OpenCV 없이 구현한 크로스플랫폼 + Python 기반의 영상처리

Grayscale Image Processing

[intel] 엣지 AI SW 아카데미 (4 기)
정용재

프로젝트 개요

- 프로젝트 목표

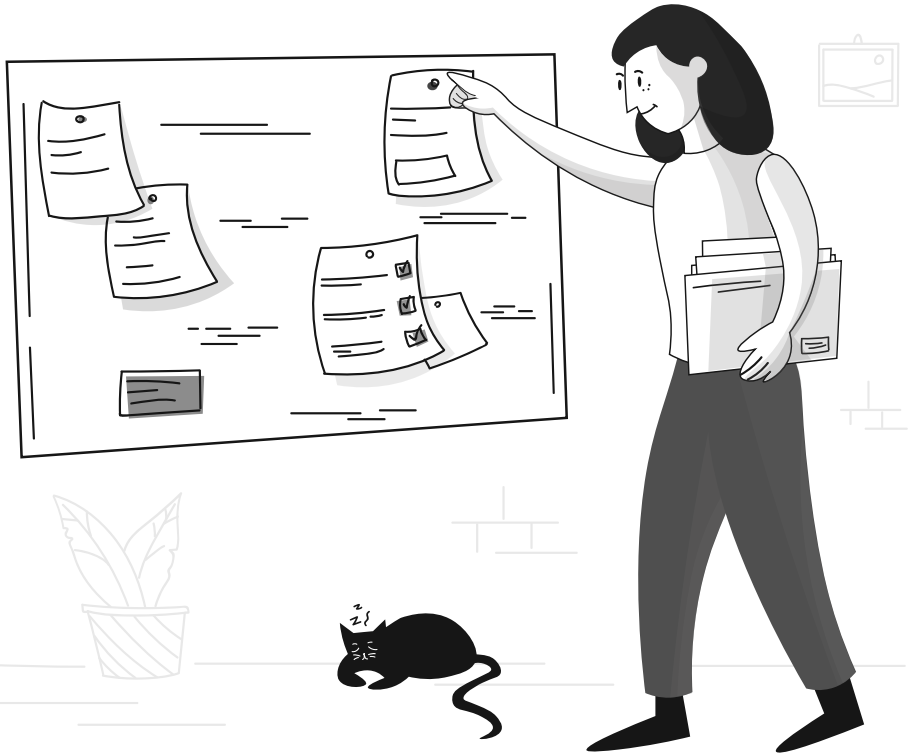
✔ 영상처리 알고리즘의 원리 이해
✔ 영상처리 기능구현 (with python)
✔ tkinter 로 GUI 기능구현

- 개발 환경

✔ Python 3.12
✔ Pycharm Community 2023

- 지원 환경

✔ Windows 11
✔ Linux



기능 요약

- 공통기능

✔ 파일열기
✔ 파일저장
✔ 프로그램 종료

- 통계기능

✔ 평균
✔ 중앙값

- 기하학 처리

✔ 이동 , 회전 , 대칭
✔ 확대 , 축소

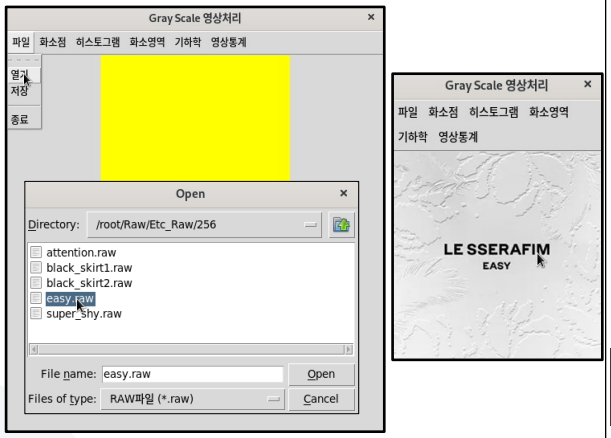
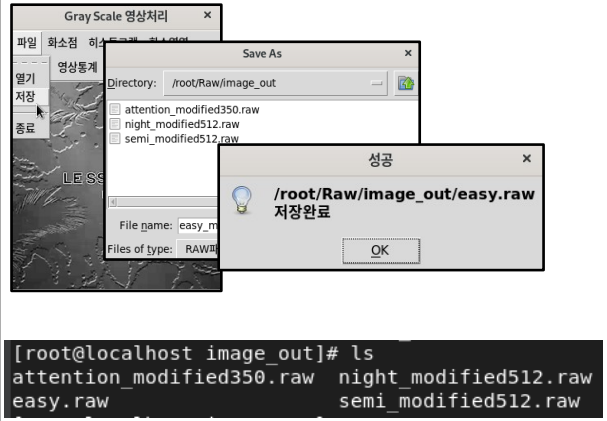
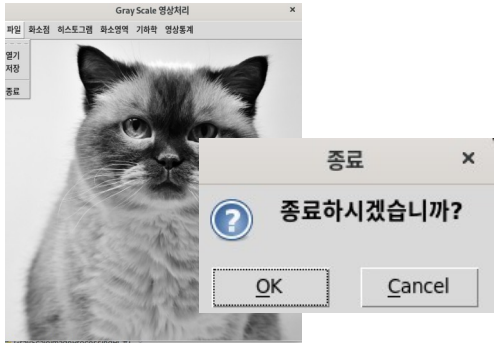
- 화소 점 처리

✔ 산술연산 , 논리연산
✔ 감마보정
✔ 흑백 , 반전
✔ 파라볼라 CAP, CUP
✔ 히스토그램 처리

- 화소 영역 처리

✔ 블러링
✔ 엠보싱
✔ 고주파 샤프닝
✔ 가우시안 스무딩
✔ 에지검출 (1 차미분)
✔ 에지검출 (2 차미분)

