파이썬 OpenCV를 이용한 영상처리

저자 허진경

파이썬 OpenCV를 이용한 영상처리

발　행 | 2021년 9월 24일
저　자 | 허진경
펴낸이 | 한건희
펴낸곳 | 주식회사 부크크
출판사등록 | 2014.07.15.(제2014-16호)
주　소 | 서울특별시 금천구 가산디지털1로 119 SK트윈타워 A동 305호
전　화 | 1670-8316
이메일 | info@bookk.co.kr

ISBN | 979-11-372-5701-6

www.bookk.co.kr
© 허진경 2021
본 책은 저작자의 지적 재산으로서 무단 전재와 복제를 금합니다.

CONTENT

1장. 영상처리 개요 ····· 1

1절. 이미지 표현 ····· 2
1.1. 영상의 종류와 표현 방법 ····· 2
1.2. 디지털 이미지와 좌표 표현 방법 ····· 3
1) 표본화와 양자화 ····· 4
1.3. 레나 이미지 ····· 5

2절. OpenCV와 이미지 읽기 ····· 6
2.1. OpenCV ····· 6
1) OpenCV란? ····· 6
2) OpenCV 설치 ····· 8
3) OpenCV 사용하기 ····· 8
2.2. cv2.imread()로 이미지 불러오기 ····· 9
2.3. cv2.imshow()로 이미지 출력하기 ····· 10
2.4. cv2.imwrite()로 이미지 저장하기 ····· 11
2.5. Matplotlib 사용해서 이미지 출력하기 ····· 12

3절. 디지털 화상과 컬러 ····· 15
3.1. 이진 화상 ····· 15
3.2. 회색조 화상 ····· 16
3.3. 컬러 화상 ····· 17
1) RGB 모델 ····· 17
2) HSV 모델 ····· 18
3) YUV 모델 ····· 19
3.4. 색상 공간 변환 ····· 20
1) cv2.cvtColor() ····· 20
2) 컬러 이미지를 그레이스케일 이미지로 변환하기 ····· 20

4절. 동영상 처리하기 ····· 22
4.1. 동영상 읽기 ····· 22
4.2. 카메라 영상 읽기 ····· 24
1) USP 카메라 영상 읽기 ····· 24
2) IP 카메라 RTSP 영상 읽기 ····· 26
4.3. 동영상 속성 ····· 27
4.4. 동영상 프레임 저장하기 ····· 31
4.5. 비디오 저장하기 ····· 33

5절. 이벤트 처리하기 ····· 35
5.1. 마우스 이벤트 ····· 35
5.2. 키보드 이벤트 ····· 39
5.3. 트랙 바 사용하기 ····· 40

6절. 도형 그리기 ····· 42
6.1. 선 그리기 ····· 42
6.2. 사각형 그리기 ····· 43
6.3. 원 그리기 ····· 44
6.4. 타원과 호 그리기 ····· 46
6.5. 다각형 그리기 ····· 48
6.6. 문자열 표시하기 ····· 49

2장. 영상 기본 연산 ····· 51

1절. 영상 기본 정보 ····· 52
1.1. 영상 기본 속성 ····· 52
1.2. 영상 화소 처리하기 ····· 53
1.3. 함수 만들어 사용하기 ····· 55
1.4. ROI ····· 58

2절. 기본 연산 ····· 60
2.1. 이미지 채널 ····· 60
2.2. 이미지 합치기 ····· 61
1) cv2.add()와 numpy.add() ····· 61
2) 이미지 블랜딩 ····· 63
3) 비트연산 ····· 65
4) cv2.seamlessClone() ····· 68
2.3. 이진화 ····· 72
1) 이진화 ····· 72
2) cv2.threshold() ····· 73
3) 적응 임계처리 ····· 74
4) 디더링 ····· 77

3절. 히스토그램 ····· 81
3.1. 이미지 히스토그램 ····· 81
3.2. 히스토그램 함수 구현 ····· 82
3.3. cv2.calcHist() ····· 84
3.4. 채널 ····· 85

3장. 이미지 변환 ····· 87

1절. 기하 변환 ····· 88
1.1. 변환 ····· 88
1) 변환의 종류 ····· 88
2) 기준 좌표 설정 방법 ····· 89

2절. 크기변환 ····· 90
2.1. 크기변환 ····· 90
2.2. 출력 영상을 기준으로 처리 ····· 93
2.3. 보간법 ····· 95

