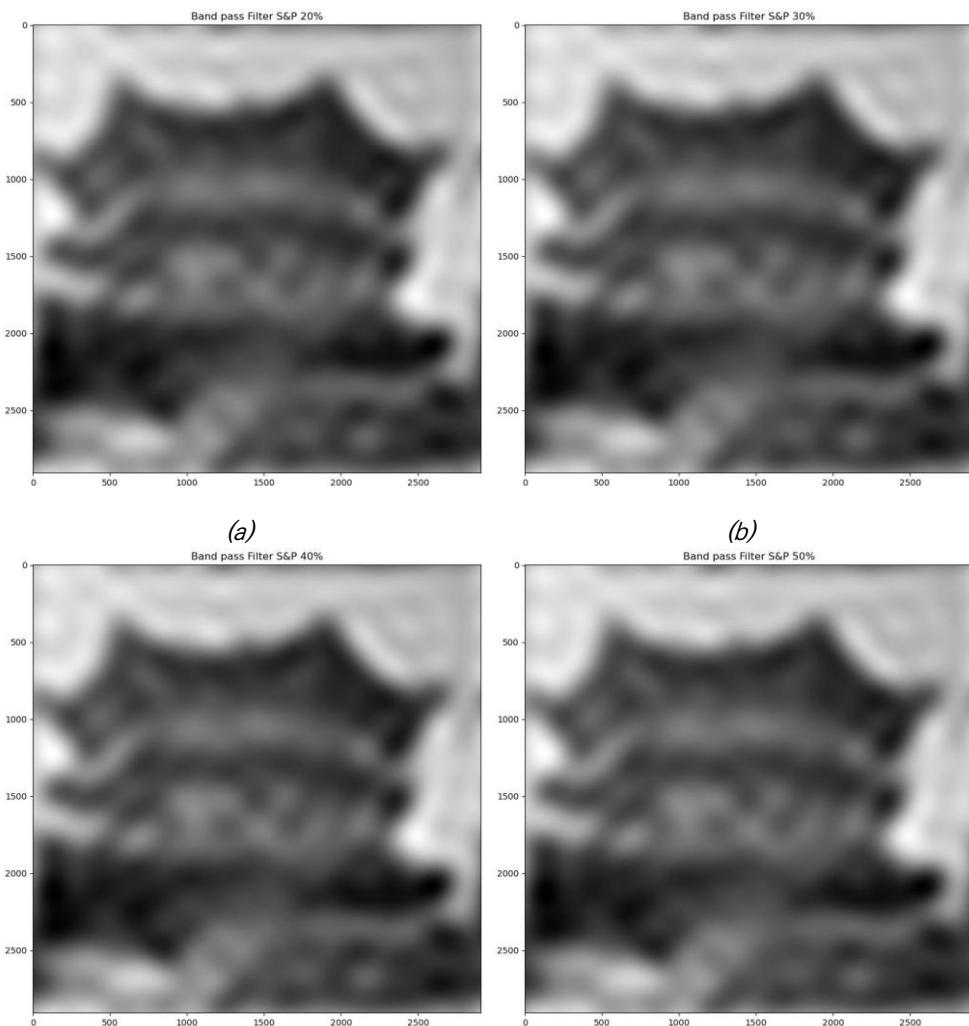
Fig 22. Low Pass Filter Mask

3.4.2 Band pass

주파수 영역에서의 band pass filter는 특정한 주파수 대역의 신호를 통과시키고, 그 외의 주파수 대역의 신호를 차단하는 필터이다. 이러한 필터는 음성, 이미지 및 비디오 신호 처리에서 주파수 대역을 분리하거나 추출하는 데 사용된다.

일반적으로 band pass filter는 주파수 대역의 하한과 상한을 정의하는 두 개의 저역통과 필터 또는 고역통과 필터를 결합하여 구현된다. 두 필터를 결합하면 주파수 대역 내부의 신호가 통과되고 주파수 대역 외부의 신호는 차단된다.

또한, low pass filter와 band pass filter는 유사한 구조를 가지고 있다. band pass filter는 low pass filter를 기반으로 만들어졌기 때문이다. 본 연구에서 low pass filter는 라이브러리에 의해 임의로 mask의 크기가 원의 형태로 정해졌으나 band pass filter는 연구자가 설정한 최적의 값으로 박스 모양의 mask 형태로 정해졌다는 차이가 존재한다. 그렇기에 Fourier transform을 실행하고 중심에서부터 20x20 pixel의 크기로 된 mask를 만들어 저주파수만 통과시키는 과정을 진행하였다.