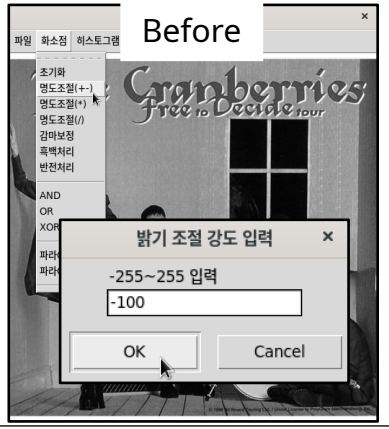
공통 기능

	파일 열기	파일 저장	프로그램 종료
화면			
핵심 코드	`# 3. 영상 바이너리로 읽기 포인터 생성` `rfp = open(image_path, "rb")` `## 영상의 크기 만큼 반복하면서 1픽셀 씩 저장` `for h in range(height):` `  for w in range(width):` `  # ord = 아스키 코드,유니 코드 읽어주기 <-> 반대는 chr` `  inImage[h][w] = ord(rfp.read(1))`	`# 포맷: B와 이미지를 구조체로 만들어서 write` `import struct` `for h in range(height):` `  for w in range(width):` `  wfp.write(struct.pack( _fmt: "B", *v: outImage[h][w]))` `# 파일 포인터 닫기` `wfp.close()` `# 저장완료 메세지 박스 보여주기` `messagebox.showinfo( title: "성공", wfp.name + "저장완료")`	`def exit_app():` `  global window` `  if messagebox.askokcancel( title: "종료", message: "종료하시겠습니까?"):` `  window.destroy()`
세부 사항	이미지의 크기에 따라 어플리케이션 화면의 크기가 조정됨 .		

화소 점 처리 - 산술연산

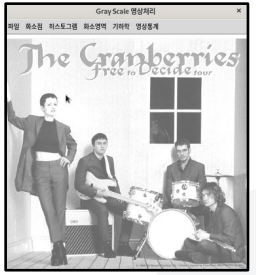
더하기 , 빼기

화면

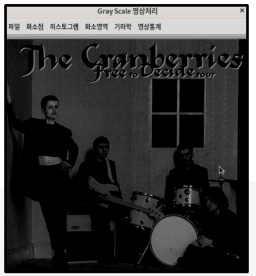


After

+100



-100



핵심 코드

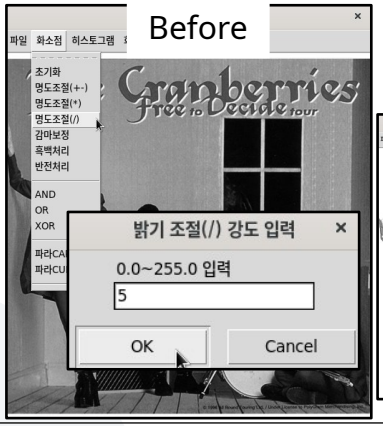
```
# 입력 이미지 +- 밝기 계산
for h in range(out_height):
    for w in range(out_width):
        temp = outImage[h][w] + iVal
        if (temp > 255):
            outImage[h][w] = 255
        elif (temp < 0):
            outImage[h][w] = 0
        else:
            outImage[h][w] = temp
```

세부 사항

이미지의 히스토그램의 분포가 그대로 좌 , 우로 이동

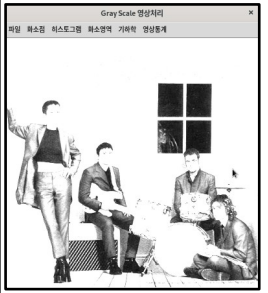
곱하기 , 나누기

화면



After

*5



/5

핵심 코드

```
# 입력 이미지 */ 밝기 계산
for h in range(out_height):
    for w in range(out_width):
        temp = outImage[h][w] * iVal
        if (temp > 255):
            outImage[h][w] = 255
        elif (temp < 0):
            outImage[h][w] = 0
        else:
            outImage[h][w] = temp
```

```
# 입력 이미지 */ 밝기 계산
for h in range(out_height):
    for w in range(out_width):
        temp = outImage[h][w] / iVal
        if (temp > 255):
            outImage[h][w] = 255
        elif (temp < 0):
            outImage[h][w] = 0
        else:
            outImage[h][w] = int(temp)
```

세부 사항

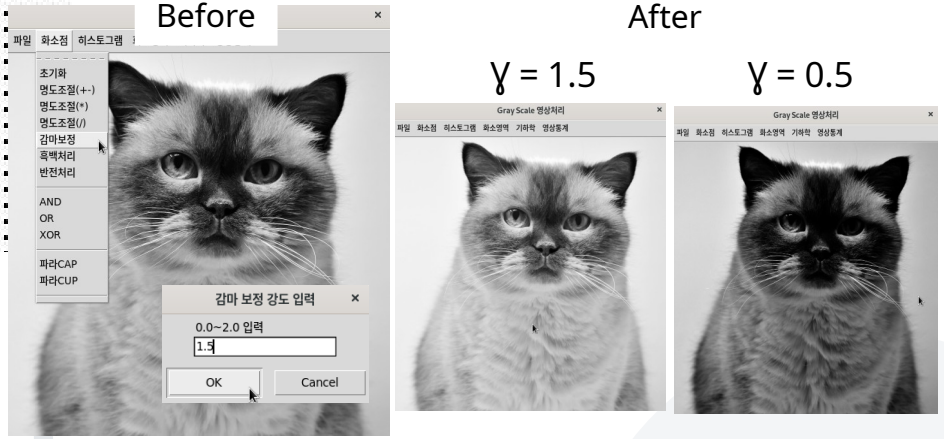
1) 곱하기 - 어두운 화소는 조금 밝아지고 , 밝은 화소는 많이 밝아짐
2) 나누기 - 전반적으로 어두워짐 . 히스토그램의 높이가 전반적으로 낮아짐