1) 최근방법 ····· 95
2) 선형방법 ····· 97
2.4. Max Pooling ····· 99
2.5. cv2.resize() ····· 101

3절. 이동 ····· 107

4절. 회전 ····· 109

5절. 아핀 변환 ····· 114
5.1. 아핀 변환 ····· 114
5.2. 변환 행렬 ····· 114
5.3. cv2.warpAffine() ····· 116

6절. 원근법 변환 ····· 119
6.1. 원근법 변환 ····· 119
6.2. 변환 행렬 ····· 119
6.3. cv2.warpPerspective() ····· 120

7절. 푸리에 변환 ····· 122
7.1. 푸리에 변환 ····· 122
7.2. 넘파이를 이용한 푸리에 변환 ····· 123
7.3. OpenCV를 이용한 푸리에 변환 ····· 126

8절. 문서 스캔하기 ····· 129

4장. 엣지 검출 ····· 133

1절. 이미지 필터와 합성곱 ····· 134
1.1. 이미지 필터 ····· 134
1.2. 합성곱 ····· 135

1.3. 합성곱 함수 구현하기 ····· 137
1.4. cv2.filter2D() ····· 139

2절. 엣지 검출 필터 ····· 141

2.1. 차분 필터 ····· 141
2.2. 로버츠 필터 ····· 144
2.3. 프리위트 필터 ····· 147
2.4. 소벨 필터 ····· 150
2.5. cv2.Sobel() ····· 152
2.6. 라플라시안 필터 ····· 154
2.7. cv2.Laplacian() ····· 156
2.8. 엣지 추출 필터 비교 ····· 157

3절. 캐니 엣지 ····· 159

3.1. 캐니 엣지 ····· 159
3.2. cv2.Canny() ····· 161

5장. 화질 개선 ····· 163

1절. 잡음 제거 ····· 164

1.1. 평균값 필터 ····· 164
1.2. cv2.blur() ····· 166
1.3. 메디안 필터 ····· 167
1.4. cv2.medianBlur() ····· 168
1.5. scipy.ndimage.median_filter() ····· 169
1.6. 가우시안 필터 ····· 171
1.7. cv2.GaussianBlur() ····· 172
1.8. cv2.bilateralFilter() ····· 173
2절. 모폴로지 연산 ····· 175
2.1. 수축 ····· 175

2.2. 팽창 ····· 176
2.3. 열기(Opening) ····· 178
2.4. 닫기(Closing) ····· 179
2.5. cv2.erode() ····· 180
2.6. cv2.dilate() ····· 181
2.7. cv2.morphologyEx() ····· 182

3절. 화질 개선 ····· 185

3.1. 히스토그램과 밝기 조절 ····· 185
1) 히스토그램 ····· 185
2) 히스토그램 밝기 조절 ····· 187
3.2. 콘트라스트 강조 ····· 189
3.3. cv2.normalize() ····· 192
3.4. 히스토그램 이퀄라이제이션 ····· 193
1) 히스토그램 이퀄라이제이션 구현 ····· 193
2) cv2.equalizeHist() ····· 196
3) 누적 히스토그램 그래프 ····· 197
3.5. 적응적 히스토그램 이퀄라이제이션 ····· 198
1) CLAHE 198
2) cv2.createCLAHE() 199
3.6. 완벽하게 평평한 히스토그램 이퀄라이제이션 ····· 203
1) 히스토그램 평탄화 알고리즘 ····· 203
2) 히스토그램 평탄화 구현 ····· 206
3) 실행 전/후 히스토그램 ····· 210
4) 누적 히스토그램 ····· 211

6장. 객체 특징 추출 ····· 213

1절. 라벨링과 특징 파라미터 ····· 214

1.1. 라벨링 ····· 214
1.2. cv2.connectedComponents() ····· 219

1.3. 특징 파라미터 ····· 220
1) 면적과 중심 ····· 220
2) 주위 길이 ····· 222
3) 원형도 ····· 226
1.4. OpenCV를 이용한 특징 추출 ····· 228
1) cv2.findContours()와 cv2.drawContours() ····· 228
2) cv2.moments() ····· 230
3) cv2.contourArea() ····· 231
4) cv2.arcLength() ····· 231
5) 컨투어 속성 ····· 235
6) cv2.approxPolyDP() ····· 236