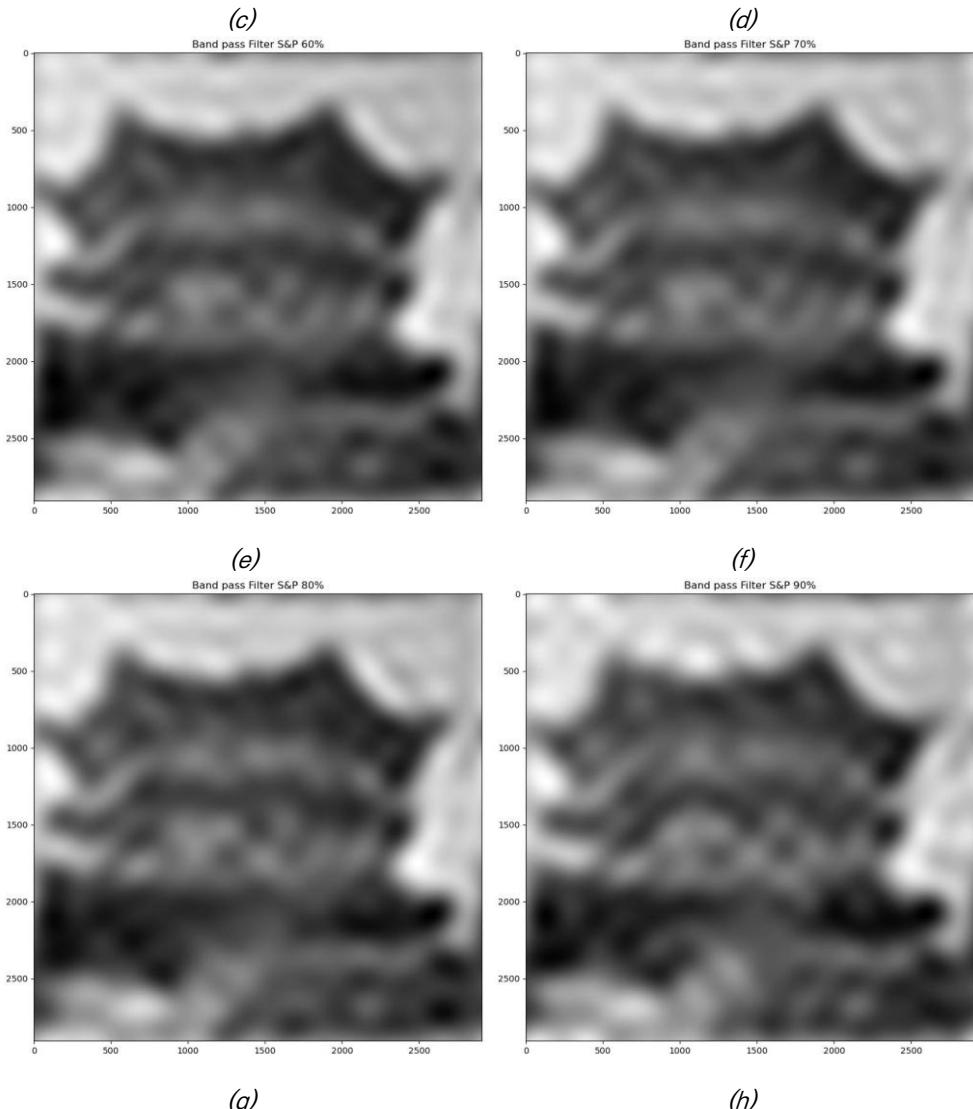


Fig 23. Band Pass Filter (a) $P=20\%$, (b) $P=30\%$, (c) $P=40\%$, (d) $P=50\%$, (e) $P=60\%$, (f) $P=70\%$, (g) $P=80\%$, (h) $P=90\%$

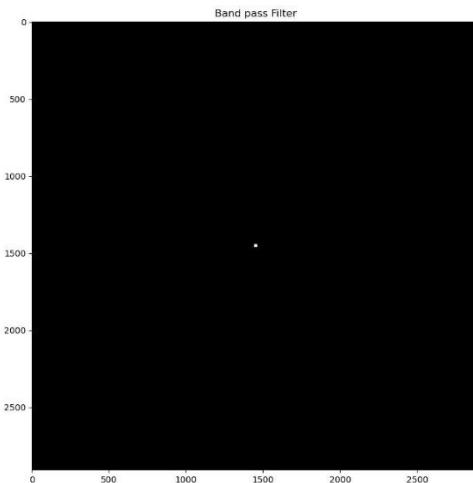


Fig 24. Low Pass Filter Mask

3.4.3 High pass filter

주파수 영역에서의 High pass filter(고역통과 필터)는 특정 주파수보다 높은 주파수의 신호를 통과시키고, 낮은 주파수의 신호를 차단하는 필터이다. 이러한 필터는 아날로그와 디지털 신호 처리에서 매우 일반적으로 사용되며, 신호에서 원치 않는 저주파 잡음을 제거하거나 필요한 고주파 성분을 추출하는 데 사용된다.

다른 유형의 high pass filter로는 대역폭을 정확하게 제어할 수 있는 Butterworth 필터, 고역통과 필터보다 더 급격하게 주파수를 차단할 수 있는 Chebyshev 필터, 주파수 응답이 정확하지만 계수가 복잡한 Elliptic 필터 등이 있다. 본 연구에서는 Fourier transform을 실행하고 고주파수만 통과시키는 mask를 만들어 high pass filtering을 진행하였다. Mask의 크기는 사용한 라이브러리가 임의로 정한 값이다.

High pass filter는 저주파 성분을 제거하여 sharpening을 데에 효과적이기 때문에 kernel의 크기가 9x9이고 sigma는 0인 Gaussian Blur를 실행한 영상에서 선명도를 높이고 경계 추출을 위해 Blurred 이미지를 사용해 High pass filter를 적용해보려 한다.

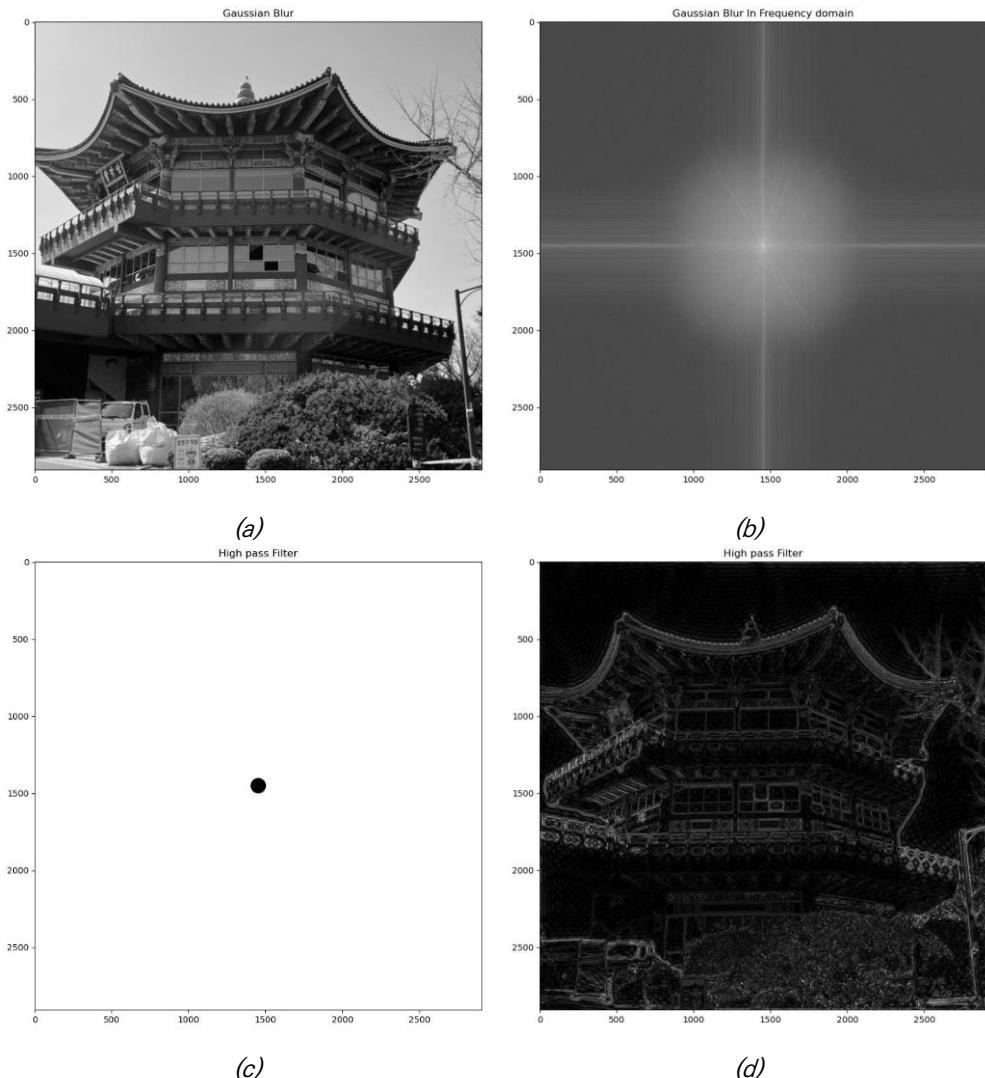


Fig 25. High Pass Filter (a) Gaussian filter Image, (b) Gaussian filter Image In Frequency domain, (c) High pass filter mask, (d) output

3.4.4 Band stop

주파수 영역에서의 band stop filter는 특정한 주파수 대역을 차단하고, 그 외의 주파수 대역은 통과시키는 필터이다. 이러한 필터는 신호 처리에서 특정 주파수 대역의 잡음을 제거하거나, 주파수 대역 내부의 신호를 제거하는 데 사용된다.