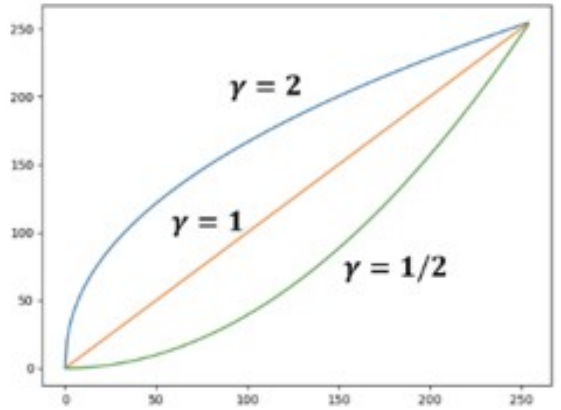
화소 점 처리 - 논리연산

	AND	OR	XOR
화면	Before / After 	Before / After 	Before / After 
핵심 코드	`# AND 처리` `for h in range(out_height):` `  for w in range(out_width):` `  outImage[h][w] = outImage[h][w] & iVal`	`# OR 처리` `for h in range(out_height):` `  for w in range(out_width):` `  outImage[h][w] = outImage[h][w] | iVal`	`# XOR 처리` `for h in range(out_height):` `  for w in range(out_width):` `  outImage[h][w] = outImage[h][w] ^ iVal`
세부 사항	특정 비트를 의도적으로 0 으로 만들 수 있다 . 예) 128 을 입력 -〉 128 이상의 화소만 128 로 표현된다	특정 비트를 의도적으로 1 으로 만들 수 있다 . 예) 127 을 입력 -〉 128 이상의 화소가 255 로 표현된다 또한 127 이하의 화소는 전부 127 이 된다 .	특정 비트를 의도적으로 반전 시킬 수 있다 . 예) 128 을 입력 -〉 128 이상의 화소는 127 을 뺄셈 127 이하의 화소는 127 을 더하는 효과

화소 점 처리 - 감마보정

화면





← 감마 커브

핵심 코드

```
# 감마 보정 강도 입력
fVal = askfloat( title: "감마 보정 강도 입력",  prompt: "0.0~2.0 입력", maxvalue=2.0, minvalue=0.0)

# gamma processing.....
for h in range(out_height):
    for w in range(out_width):
        outImage[h][w] = int(255 * (outImage[h][w] / 255) ** (1.0 / fVal))
```

$$y = 255 * (\frac{x}{255})^{1/\gamma}$$

감마 보정의 수식

세부 사항

아주 어두운 부분과 아주 밝은 부분을 제외하고 , 중앙값 부근의 밝기를 조절하여
1) 대비가 좋아지는 결과 2) 굴절된 모니터에서 왜곡을 감소시키는 역할을 할 수 있다 .

화소 점 처리 - 흑백 , 반전

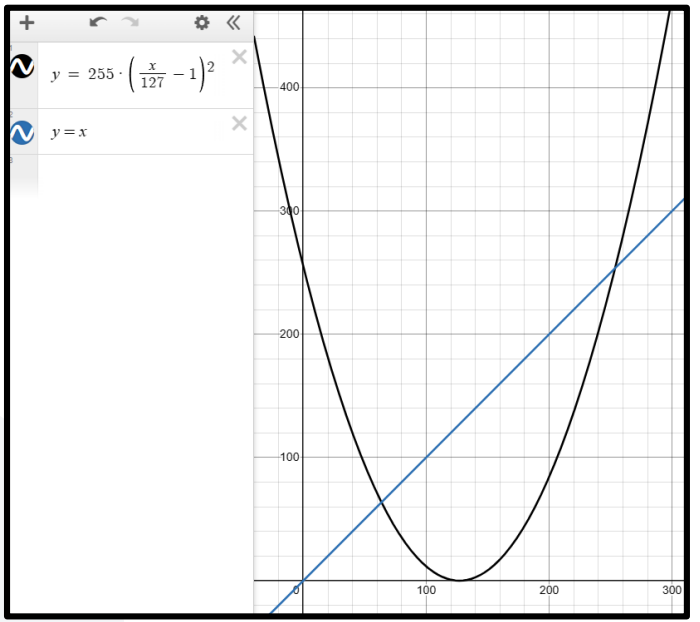
	흑백	반전
화면	 	 
핵심 코드	```# 흑백 처리 기준값 설정 iVal = askinteger(title: "흑백 처리 기준값 입력", prompt: "0~255 입력", maxvalue=255, minvalue=0) # 기준값으로 흑백 나누기 for h in range(out_height): for w in range(out_width): if outImage[h][w] < iVal: outImage[h][w] = 0 else: outImage[h][w] = 255```	```# 반전시키기 for h in range(out_height): for w in range(out_width): outImage[h][w] = 255 - outImage[h][w]```
세부 사항	입력값을 기준으로 크면 최댓값 , 작으면 최솟값을 저장	최댓값과 화소의 차분을 저장 = 반전

화소 점 처리 - 파라볼라 CAP

화면

Before

After



← 파라볼라 CAP 그래프

핵심 코드

```
# para_cap 처리
for h in range(out_height):
    for w in range(out_width):
        temp = int(255 * ((outImage[h][w] / 127 - 1) ** 2))
        outImage[h][w] = 255 if temp > 255 else 0 if temp < 0 else int(temp)
```

$$y = 255 * (\frac{x}{127} - 1)^2$$

세부 사항

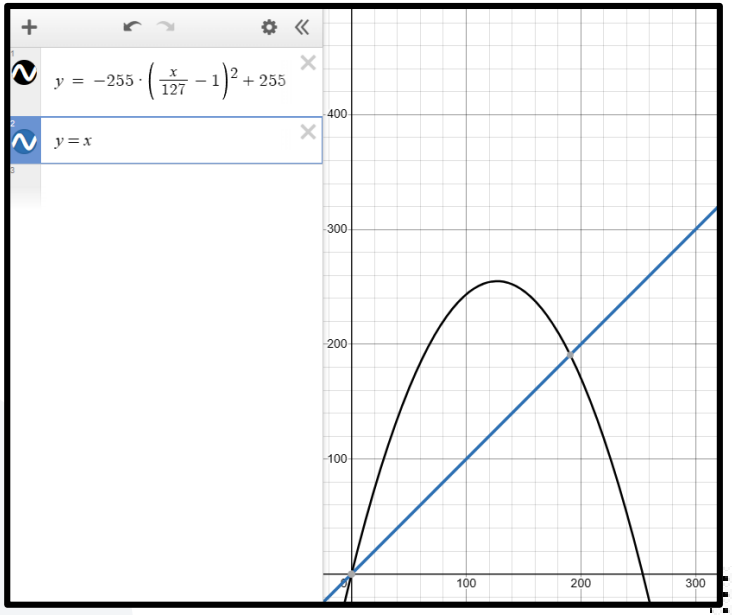
1) 어두운 화소들이 부드럽게 반전 되는 효과
2) 중앙값 부근의 화소들이 실제보다 밝아지는 효과
3) 밝은 화소들의 기울기가 가파르게 바뀌는 효과