2절. 객체 탐지 ····· 239
2.1. 경계상자 ····· 239
1) 사각형으로 둘러쌓기 ····· 239
2) 원으로 둘러쌓기 ····· 240
2.2. cv2.convexHull() ····· 242

3절. OpenCV와 머신러닝을 이용한 필기체 숫자 인식 ····· 244
3.1. 필기체 숫자 데이터 불러오기 ····· 244
3.2. 머신러닝 분류모형 만들기 ····· 245
3.3. 머신러닝 모형 저장하고 불러오기 ····· 246
3.4. 카메라를 이용한 실시간 필기체 숫자 인식하기 ····· 248

이 책은 아래의 수정 이력이 있습니다.

- 1차 수정일 : 2021년 11월 25일
- 2차 수정일 : 2021년 12월 24일

1장. 영상처리 개요

1절. 이미지 표현

1.1. 영상의 종류와 표현 방법

우리가 알고 있는 영상은 크게 정지영상(image)과 동영상(movie)이 있습니다. 정지영상(이 책은 이미지라고 부름)은 문자 또는 도형 등을 표현한 2진영상, 흑백사진에 해당하는 그레이스케일(Grayscale) 영상, 자연색을 표현한 컬러 영상, 그리고 다중분광(Multispectrum, 멀티스펙트럼) 영상 등이 있습니다. 동영상에는 흑백 동영상과 컬러 동영상이 있습니다.

그림 1. 영상의 종류

2진 영상의 경우 검은색과 흰색으로만 표현된 영상을 말합니다. 이 영상을 우리가 평소에 마주할 일을 흔하지 않습니다. 그러나 영상처리를 하다 보면 2진 영상을 사용할 일이 매우 많다는 것을 느끼게 됩니다. 모노크롬 영상은 흑백 영상이라고 부르기도 하는데 흑백영상이라

고 해서 검은색과 흰색만 사용하는 것이 아니고 검은색과 흰색 사이에 여러 단계를 두어 회색도 표현할 수 있는 영상들입니다. 이 책에서 모노크롬 영상은 그레이스케일(Grayscale) 또는 회색조 영상이라고 부릅니다. 멀티스펙트럼 영상은 우리가 눈으로 볼 수 있는 가시광선 영역의 정보만 표현하는 것이 아니라 자외선 등 우리 눈으로 볼 수 없는 영역의 스펙트럼도 정보를 저장한 영상을 의미합니다. 요즘은 열화상 카메라를 이용하면 온도 정보를 저장한 이미지를 얻을 수 있으며, 수분 스트레스 지수 카메라를 이용하면 피부의 수분 상태를 확인할 수 있는 이미지를 얻을 수 있습니다. 이들처럼 우리가 눈으로 볼 수 없는 영역들을 시각화할 수 있는 기술들이 있습니다.

1.2. 디지털 이미지와 좌표 표현 방법

원본 데이터가 어떤 것이든지 영상을 디지털 이미지로 변환해서 컴퓨터를 이용해 처리하는 것을 영상처리라 합니다. 영상처리를 위해서는 사진, 도형 등의 아날로그 이미지를 컴퓨터에 저장한 디지털 이미지로 변환해야 합니다. 컴퓨터에 저장된 디지털 이미지는 2차원 배열 구조입니다.

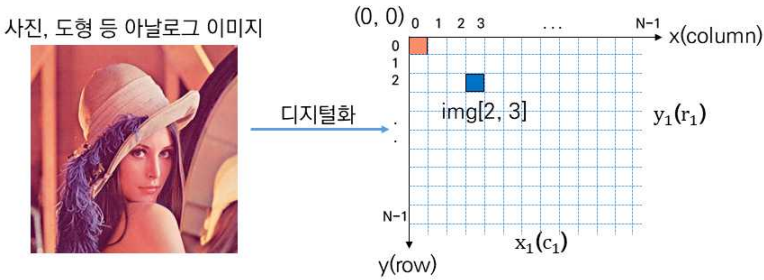


그림 2. 디지털화

그림의 폭은 영상에서 가로 방향 즉, x 방향의 길이를 의미하며, 높이는 세로 방향 즉, y 방향의 길이를 의미합니다. 디지털 이미지에서 기준점은 영상의 가장 왼쪽 위 지점이며, 이곳의 좌표는 (0, 0)에 해당합니다. (0, 0)에서 영상의 오른쪽으로 이동할수록 x 좌표가 커지며, 아래쪽으로 이동할수록 y 좌표가 커집니다. 이미지 객체 변수가 img이면 (x, y) 위치의 화소는 img[y, x] 형식으로 인덱싱됩니다. 인덱싱 첫 번째 요소가 행의 위치를 나타내는 y임을 주의하세요.