일반적으로 band stop filter는 두 개의 고역통과 필터 또는 저역통과 필터를 결합하여 구현된다. 이러한 필터는 주파수 대역의 하한과 상한을 정의하는 두 개의 저역통과 필터 또는 고역통과 필터를 결합하여 구현된다. 두 필터를 결합하면 주파수 대역 내부의 신호가 차단되고, 주파수 대역 외부의 신호는 통과된다.

High pass filter와 band stop filter는 서로 유사한 구조를 가지고 있다. 본 연구에서 high pass filter는 라이브러리에 의해 임의로 mask의 크기가 원의 형태로 정해졌으나 band pass filter는 연구자가 설정한 최적의 값으로 박스 모양의 mask 형태로 정해졌다는 차이가 존재한다. 그렇기에 fourier transform을 실행하고 중심에서부터 20x20 pixel의 크기로 된 mask를 만들어 고주파수만 통과시키는 과정을 진행하였다.

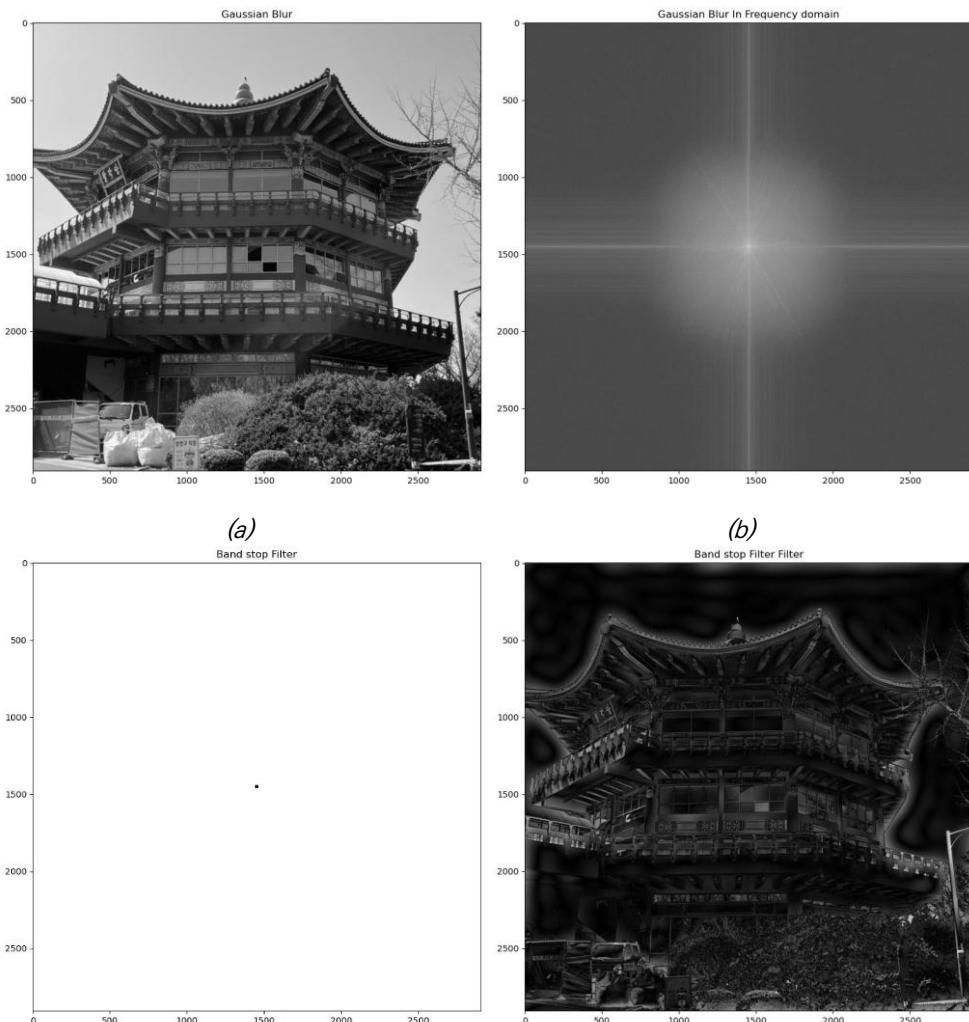




Fig 26. Band Stop Filter (a) Gaussian filter Image, (b) Gaussian filter Image In Frequency domain, (c) High pass filter mask, (d) output

4. 결과 및 코드 분석

4.1 Salt & pepper noise 가중치에 따른 변화

Pseudo-Code 1: Salt and Pepper noise

Input: Image, Probability

Output: Noisy Image

1. Copy Image
 2. *IF*- image shape == 2 (black, white)
 3. *THEN*- black:0, white:255
 4. *ELSE* - black: [0,0,0], white: [255,255,255] #RGB
 5. probs: 2 array random number
 6. *IF* prob *LT* prob/2 *THEN* black
 7. *ELIF* prob *GT*(1 – prob/2) *THEN* white
-

Table 1. Salt and Pepper noise

4.2 Spatial filtering

Pseudo-Code 2: Classic Smoothing Filtering

Input: Noisy Image

Output: Filtered Image

1. Load Original Image (RGB)
 2. Convert it to GRAY scale
 3. Import cv2 library
 4. *FOR LOOP.*
 5. Add noise to Image for prob 20 to 100
 6. Do BLUR, GaussianBlur, MedianBlur, bilateralFilter
 7. Save Image
 8. *END*
-

Table 2. Classic Smoothing Filtering

Pseudo-Code 3: Proposed algorithm for Smoothing Filter

Input: Noisy Image

Output: Filtered Image

1. *DEFINE* custom_function(img):
 2. *IF* img[i,j] == Not noise *THEN* img[i,j]=1
 3. *ELSE* make cross kernel, count the number of no-noise, list of no-noise(NM1)
 4. *IF* count == 2, *THEN*, img[i,j] = 0.5*(min(NM1) + max(NM1))
 5. *ELIF* count == 1, *THEN*, img[i,j] = NM1
 6. *ELSE*, create 3 by 3 kernel, count the number of no-noise, list of no-noise(NM2)
 7. *IF* count is odd *THEN*, median(NM2)
 8. *ELIF* count is even *THEN*, 0.5*(NM2[k/2] + NM2[k/2-1])
 9. *ELSE* create 6 by 6 kernel, noise density, count
 10. *IF* density *LT* 0.5, *THEN*, calculate weight distance
 11. *ELIF* density *GT* 0.5, process 3 by 3 kernel at line 6
 12. *IF* count == 0 *THEN*, calculate precalculated equation.
-

Table 3. Proposed algorithm for Smoothing Filter

본 연구를 통해 다양한 공간 필터링 기법들과 주파수 영역 필터링 기법들을 적용해 보았다. 각 필터들에 대한 성능 분석을 하고자 다양한 평가 지표들을 활용하고자 한다. 우선 Smoothing 필터들에 대해서 S&P 잡음을 효과적으로 지울 수 있는지에 대한 평가를 하기 위해 MSE와 PSNR을 계산하여 이들을 비교하였다. PSNR은 원본 영상과 필터링이 수행된 두 영상을 가지고 계산될 수 있으며 이 값의 커질수록 두 영상의 유사도는 커진다, 즉 잡음이 잘 제거됐다고 판단할 수 있다.