화소 점 처리 - 파라볼라 CUP

화면

Before

After



← 파라볼라 CUP 그래프

핵심 코드

```
# para_cup 처리
for h in range(out_height):
    for w in range(out_width):
        temp = int(-255 * (outImage[h][w] / 127 - 1) ** 2 + 255)
        outImage[h][w] = 255 if temp > 255 else 0 if temp < 0 else int(temp)
```

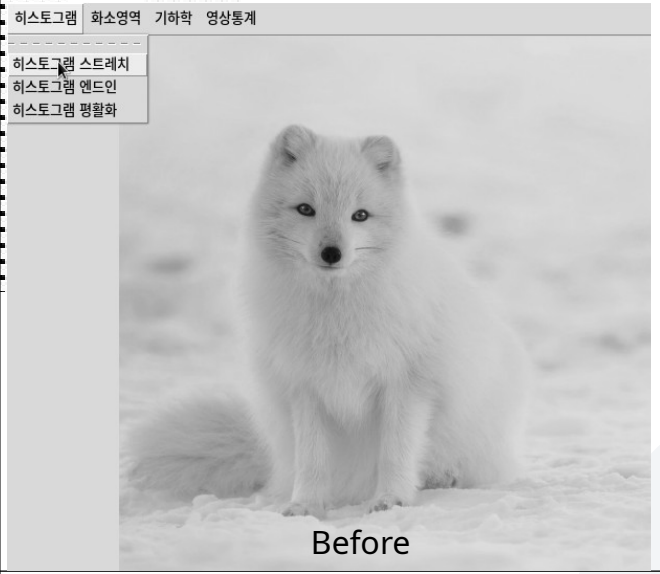
$$y = -255 * (\frac{x}{127} - 1)^2 + 255$$

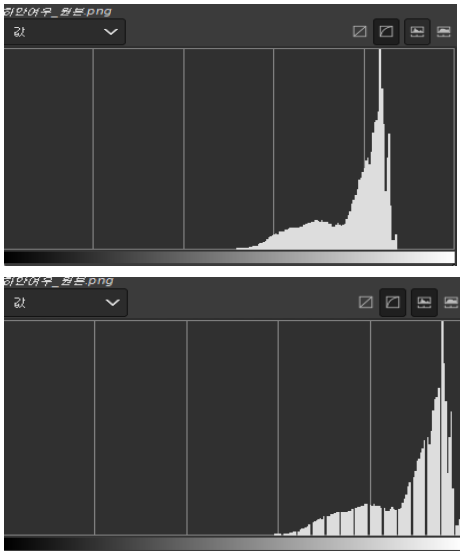
세부 사항

1) 어두운 화소들이 급격하게 밝은 값으로 바뀌는 효과
2) 중앙값 부근의 화소들이 실제보다 밝아지는 효과
3) 밝은 화소들이 급격하게 어두운 값으로 바뀌는 효과

화소 점 처리 - 히스토그램 스트레치

화면





핵심 코드

```
# 최대,최소 명도를 이용해서 정규화
for h in range(out_height):
    for w in range(out_width):
        outImage[h][w] = int((outImage[h][w] - lowest) / (highest - lowest) * 255.0)
```

$$new\ pixel = \frac{old\ \ pixel - low}{high - low} \times 255$$

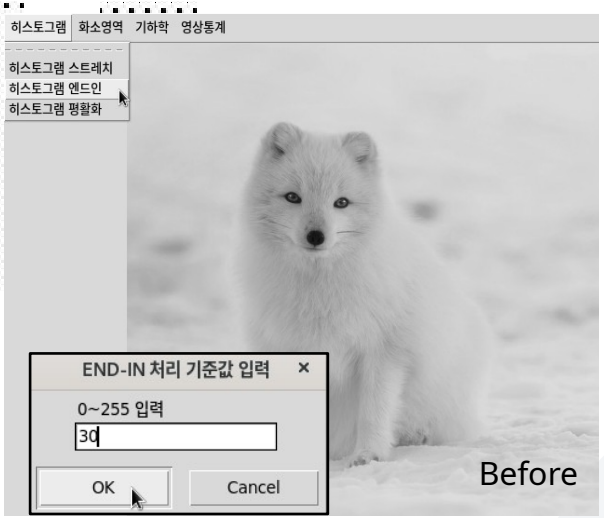
세부 사항

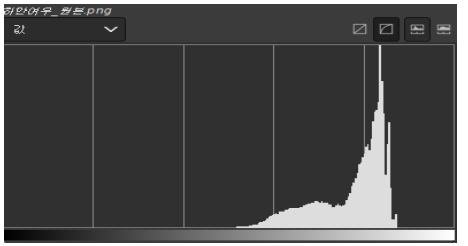
1) 편향된 화소의 분포를 정규화하는 효과 → 명암의 대비가 증가

2) **문제점 : 분포가 완만한 이미지는 더 이상 스트레치 불가**

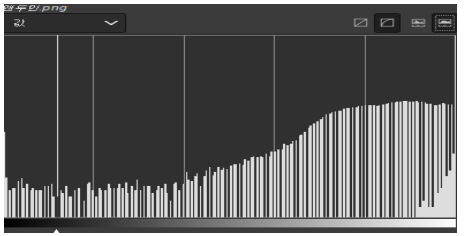
화소 점 처리 - 히스토그램 엔드인 스트레치

화면





BEFORE



AFTER

핵심 코드

```python
# input end-in value
iVal = askinteger( title: "END-IN 처리 기준값 입력",  prompt: "0~255 입력", maxvalue=255, minvalue=0)
# 적절히 endin 조절
highest -= iVal
lowest += iVal
# 최대,최소 명도를 이용해서 정규화
for h in range(out_height):
    for w in range(out_width):
        temp = int((outImage[h][w] - lowest) / (highest - lowest) * 255.0)
        outImage[h][w] = 255 if temp > 255 else 0 if temp < 0 else int(temp)
```