이미지는 화소(Pixel)들로 구성되어 있습니다. 화소 하나는 영상에서 점(Point) 하나에 해당합니다.

그림 3. 화소

1) 표본화와 양자화

표본화는 공간적으로 연속되는 화상을 이산적인 화소의 집합으로 나누는 것입니다. 우리가 1024*768 이미지, 800*600 이미지 등을 말할 때 숫자가 의미하는 것이 원본 이미지를 디지털화할 때 가로와 세로의 점의 수입니다. 같은 이미지라면 800개 점을 이용해서 표현하는 것 보다 1024개의 점을 이용해서 표현하는 것이 더 선명한(좋은) 이미지라고 할 수 있습니다. 이렇듯이 표본화를 통해 몇 개의 점을 이용해서 표현하는지에 따라 이미지의 화질이 달라집니다.

양자화는 이미지의 농담을 이산적인 정수값으로 변환 조작하는 것을 의미합니다. 1비트(0 또는 1)로 양자화한 것은 2진 화상이라고 합니다. 화소를 저장하는 비트 수를 증가시킬수록 표현이 풍부해지는데 6비트 이상일 경우 사람의 눈으로 거의 구별할 수 없습니다. 6비트를 사용하면 하나의 색을 64(2^6)단계로 나누는데 1단계의 차이를 우리의 눈으로는 구분해 낼 수 없다는 의미입니다. 6비트이면 자연의 색을 표현하기에 충분하지만 컴퓨터에서 다룰 수 있도록 고려하여 화소당 1바이트(8비트)를 사용합니다. 그렇게 하면 화소의 농담을 256단계 0~255의 수치로 표현할 수 있습니다.

1.3. 레나 이미지

레나(Lenna 또는 Lena) 이미지는 플레이보이 잡지 1972년 11월자 센터폴드(잡지 중간에 접어서 들어가 있는 페이지)에 실린 스웨덴의 모델인 레나 포르센(1951년 3월 31일 -)의 사진의 일부분을 말합니다.

그림 4. 레나

얼굴과 어깨만 취한 이 그림 파일은 모든 종류의 영상처리 알고리즘(영상 압축, 영상 노이즈 제거 등)과 관련 과학적 출판물에서 시험용 이미지로 가장 널리 사용됩니다. 실제로 대부분의 영상처리, 신호처리 교과서에는 레나의 사진을 이용한 결과 비교가 나옵니다. 이 책에서도 레나 이미지를 이용해서 OpenCV의 기능들을 설명합니다.

2절. OpenCV와 이미지 읽기

2.1. OpenCV

1) OpenCV란?

OpenCV(Open Source Computer Vision Library)[1]는 오픈소스 컴퓨터 비전 및 기계 학습 소프트웨어 라이브러리입니다. OpenCV는 컴퓨터 비전 애플리케이션을 위한 공통 인프라를 제공하고 상용 제품에서 기계 인식의 사용을 쉽고 빠르게 처리하기 위해서 구축되었습니다. OpenCV는 BSD 라이선스 제품이므로 누구든지 무료로 사용할 수 있습니다.

그림 5. OpenCV 로고

라이브러리에는 2,500개 이상의 최적화 된 알고리즘이 있으며 여기에는 전통적인 영상처리 알고리즘 및 최첨단 컴퓨터 비전과 머신러닝/딥러닝 알고리즘들을 포함하고 있습니다.

이러한 알고리즘은 얼굴을 감지 및 인식하고, 물체를 식별하고, 비디오에서 인간의 행동을 분류하고, 카메라 움직임을 추적하고, 움직이는 물체를 추적하고, 물체의 3D 모델을 추출하고, 스테레오 카메라에서 3D 포인트 클라우드를 생성하고, 이미지를 결합하여 고해상도를 생성하는 데 사용할 수 있습니다.