$$MSE = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N \{O(i,j) - I(i,j)\}^2 \quad (\text{식 4.1})$$

$$PSNR = 10 \log_{10} \frac{255^2}{MSE} \quad (\text{식 4.2})$$

4.2.1 Smoothing 필터 간의 비교

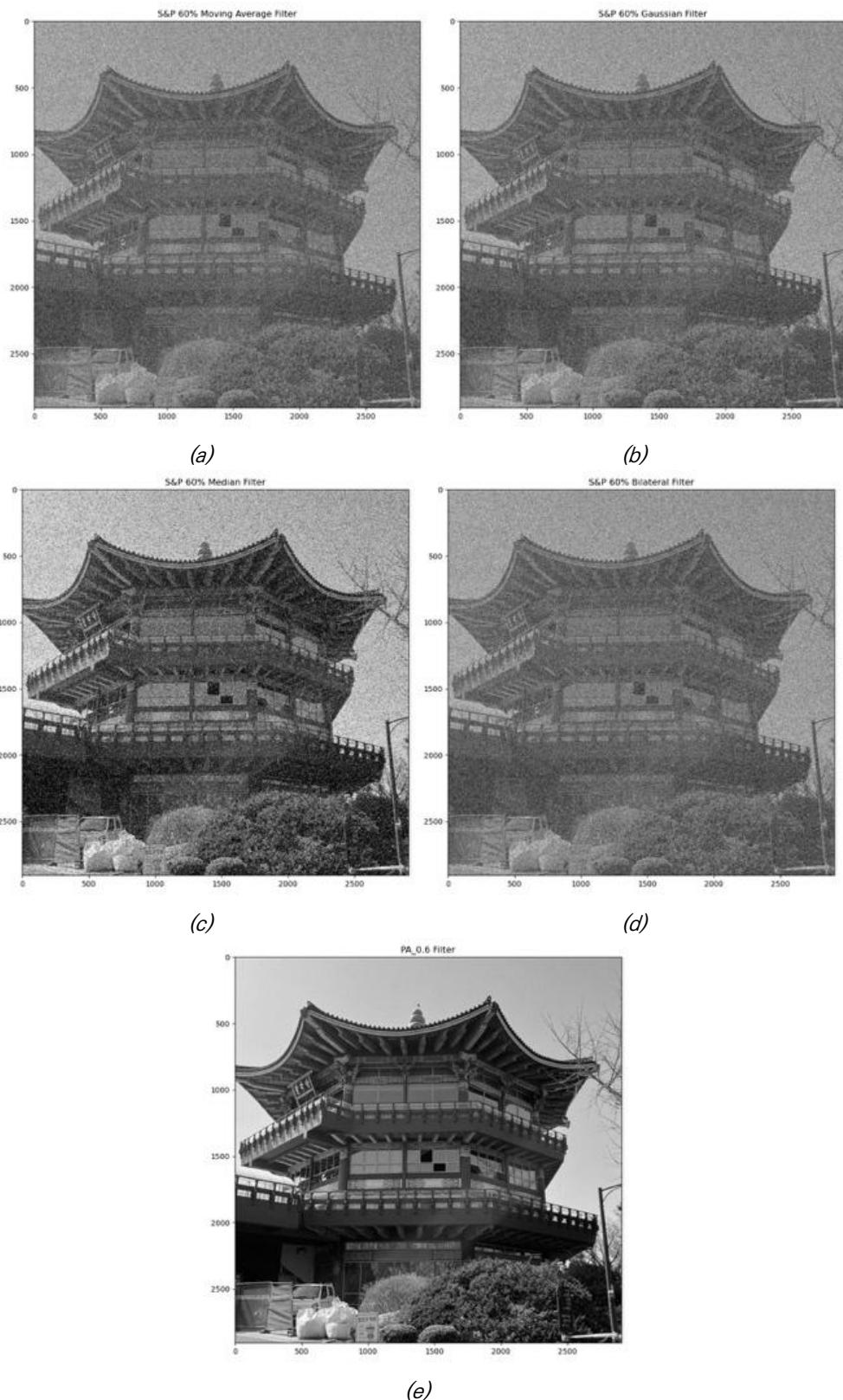


Fig 27. Smoothing Filters for 60% Noise (a) Moving Average filter, (b) Gaussian filter, (c) Median filter mask, (d) Bilateral filter, (e) PA filter

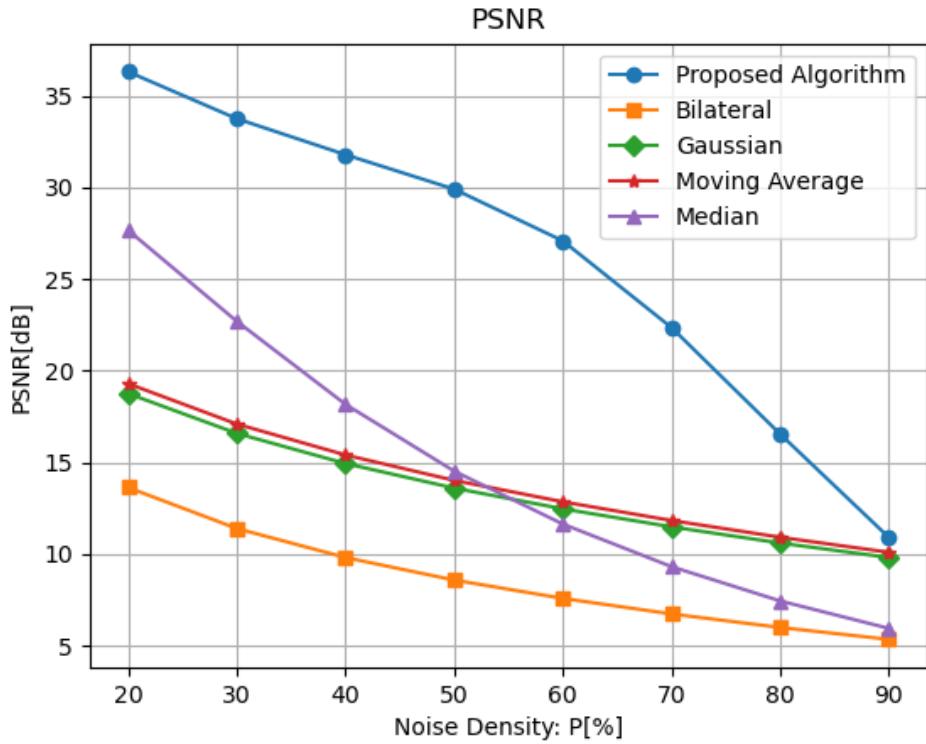


Fig 28. PSNR of smoothing filters

| Density [%] \ Filters | Bilateral | Gaussian | Moving Average | Median | PA |
|-----------------------|-----------|----------|----------------|---------|---------|
| 20 | 13.62dB | 18.75dB | 19.29dB | 27.67dB | 36.32dB |
| 30 | 11.39dB | 16.58dB | 17.09dB | 22.70dB | 33.75dB |
| 40 | 9.80dB | 14.93dB | 15.40dB | 18.16dB | 31.79dB |
| 50 | 8.57dB | 13.59dB | 14.01dB | 14.52dB | 29.91dB |
| 60 | 7.57dB | 12.46dB | 12.84dB | 11.63dB | 27.08dB |
| 70 | 6.72dB | 11.47dB | 11.82dB | 9.30dB | 22.35dB |
| 80 | 5.98dB | 10.59dB | 10.91dB | 7.43dB | 16.53dB |
| 90 | 5.33dB | 9.80dB | 10.09dB | 5.93dB | 10.89dB |