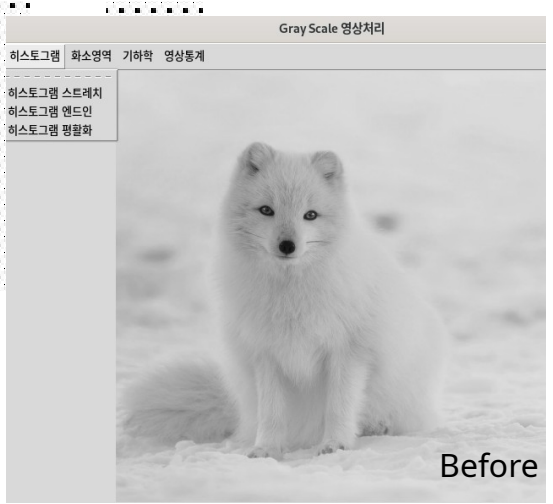
$$new\ pixel = \begin{cases} 0 & old\ pixel \le low \\ \dfrac{old\ pixel - low}{high - low} \times 255 & low \le old\ pixel \le high \\ 255 & high \le old\ pixel \end{cases}$$

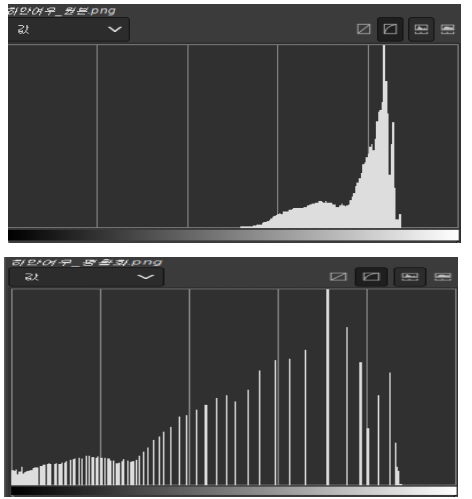
세부 사항

1) 편향된 화소의 분포를 정규화하는 효과 → 명암의 대비가 증가
2) **문제점 : 일부 픽셀 값의 정보를 손실**

화소 점 처리 - 히스토그램 평활화

화면





BEFORE

AFTER

핵심 코드

```
# 히스토그램(도수 분포표) 만들기
for h in range(out_height):
    for w in range(out_width):
        histo_gram[outImage[h][w]] += 1
# 0번째 값만 복사
sum_histo_gram[0] = histo_gram[0]
# 누적 히스토그램 (누적 도수 분포표) 만들기
for i in range(1, len(sum_histo_gram) - 1, 1):
    sum_histo_gram[i] = sum_histo_gram[i - 1] + histo_gram[i]
# 정규화 히스토그램 만들기
for i in range(len(sum_histo_gram)):
    norm_histo_gram[i] = int(sum_histo_gram[i]*(1.0/n_pixels)*highest)
```

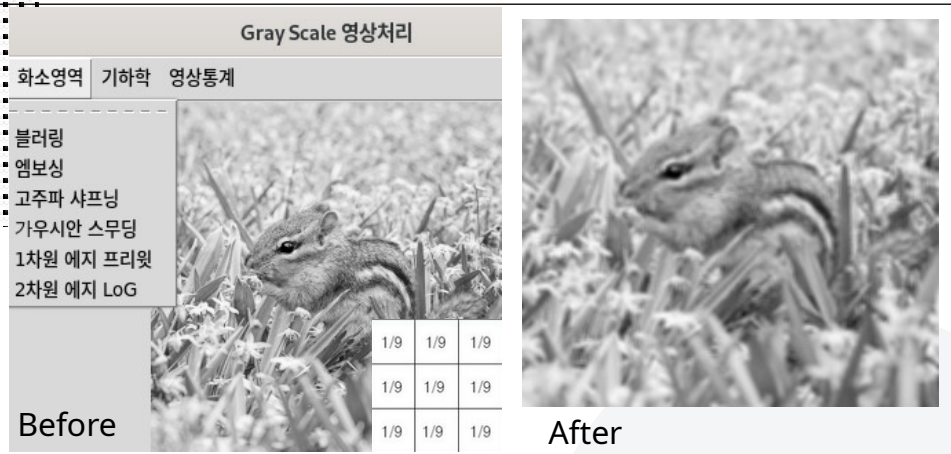
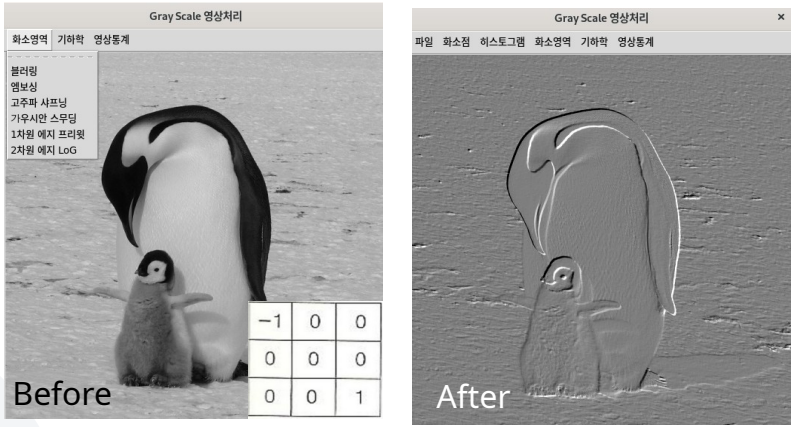
$$sum[i] = \sum_{j=0}^{i} hist[j]$$

$$n[i] = sum[i] \times \frac{1}{N} \times I_{\max}$$

세부 사항

명암의 대비가 효과적으로 개선

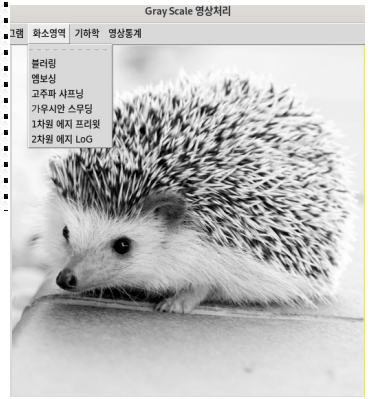
화소 영역 처리 - 블러링 , 엠보싱

	블러링	엠보싱
화면		
핵심 코드	`# 마스크 적용하기` `for h in range(out_height):` `  for w in range(out_width):` `  temp = 0.0` `  for mh in range(mask_height):` `  for mw in range(mask_width):` `  temp += tempImage[h + mh][w + mw] * mask[mh][mw]` `  outImage[h][w] = 255 if int(temp) > 255 else 0 if int(temp) < 0 else int(temp)`	`# 마스크 적용하기` `for h in range(out_height):` `  for w in range(out_width):` `  temp = 0.0` `  for mh in range(mask_height):` `  for mw in range(mask_width):` `  temp += tempImage[h + mh][w + mw] * mask[mh][mw]` `  # 후처리 (마스크의 총 합이 1이기 때문에)` `  temp += 127.0` `  outImage[h][w] = 255 if int(temp) > 255 else 0 if int(temp) < 0 else int(temp)`
세부 사항	마스크의 총 합이 1 이기 때문에 후처리 (+127) 를 하지 않음	마스크의 총 합이 0 이기 때문에 후처리 (+127) 를 적용