전체 장면의 이미지, 이미지 데이터베이스에서 유사한 이미지 찾기, 플래시를 사용하여 촬영한 이미지에서 적목 현상 제거, 눈의 움직임

1) https://opencv.org/

추적, 풍경 인식 및 증강 현실 오버레이를 위한 마커 설정 등을 사용할 수 있습니다.

라이브러리를 사용하는 Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota와 같은 잘 알려진 회사와 함께 OpenCV를 광범위하게 사용하는 Applied Minds, VideoSurf 및 Zeitera와 같은 많은 신생 기업이 있습니다.

OpenCV의 배포된 사용 범위는 스트리트 뷰 이미지를 함께 연결하고, 이스라엘의 감시 비디오에서 침입을 감지하고, 중국의 광산 장비를 모니터링하고, 로봇이 Willow Garage에서 물체를 탐색하고 집을 수 있도록 돕고, 유럽에서 수영장 익사 사고를 감지하고, 인터랙티브 아트를 실행하는 것입니다. 스페인과 뉴욕은 터키의 활주로에서 잔해물이 있는지 확인하고, 일본의 빠른 얼굴 감지에 이르기까지 전 세계 공장의 제품 라벨을 검사합니다.

OpenCV는 C++, Python, Java 및 MATLAB 인터페이스가 있으며 Windows, Linux, Android 및 Mac OS를 지원합니다. OpenCV는 주로 실시간 영상처리 애플리케이션에 사용되며, MMX(MultiMedia eXtension) 및 SSE(Streaming SIMD(Single Instruction Multi Data) Extensions) 명령을 활용하여 더 빠른 처리도 가능합니다. 현재 많은 기능을 갖춘 CUDA 및 OpenCL 인터페이스가 현재 활발히 개발되고 있습니다. 500개 이상의 알고리즘이 있으며 이러한 알고리즘을 구성하거나 지원하는 함수는 약 10배 더 많습니다. OpenCV는 기본적으로 C++로 작성되었으며 파이썬용 OpenCV는 파이썬에서 이미지 처리를 위한 인터페이스를 가지고 있습니다.

파이썬용 OpenCV 튜토리얼은 아래 주소를 참고하세요.

- https://docs.opencv.org/master/d6/d00/tutorial_py_root.html

2) OpenCV 설치

파이썬에서 OpenCV를 사용하기 위해서는 패키지를 먼저 설치해야 합니다.

다음 명령은 파이썬용 OpenCV를 설치합니다.

```
pip install opencv-python
```

만일 OpenCV 주요 모듈과 함께 추가 모듈도 설치하려면 다음 명령을 이용하세요.

```
pip install opencv-contrib-python
```

OpenCV에서 불러온 이미지는 넘파이(NumPy)의 다차원 배열(ndarray) 객체입니다. 그러므로 OpenCV를 사용하려면 넘파이 패키지가 설치되어 있어야 합니다. 넘파이 패키지가 설치되어 있지 않다면 아래 명령으로 넘파이 패키지를 설치해야 합니다.

```
pip install numpy
```

3) OpenCV 사용하기

파이썬에서 OpenCV를 사용하려면 먼저 임포트해야 합니다. 임포트 명령이 실행되고 버전이 출력되면 OpenCV가 설치된 것입니다.

```
1  import cv2
2  print(cv2.__version__)
```

```
4.2.1
```

2.2. cv2.imread()로 이미지 불러오기

cv2.imread()는 이미지 파일을 불러와 넘파이 배열로 반환합니다. 인수로 전달되는 값은 파일 이름과 플래그입니다.

cv2.imread(fileName, flags) -> img

- fileName - 이미지 파일의 경로를 지정합니다.
- flags - 이미지 파일을 읽을 때의 옵션입니다.
  파이썬의 OpenCV에서 지원하는 플래그는 IMREAD_COLOR, IMREAD_GRAYSCALE, IMREAD_UNCHANGED가 있습니다.
  - cv2.IMREAD_COLOR : 이미지 파일을 컬러로 읽습니다. 투명한 부분은 무시되며, 기본값입니다. 숫자 1을 사용할 수 있습니다.
  - cv2.IMREAD_GRAYSCALE : 이미지를 그레이스케일로 읽어 들입니다. 숫자 0을 사용할 수 있습니다.
  - cv2.IMREAD_UNCHANGED : 이미지를 알파 채널까지 포함하여 읽어 들입니다. 알파 채널은 불투명도를 표시하며 값이 0이면 완전하게 투명합니다. 숫자 -1을 사용할 수 있습니다.