Table 4. PSNR dB values

Table 4와 Fig 27을 통해 다양한 Smoothing 필터들에 대한 성능을 확인할 수 있다. S&P잡음 제거에 효과적으로 알려진 Median 필터의 경우 잡음 밀도 50% 구간까지는 양호한 모습을 보였으나, 그 이상의 잡음 밀도에 대해서는 Gaussian 필터와 이동 평균 필터에 비해 떨어지는 것을 확인할 수 있다. 제안된 PA 필터는 모든 구간에서 기타 필터에 비해 좋은 성능을 보였다. 특히 잡음 밀도 60% 구간에서 이동 평균 필터와 비교해서 15.45dB 정도 더 뛰어난 성능을 보였다. 각 편차를 비교해 보았을 때 60% 구간에서 PA 필터가 제일 효과가 좋은 것을 확인했다. 반면에 Bilateral 필터의 경우 잡음 제거 효과가

뛰어나지는 못했으나, 경계 부분의 영상 특징을 훌륭히 보존했다. 만약 S&P 잡음이 20% 이하로 적을 경우 시간적으로는 Median 필터를 사용하는 것이 효율적으로 보인다. Median 필터는 매우 훌륭한 잡음 제거 효과를 보였는데, 일상적인 영상 잡음 제거를 수행해야 할 때 활용될 수 있을 것 같다. 반면에 잡음을 최소화시키고 이를 통한 영상의 품질을 향상시키고 싶으면 시간이 걸리더라도 PA 필터를 적용하는 것이 적절하다고 생각한다. 따라서 다양한 목적, 상황에 맞게 필터를 적용시키면 될 것 같다.

4.2.2 sharpening 필터 간의 비교

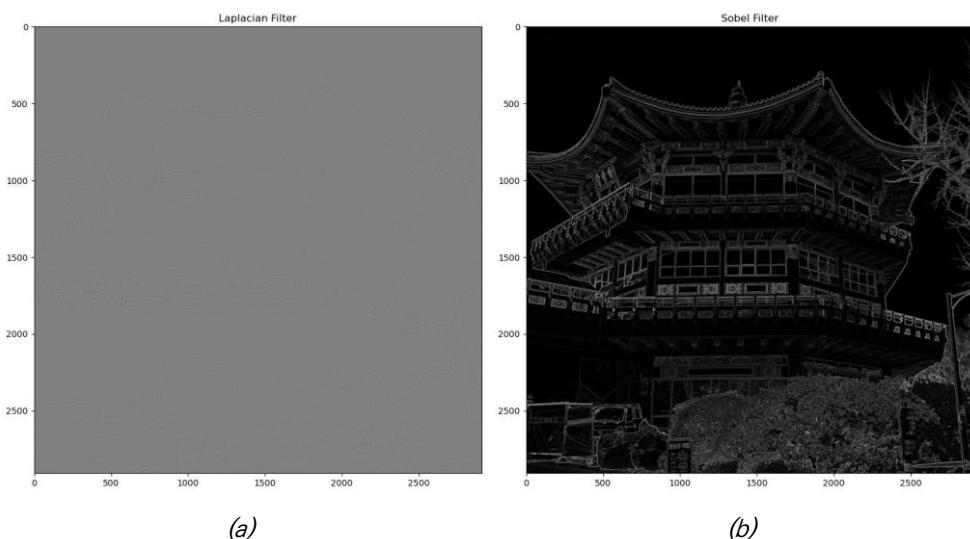
Pseudo-Code 4: Sharpening Filters

Input: Blurred Image

Output: Filtered Image

1. Load Original Image (RGB)
 2. Convert it to GRAY scale
 3. Import cv2 library
 4. *FOR LOOP.*
 5. *DO* Gaussian Blur to image, *END*
 6. *DO* Sobel to blur image, *END*
 7. *DO* Unsharped mask, weight calculation, *END*
 8. *DO* CannyFilter for Blur image, *END*
 9. Plot and save the Image
 10. *END*
-

Table 5. Sharpening Filters



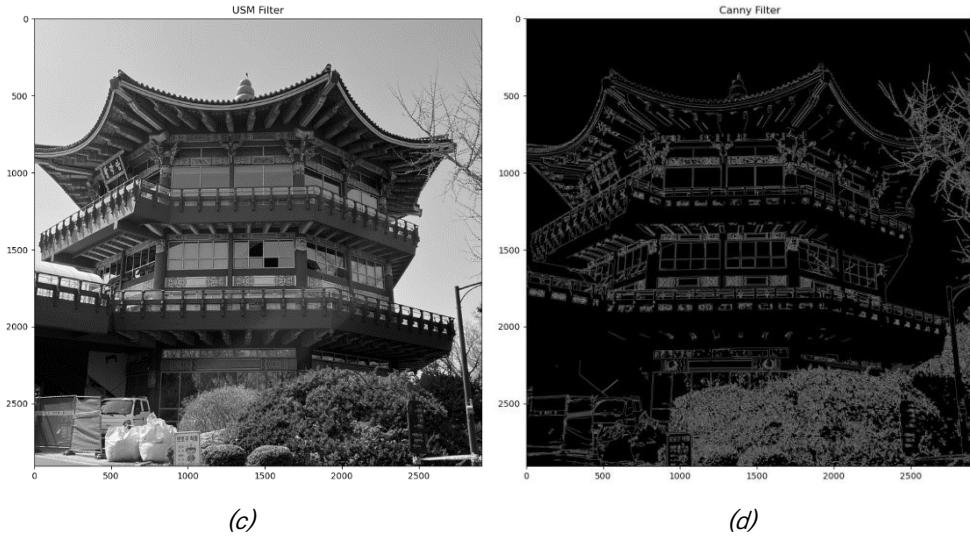


Fig 29. Sharpening Filters (a) Laplacian filter, (b) Sobel filter, (c) USM filter mask, (d) Canny filter

Sharpening filtering을 통해 정규화 이전의 다음 4가지와 같은 그림을 얻었다. 우선 Laplacian 필터의 경우 미분 값을 사용하여 높은 주파수 성분을 더욱 더 강화시키므로 경계를 제외한 나머지 부분에 대해서 잡음이 더해진 것을 확인할 수 있다. 두번째 그림의 Sobel 필터의 경우 수평 및 수직 경계에 대한 부분이 조금 더 강조된다. 또한 명암 구분이 잘 되어있어 경계를 뚜렷이 관찰할 수 있는 것을 확인할 수 있다. USM 필터는 이미지를 더욱 선명하게 만들기 위해 사용되는 필터로, 원본 이미지에서 Blur 처리된 이미지를 뺀 뒤 원본 이미지에 다시 추가하여 이미지를 선명하게 만든다. 그림에서 볼 수 있듯이 원본 영상과 거의 유사하지만 선명도가 조금 더 올라간 것을 확인할 수 있다. 마지막으로 Canny 필터의 경우 경계 추출을 하는 필터 중 성능이 매우 우수한 것 중 하나로, 이 알고리즘은 Gaussian Blur를 사용하여 이미지를 미리 Smoothing 함으로써 잡음을 줄어들어 영상 품질이 우수하다.