화소 영역 처리 - 고주파 샤프닝 , 가우시안 스무딩

고주파 샤프닝

화면



Before

After

핵심 코드

```
# 마스크 적용하기
for h in range(out_height):
    for w in range(out_width):
        temp = 0.0
        for mh in range(mask_height):
            for mw in range(mask_width):
                temp += tempImage[h + mh][w + mw] * mask[mh][mw]
        # 원본 영상에 추출한 고주파 값을 더해주어 샤프닝 효과 적용
        temp += outImage[h][w]
        outImage[h][w] = 255 if int(temp) > 255 else 0 if int(temp) < 0 else int(temp)
```

−1/9	−1/9	−1/9
−1/9	8/9	−1/9
−1/9	−1/9	−1/9

세부 사항

“ 고주파 성분”이란 , 주변영역과 색 차이가 크게 나는 부분이다 .
고주파 샤프닝은 경계선을 뚜렷하게 만들어 이미지를 선명하게 만든다 .

가우시안 스무딩

화면

고주파 샤프닝

가우시안 스무딩

핵심 코드

```
# 마스크 적용하기
for h in range(out_height):
    for w in range(out_width):
        temp = 0.0
        for mh in range(mask_height):
            for mw in range(mask_width):
                temp += tempImage[h + mh][w + mw] * mask[mh][mw]

        outImage[h][w] = 255 if int(temp) > 255 else 0 if int(temp) < 0 else int(temp)
```

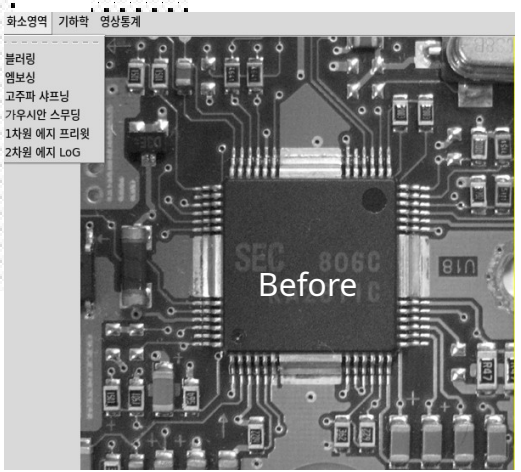
1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

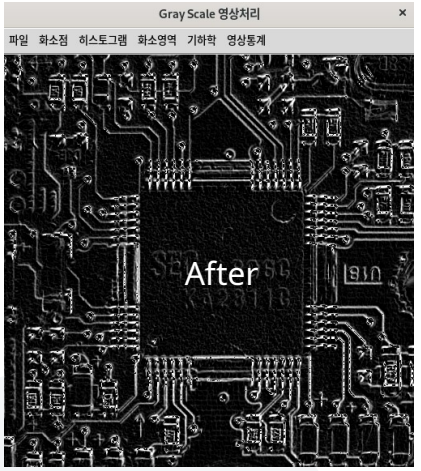
세부 사항

가우시안 스무딩은 정규분포의 값을 일반화하여 마스크를 만든다 .
따라서 , 화소값을 주변과의 차이가 적은 쪽으로 변환한다 .

화소 영역 처리 - 1 차미분 에지검출 (프리윗)

화면





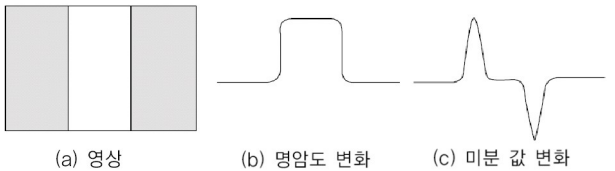
행 검출 마스크 열 검출 마스크

프리윗 $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

1) 위 아래 주변 행의 변화량이 클 수록 해당 화소의 절댓값이 커진다 .
2) 좌 주 주변 열의 변화량이 클 수록 해당 화소의 절댓값이 커진다 .
마스크의 부호를 양수 , 음수로 하여 극명하게 차이가 나게 된다 .

핵심 코드

```
# 마스크 적용하기
for h in range(out_height):
    for w in range(out_width):
        temp = 0.0
        for mh in range(mask_height):
            for mw in range(mask_width):
                # 수직마스크 적용
                temp += tempImage[h + mh][w + mw] * h_mask[mh][mw]
                # 수평마스크 적용
                temp += tempImage[h + mh][w + mw] * v_mask[mh][mw]
```



1 차미분으로 에지를 검출하는 원리

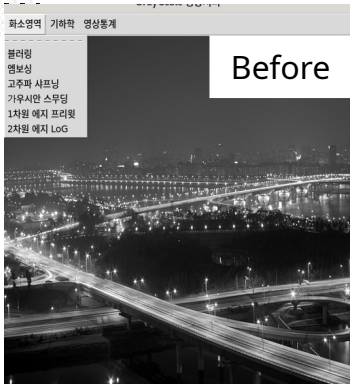
세부 사항

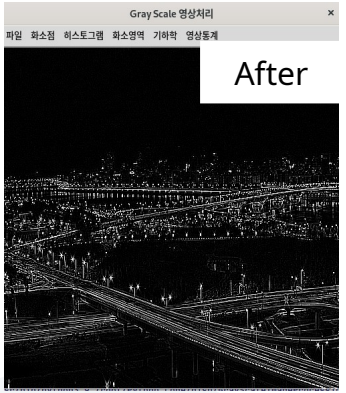
장점) 점진적으로 변화하는 변화량에도 반응한다 .