- 반환값 img : 이미지 정보를 저장한 넘파일 배열입니다. 그레이스케일 이미지는 2차원 배열이며, 컬러 이미지는 3차원 배열입니다.

다음 코드는 lena.jpg 파일을 불러오는 코드입니다. 컬러 이미지의 경우 B(Blue), G(Green), R(Red) 3개의 채널 정보를 포함한 3차원 배열이며, 그레이스케일 이미지의 경우 2차원 배열입니다.

실습에 사용할 이미지 파일은 아래의 주소에서 내려받을 수 있습니다.
- http://coderby.com/img 또는 https://github.com/hjk7902/opencv

```
1  lena_img = cv2.imread("lena.jpg")
2  print(lena_img.shape)
```

```
(512, 512, 3)
```

```
1  lena_gray = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
2  print(lena_gray.shape)
```

```
(512, 512)
```

2.3. cv2.imshow()로 이미지 출력하기

파일에서 읽은 이미지를 화면에 출력하기 위해 cv2.imshow() 함수를 사용합니다.

cv2.imshow(title, image)

- title – str 타입이며 윈도우 창의 제목(title)을 지정합니다.
- image – 화면에 표시할 numpy.ndarray타입 이미지 객체입니다.
- 사용 예:

```
cv2.imshow("Lena", lena_img)
cv2.waitKey()
cv2.destroyAllWindows()
```

일반적으로 imshow()는 waitKey() 함수와 destroyAllWindows() 함수를 같이 사용합니다.

- cv2.waitKey() : 키보드(keyboard) 입력을 대기하는 함수입니다. 인수가 0이면 키(key) 입력까지 무한대기이며 특정 시간 동안 대기하려면 밀리초(milisecond) 값을 넣어주면 됩니다.
- cv2.destroyAllWindows() : 화면에 나타난 모든 윈도우를 종료합니다.

다음 코드는 lena.jpg 파일을 불러와 화면에 출력하는 예입니다.

```
import cv2

fname = 'lena.jpg'
color = cv2.imread(fname, cv2.IMREAD_COLOR)
grayscale = cv2.imread(fname, cv2.IMREAD_GRAYSCALE)
unchanged = cv2.imread(fname, cv2.IMREAD_UNCHANGED)

cv2.imshow('Color', color)
cv2.imshow('Grayscale', grayscale)
cv2.imshow('Unchanged', unchanged)
```

```
11
12  cv2.waitKey()
13  cv2.destroyAllWindows()
```

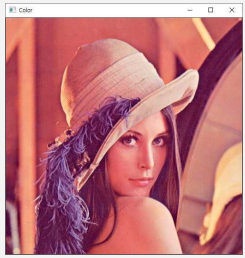


그림 6. COLOR

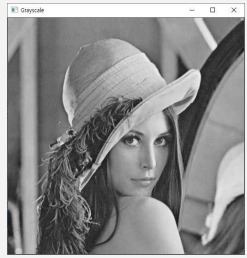


그림 7. GRAYSCALE

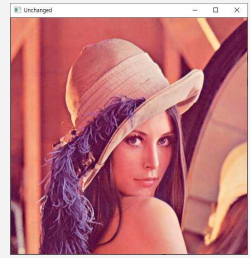


그림 8. UNCHANGED

- jpg 파일은 RGB 색상정보를 가지고 있으므로 위 코드에서 unchanged 변수의 shape[2]의 값은 3이지만 화소에 투명(alpha) 정보가 있는 png 파일이라면 4가 될 수 있습니다.

2.4. cv2.imwrite()로 이미지 저장하기

imwrite() 함수를 이용하면 N-차원 배열을 이미지 파일에 저장할 수 있습니다. 파일의 형식은 파일명의 확장자를 통해 지정됩니다. 만일 파일명이 “lena.jpg”라면 jpg 형식으로 저장되며, “lena.png”이라면 png 형식으로 저장됩니다.

cv2.imwrite(fileName, image) -> retval

- fileName – str 타입이며 저장될 파일명을 지정합니다.
- image – numpy.ndarray 타입이며 저장할 이미지 객체입니다.