4.3 frequency domain filtering

Pseudo-Code 5: Frequency domain filtering

Input: Noisy Image, Blurred Image

Output: Filtered Image

1. Load Original Image (RGB)
 2. Convert it to GRAY scale
 3. Import cv2 library
 4. *FOR LOOP.*
 5. *DO* Gaussian Blur to image, *END*
 6. *DO* Sobel to blur image, *END*
 7. *DO* Unsharped mask, weight calculation, *END*
 8. *DO* CannyFilter for Blur image, *END*
-

9. Plot and save the Image

10. END

Table 6. Frequency domain filtering

4.3.1 필터 간의 비교

1) Low pass vs. Band pass

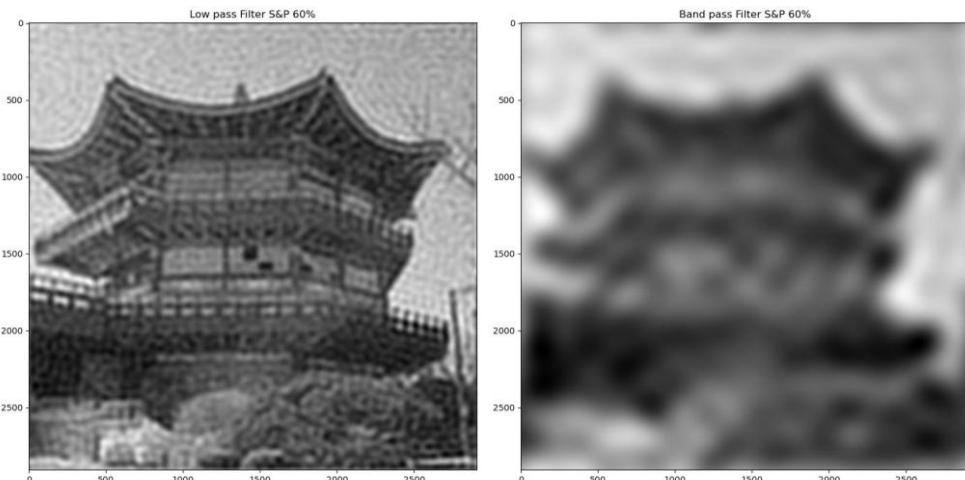
Pseudo-Code 6: Low pass and Band pass filter

Input: Noisy Image

Output: Filtered Image

1. Load Original Image (RGB)
 2. Convert it to GRAY scale
 3. Import cv2, numpy, matplotlib library
 4. *DEFINE* low_pass(img) and band_pass(img)
 5. Fourier Transform img
 6. Get image shape
 7. Make mask (Different mask used Low/Band)
 8. Convolution mask and FFT image
 9. IFFT Convolution output
 10. Abs
 11. Save Image
 12. END
-

Table 7. Low pass and Band pass filter



(a) (b)

Fig 30. (a) Low pass filter, (b) Band pass filter

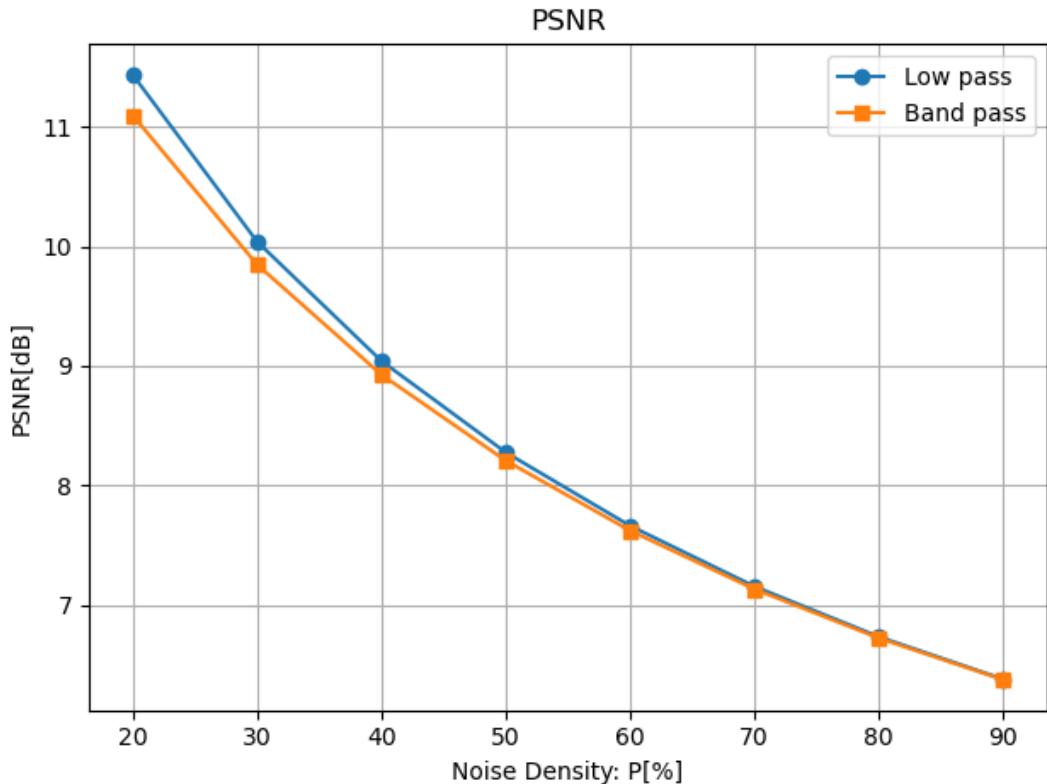


Fig 31. PSNR of Low pass and Band pass filter

| Density [%] \ Filters | Low Pass | Band Pass |
|-----------------------|----------|-----------|
| 20 | 11.10dB | 11.44dB |
| 30 | 9.85dB | 10.04dB |
| 40 | 8.93dB | 9.04dB |
| 50 | 8.21dB | 8.75dB |
| 60 | 7.62dB | 7.66dB |
| 70 | 7.13dB | 7.15dB |
| 80 | 6.72dB | 6.73dB |
| 90 | 6.37dB | 6.38dB |

Table 8. PSNR dB values

Table 8과 Fig 28을 통해 Low pass filter와 Band pass filter에 대한 성능을 확인할 수 있다. 앞서

filtering 진행했던 공간영역의 filter들과는 다르게 전체적으로 낮은 PSNR 값을 보인다. 실제로 잡음을 제거시킨 결과물을 봐도 원본 영상과 매우 거리가 먼 모습이라는 것을 알 수 있다. 또한, 두 filter를 비교해보자면 거의 차이가 없음을 알 수 있다. 이 차이 조차도 filtering에 사용된 mask의 크기가 다르기 때문이라고 볼 수 있다. 이러한 점들을 미루어 보아 Low pass filter와 Band pass filter는 이미지의 잡음을 감소하는 데에 큰 차이가 존재하지 않고, 그 성능조차 그렇게 좋지 않다는 것을 알 수 있었다.