단점) 너무 많은 에지가 나타날 수 있다 .

화소 영역 처리 - 2 차미분 에지검출 (LoG)

화면





1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

가우시안 필터

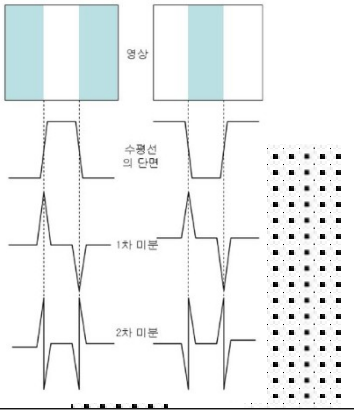
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

라플라시안 마스크

핵심 코드

```
# 가우시안 스무딩 마스크 적용
for h in range(out_height):
    for w in range(out_width):
        temp = 0.0
        for mh in range(mask_height):
            for mw in range(mask_width):
                # 가우시안 스무딩 적용
                temp += tempImage[h + mh][w + mw] * mask_gau[mh][mw]
        outImage[h][w] = 255 if int(temp) > 255 else 0 if int(temp) < 0 else int(temp)
```

```
# 라플라시안 마스크 적용
for h in range(out_height):
    for w in range(out_width):
        temp = 0.0
        for mh in range(mask_height):
            for mw in range(mask_width):
                temp += tempImage[h + mh][w + mw] * mask_lap[mh][mw]
        # 후처리 (중앙값)
        temp += median
        outImage[h][w] = 255 if int(temp) > 255 else 0 if int(temp) < 0 else int(temp)
```

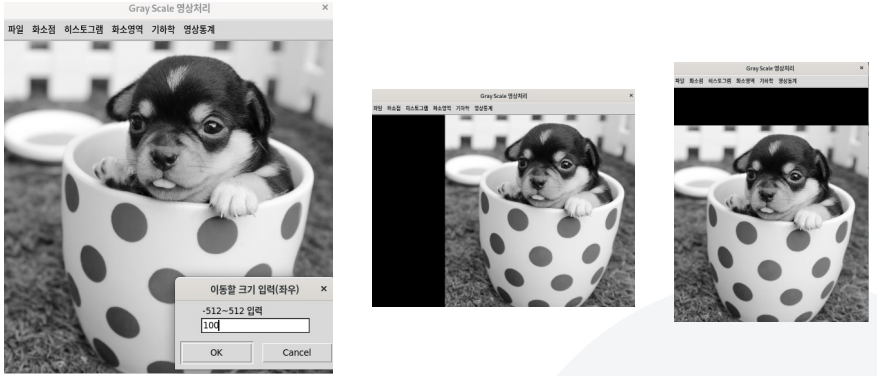


세부 사항

장점) 미분을 두 번 수행하기 때문에 중앙을 중심으로 가늘게 폐곡선의 에지를 검출 .

단점) 점차적으로 밝기 값이 변하는 영역에는 반응하지 않는다 .

기하학 처리 - 이동 , 대칭

	이동	대칭
화면	 Before After	 Before After
핵심 코드	``` # 이동 안 살림 # 입력한 정수가 양수일 때 if (iVal > 0): for h in range(in_height): for w in range(in_width): pixel_moved = w + iVal - 1 tempImage[h][pixel_moved] = inImage[h][w] # 입력한 정수가 0 이하일 때 elif(iVal <=0): for h in range(in_height): for w in range(in_width): tempImage[h][w] = inImage[h][w] ```	``` # 상하 미러링 for h in range(out_height): for w in range(out_width): outImage[out_height - 1 - h][w] = inImage[h][w] outImage[h][w] = inImage[out_height - 1 - h][w] ```
세부 사항	이동한 이미지가 잘리지 않도록 입력 값을 이용하여 적절히 화면을 늘림	

기하학 처리 - 회전

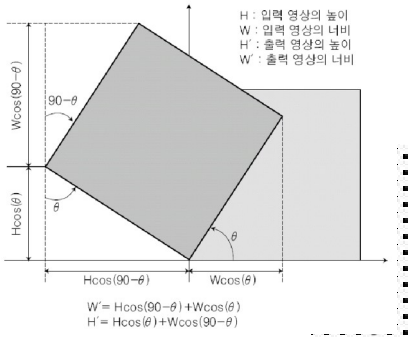
화면

$$\begin{bmatrix} x_{dest} \\ y_{dest} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{source} - C_x \\ y_{source} - C_y \end{bmatrix} + \begin{bmatrix} C_x \\ C_y \end{bmatrix}$$

⊖ 만큼 회전한 이후 좌표의 위치를 구하는 공식

핵심 코드

```python
# 영상 회전(백워딩) 및 중심 좌표 보정
for h in range(out_height):
    for w in range(out_width):
        xd = h
        yd = w
        # x = xcosθ - ysinθ  , y = xsinθ + ycosθ
        # 출력 이미지 중앙점 보정
        xs = math.cos(radian) * (xd - out_center_height) - math.sin(radian) * (yd - out_center_width)
        ys = math.sin(radian) * (xd - out_center_height) + math.cos(radian) * (yd - out_center_width)
        # 원본 이미지 중앙점 보정
        xs += in_center_height
        ys += in_center_width
        if ((0 <= xs and xs < in_height) and (0 <= ys and ys < in_width)):
            outImage[int(xd)][int(yd)] = inImage[int(xs)][int(ys)]
```