2) High pass vs. Band Stop

Pseudo-Code 7: High pass and Band stop filter

Input: Blurred Image

Output: Filtered Image

1. Load Original Image (RGB)
 2. Convert it to GRAY scale
 3. Import cv2, numpy, matplotlib library
 4. *DEFINE* high_pass(img) and band_stop(img)
 5. Fourier Transform img
 6. Get image shape
 7. Make mask (Different mask used Low/Band)
 8. Convolution mask and FFT image
 9. IFFT Convolution output
 10. Abs
 11. Save Image
 12. *END*
-

Table 9. High pass and Band stop filter

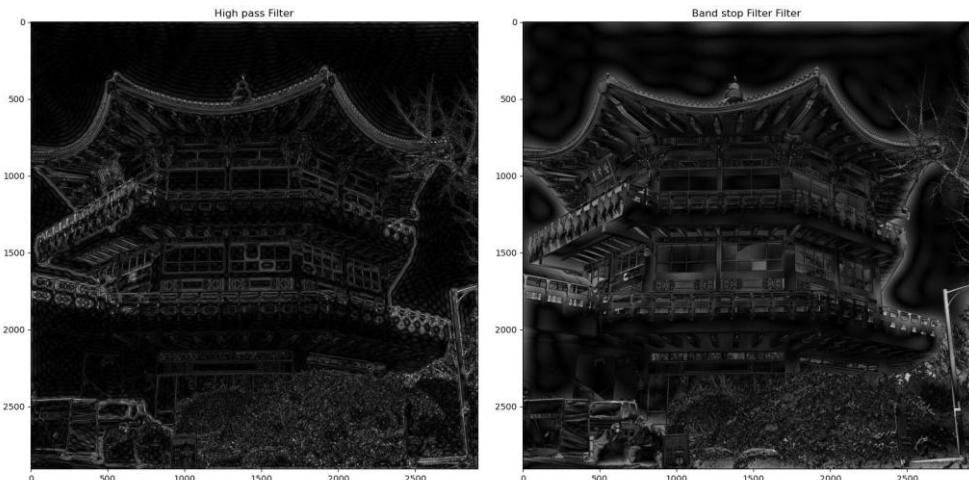




Fig 32. (a) High pass filter, (b) Band stop filter

High Pass와 Band stop 필터는 서로의 본질적인 특성이 같다. 주파수 영역에서 신호를 처리할 때 특정 성분을 통과시키고, 특정 성분은 차단한다. 이때 High-pass 필터는 낮은 주파수 성분은 차단하는 반면 Band stop 필터는 특정 주파수 범위의 성분을 차단하고 나머지 성분을 통과시킨다. 본 연구에서는 Band stop 필터를 high pass 필터와 같이 낮은 주파수 성분은 차단하도록 마스크를 설정했다. 위의 그림에서도 확인할 수 있듯이 이 둘 필터 모두 경계(Edge) 부분을 잘 검출하는 것을 확인할 수 있다. 반면 경계가 아닌 다른 이미지 특성에 대해서는 검정색으로 표현되는 것을 확인할 수 있다.

4.4 Smoothing Effect : Spatial vs. Frequency

| Density [%] \ Filters | Low Pass | Band Pass | Moving Average | Median | PA |
|-----------------------|----------|-----------|----------------|---------|---------|
| 20 | 11.10dB | 11.44dB | 19.29dB | 27.67dB | 36.32dB |
| 30 | 9.85dB | 10.04dB | 17.09dB | 22.70dB | 33.75dB |
| 40 | 8.93dB | 9.04dB | 15.40dB | 18.16dB | 31.79dB |
| 50 | 8.21dB | 8.75dB | 14.01dB | 14.52dB | 29.91dB |
| 60 | 7.62dB | 7.66dB | 12.84dB | 11.63dB | 27.08dB |
| 70 | 7.13dB | 7.15dB | 11.82dB | 9.30dB | 22.35dB |
| 80 | 6.72dB | 6.73dB | 10.91dB | 7.43dB | 16.53dB |
| 90 | 6.37dB | 6.38dB | 10.09dB | 5.93dB | 10.89dB |

Table 10. PSNR dB values

Smoothing filter와 Low, Band pass filter 모두 이미지의 노이즈를 없애는데 효과가 있다. 그렇기에 각각의 filter를 비교하기 위해 사용했던 PSNR 값을 가지고 공간영역에서의 Smoothing filter와 Low pass filter를 비교해보자 한다.

공간영역의 Smoothing filter는 대표적으로 Moving Average, Median, PA filter의 값을 가져오고 주파수 영역은 Low, Band pass filter의 값을 사용했다. Table 10을 보면 값 자체가 크면 20dB이상으로 차이가 남을 수 있다. 가져온 Median, PA filter 등과 같은 경우는 최대한 원 데이터가 손상되지 않게 smoothing을 하는 거라면 Low, Band pass filter는 특정 주파수의 값을 아예 차단해 smoothing을 하는 것이기 때문에 어떻게 보면 당연히 성능이 Spatial filtering이 잡음을 없애는 데엔 Frequency filtering보다 좋게 나올 수밖에 없다. 그렇기에 이런 상황에서는 Frequency domain의 Low, Band pass filter보다는 Spatial domain의 Moving Average, Median filter 등을 사용하는 것이 더 적합할 것 같다는 분석이 도출되었다.

4.5 Chat GPT와의 비교

본 연구를 통해 프로그래밍 언어에 대한 많은 학습이 이루어졌다. 특히 Google, StackOverFlow와 ChatGPT 등의 다양한 플랫폼을 통해 얻어진 코드를 분석하면서 많은 도움을 받았다. 최근, 인공지능 기술의 산물이라 평가되는 ChatGPT의 경우 장단점이 존재했다. 우선 이 도구를 이용하면 빠른 시간 안에 전체 코드는 아니지만 많은 코드 조각들을 획득할 수 있었다. 또한, 2021년의 정보까지 학습이 되어있어, 웬만한 개념과 지식에 대해서는 우리보다 뛰어났던 것 같다. GPT의 경우 오픈소스 GitHub로부터 많은 사람들의 프로그래밍 스타일과 패턴을 학습했기 때문에, 유연하게 작성이 가능하다는 것을 확인했다. 사용자가 작성한 질문을 토대로 답변을 작성해주지만, 질문의 형식이 조금만 바뀌어도 결과가 매우 달랐다. 프로그래밍 언어 역시 마찬가지였다. 한 문제에 대해 다양한 결과물을 내놓을 수 있었고, 이를 사용자들이 이해하기 쉽게 예제 코드까지 넣어주면서 설명을 해준다.

하지만 GPT의 단점도 분명 존재했다. 질문이 정확하지 않으면 사용자 원하는 솔루션과 다른 답을 내놓았다. 또한 항상 완벽한 코드를 작성하지 않는 것을 미루어 보아, 모델의 허점이 많이 존재하는 것 같다. 2021년 9월까지의 지식을 기반으로 학습이 되어있어 최신 연구 및 지식에 대해 무지한 경우가 많았으며, 사용자의 현 상황에 대한 맥락 설명이 충분하지 않으면 전혀 다른 코드를 가져다 준다. 이번 연구에서도 GPT 활용의 장단점을 겪을 수 있었는데, PSNR에 대한 함수를 답변 받았지만 실제 PSNR 구하는 식과는 다른 함수를 내놓아 결과가 다르게 나왔다. 반면에, PA(Proposed Algorithm)의 코드를 작성한 후 실행했으나, TypeError를 포함한 다양한 버그로 인해 코드 실행까지의 과정이 길었다. 이러한 어려움에 부딪혔으나 GPT가 Error에 대한 구체적인 해결 방안과 수정 코드를 작성해줘서 버그를 잡을 수 있었다. 최근 GPT 모델이 스스로 학습하고 피드백 한다는 논문이 발표되었다. 앞으로 AI 모델을 적극적으로 잘 활용하는 능력이 요구될 것으로 예상된다.

The screenshot shows a user interaction with a large language model. The user asks, "Can you define a code for PSNR(Peak Singal to Noise ratio)." The model responds with a Python code snippet:

```
arduino
import numpy as np

def psnr(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if mse == 0:
        return float('inf')
    return 20 * np.log10(255.0 / np.sqrt(mse))
```

Below the code, there is a note explaining its assumptions:

This implementation assumes that the input images are represented as numpy arrays and that the pixel values are in the range [0, 255]. The `psnr` function calculates the mean squared error (MSE) between the original and compressed images, and then uses the MSE to compute the PSNR.

Note: The `log10` function is used to calculate the logarithm with base 10.

The screenshot shows a Chat GPT interface. The user asks, "Which is more accurate? Between your PSNR function, and cv2.PSNR function?" The AI responds that the OpenCV library's `cv2.PSNR` function is likely more accurate and reliable than a custom implementation. It explains that OpenCV is a well-established library developed by a large community of experts, making its functions well-tested and optimized. On the other hand, the AI's previous implementation is simple and straightforward but might not be as robust or optimized. The AI concludes by recommending the use of `cv2.PSNR` from the OpenCV library for better accuracy and reliability.

Fig 33. Compare with Chat GPT

5. 결론 및 향후 연구

공간 필터링부터 주파수 영역에 대한 필터링까지 다양한 기법들을 연구한 결과 각 방법에 대한 특징들과 장단점에 대해서 파악할 수 있었다. S&P 잡음을 최소화시키는 필터들의 성능을 비교하기 PSNR 지표를 활용하여 분석했다. 비교 결과, 제안된 PA 기법은 잡음 제거에 뛰어난 성능을 보이는 것을 확인했다. Median 필터 역시 특정 잡음 밀도 이하로는 우수한 성능을 보여줬다. Sharpening 필터링의 경우 각 기법을 적용한 영상들을 얻었으나, 경계 부분 특징들을 비교하고 성능을 비교하는 방법을 찾기 어려웠다. 또한 결과물이 smoothing 필터링 결과들과는 다르게 원본 영상과 유사하면서 선명도가 높은 영상이 있는 반면, 경계 부분만 추출된 영상들이 있어 이들을 평가하는데 많은 어려움이 있었다. Gaussian Blur을 적용할 때 잡음 strength (표준 편차) 값을 증가시키면서 Sharpening 필터들의 결과가 안 좋아지는 것을 확인했지만 이를 PSNR, SSIM, RMSE 등의 기법을 통해 비교했을 때는 실제 결과와는 맞지 않는다는 결론을 내렸다. 주파수 필터링의 Low-pass 내부의 마스크 형태에 따라 low, band pass로 나눴다. 하지만 크기의 차이가 많이 없어 PSNR 결과에는 큰 영향이 없었다. High-pass와 band stop의 경우에도 기존과 마찬가지로 진행 완료하였다. 이번 연구에서는 Frequency가 거의 유사한 밴드에서 진행해서 아쉬움이 있었다. 다음 연구에서는 다양한 주파수 대역을 가진 이미지에서 원하는 주파수 대역 등을 추출하는 과정을 추가로 해보고 싶다. 또한 혹시 가능하다면 공간 영역과 주파수 영역을 같이 처리하는 기법에 대해서도 추가적으로 연구하고 싶다.

6. 첨언

본 연구는 세종대학교 에너지자원공학과 유희영과 지구자원시스템공학과 이가람에 의해 공동 진행되었다.

코드작성, 논문작성, 검수를 모두 함께하여 역할에 대해 이분법적으로 나누기 어렵다는 점 참고 부탁드립니다.

7. 참고문헌

- [1] Seo, Suyoung. "Evaluation of Denoising Filters Based on Edge Locations." *Korean Journal of Remote Sensing* 36.4 (2020): 503-513.
- [2] Hong, Sang-Woo, and Nam-Ho Kim. "A Study on Removal of Salt and Pepper Noise using Deformable Masks Depending on the Noise Density." *Journal of the Korea Institute of Information and Communication Engineering* 19.9 (2015): 2173-2179.
- [3] Yinyu, Gao, and Nam-Ho Kim. "A Study on Improved Denoising Algorithm for Edge Preservation in AWGN Environments." *Journal of the Korea Institute of Information and Communication Engineering* 16.8 (2012): 1773-1778.
- [4] Kwon, Se-Ik, and Nam-Ho Kim. "Salt and Pepper Noise Removal using Linear Interpolation and Spatial Weight value." *Journal of the Korea Institute of Information and Communication Engineering* 20.7 (2016): 1383-1388.
- [5] gutierrezps. 'cv2_noise.py', Github. (2023)
<https://gist.github.com/gutierrezps/f4ddad3bbd2ad5a9b96e3c06378e28b4>
- [6] Lee, Dae-Geon · Shin, Young-Ha · Lee, Dong-Cheon. "Training Performance Analysis of semantic Segmentation Deep Learning Model by Progressive Combining Multi-modal Spatial Information Datasets." *Journal of the Korean Society of Surveying, Geodesy* 40.2 (2022): 91-108
- [7] Ji-Hyeon Baek, Nam-Ho Kim. "Salt and Pepper Noise Removal using Processed Pixels." *Journal of the Korea Institute of Information and Communication Engineering* 23.9 (2019): 1076-1081
- [8] adenarayana. "Python#007 Frequency Domain Filter Gaussian Filter.py". Github. (2022)
<https://github.com/adenarayana/digital-image-processing/blob/main/Python%23007%20Frequency%20Domain%20Filter%20Gaussian%20Filter.py>
- [9] OpenCV-Tutorial. https://docs.opencv.org/4.x/d9/df8/tutorial_root.html
- [10] Chat GPT. <https://chat.openai.com/>