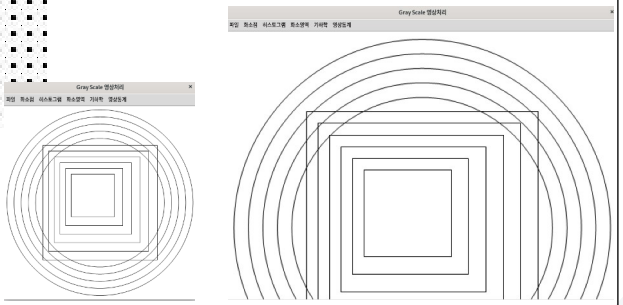
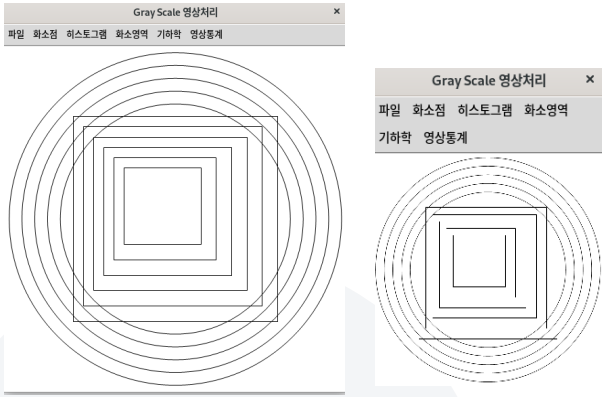
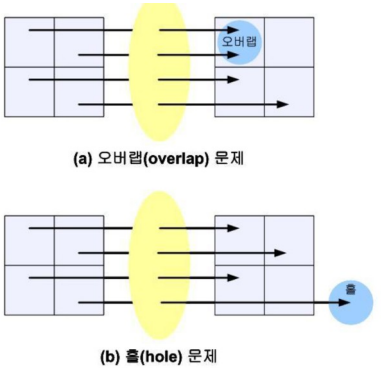
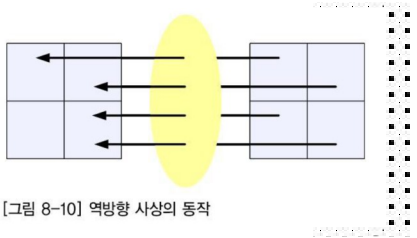


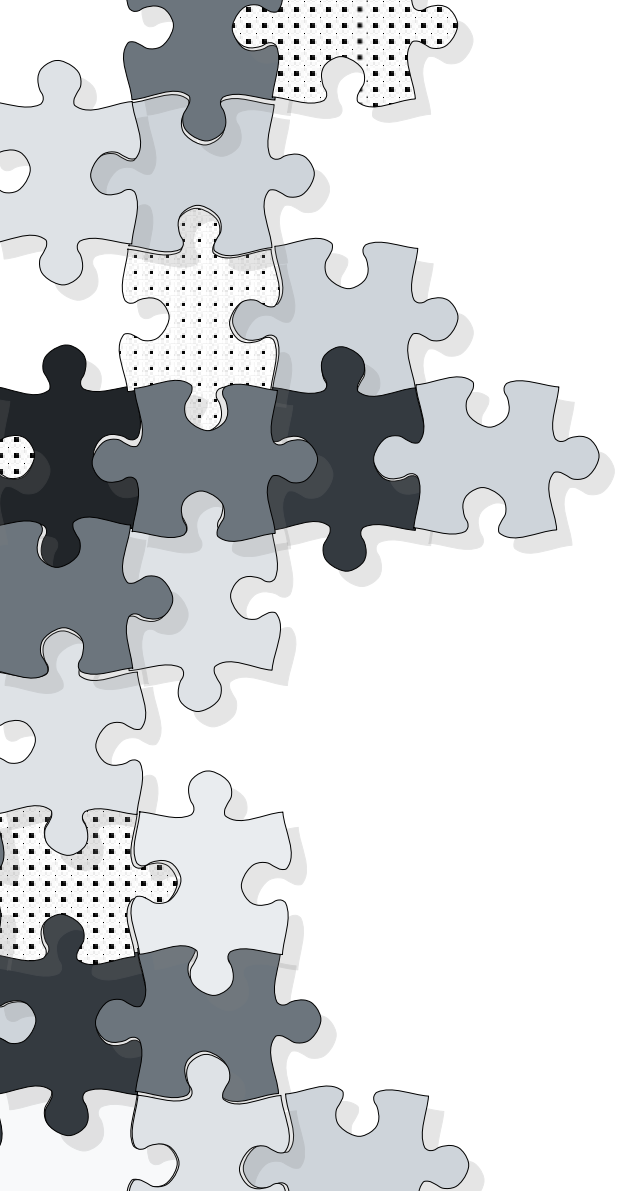
배경의 크기를 구하는 공식

세부 사항

1. 백워딩 기법을 사용하여 영상의 정보 손실을 줄임
2. 변환된 이미지가 잘리지 않도록 배경의 크기도 동시에 조정

기하학 처리 - 확대 , 축소

	확대	축소	포워딩 , 백워딩
화면	 Before After	 Before After	 (a) 오버랩(overlap) 문제 (b) 홀(hole) 문제 포워딩 기법으로 축소 , 확대를 하면 정보가 많이 손실된다
핵심 코드	`# 몇 배 늘릴까` `scale = 2` `out_height = in_height * scale` `out_width = in_width * scale` `outImage = [[0 for _ in range(out_width)] for _ in range(out_height)]` `# 확대 백워딩` `for h in range(out_height):` `  for w in range(out_width):` `  inH = int(h / scale)` `  inW = int(w / scale)` `  if((0<= inH <in_height) and (0<= inW <in_width)):` `  outImage[h][w] = inImage[inH][inW]`	`# 몇 배 줄일까` `scale = 2` `out_height = int(in_height / scale)+1` `out_width = int(in_width / scale)+1` `outImage = [[0 for _ in range(out_width)] for _ in range(out_height)]` `# 축소 포워딩` `for h in range(in_height):` `  for w in range(in_width):` `  outH = int(h / scale)` `  outW = int(w / scale)` `  if ((0 <= outH < out_height) and (0 <= outW < out_width)):` `  outImage[outH][outW] = inImage[h][w]`	 [그림 8-10] 역방향 사상의 동작 백워딩 기법으로 축소 , 확대를 하면 정보 손실을 줄일 수 있다
세부 사항	백워딩을 적용하여 영상 정보의 손실을 줄임		



마무리

느낀 점

알고리즘을 수식를 이해하고 그래프로 그려보면서 알고리즘의 역할을 제대로 알 수 있었고 다양하게 응용할 수 있다고 느낌

보완하고 싶은 점

1. 화소영역처리 시 , 다양한 마스크를 사용할 수 있게 만들고 싶다
2. 더 성능이 좋은 화소 보간법을 구현하고 싶다 .
3. AI 영상처리에 중요하게 사용되는 에지검출에 대해 더 공부하고 싶다 .