- 반환값 retval – 파일에 정상적으로 저장했을 경우 True를 반환하며, 파일에 저장되지 않았을 경우 False를 반환합니다.

다음 코드는 이미지를 그레이스케일로 불러와 파일에 저장합니다.

```
1  import cv2
2
3  img = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
4  cv2.imwrite('lenagray.png', img)
```

```
True
```

2.5. Matplotlib 사용해서 이미지 출력하기

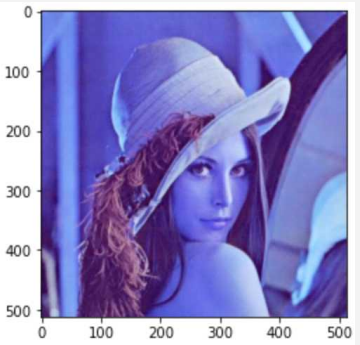
맷플롯립(Matplotlib) 패키지는 다양한 그래프 기능을 가진 파이썬 라이브러리입니다. 맷플롯립 패키지를 이용해서 이미지를 화면에 출력할 수 있습니다. 이 패키지는 이미지를 확대/축소해서 화면에 출력하거나 하나의 화면에 여러 개 이미지를 보고자 할 때 유용하게 사용할 수 있습니다. matplotlib.pyplot 모듈의 imshow() 함수는 N-차원 배열을 화면에 이미지로 출력해 줍니다.

matplotlib.pyplot.imshow(image)

- image – 화면에 출력할 numpy.ndarray 타입 이미지 객체입니다.

다음 코드는 맷플롯립 패키지를 이용해서 이미지를 시각화합니다. 배열이 가져야 할 요소의 값은 0~1 사이의 실수 또는 0~255 사이의 정수여야 합니다.

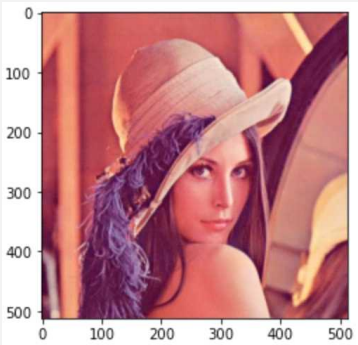
```
1  import cv2
2  from matplotlib import pyplot as plt
3
4  img = cv2.imread('lena.jpg', cv2.IMREAD_COLOR)
5
6  plt.imshow(img)
7  plt.show()
```



레나 영상은 붉은색 계열인데, 출력 결과는 파란색 계열로 보입니다. 그 이유는 OpenCV는 BGR(Blue, Green, Red) 이미지를 사용하지만, 맷플롯립은 RGB(Red, Green, Blue)로 이미지를 보여주기 때문입니다. 그래서 OpenCV로 읽은 파일을 맷플롯립으로 출력하려면 3차원 배열의 값 중 첫 번째와 세 번째 배열의 값을 서로 바꿔 주어야 합니다.

다음 코드는 split() 함수를 이용해 B, G, R 채널별로 데이터 배열을 나누고 merge() 함수를 이용해 다시 하나의 배열로 합칩니다. 그리고 imshow() 함수를 이용해 출력하면 이미지의 색상정보가 정상적으로 출력됨을 알 수 있습니다.

```
1  import cv2
2  from matplotlib import pyplot as plt
3
4  img = cv2.imread('lena.jpg', cv2.IMREAD_COLOR)
5
6  b, g, r = cv2.split(img)
7  img2 = cv2.merge([r, g, b])
8
9  plt.imshow(img2)
10 plt.show()
```



cv2.split() 함수와 cv2.merge() 함수를 이용해서 채널을 분리하고 합쳐 색상 채널의 위치를 바꿀 수 있습니다. 그러나 단순히 BGR 채널의 순서를 RGB 채널로 변환하여 사용할 거라면 cv2.cvtColor() 함수를 사용해서 변환할 수 있습니다. 앞의 코드에서 6, 7라인은 다음처럼 cv2.cvtColor() 함수로 대체할 수 있습니다.

```
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

색상 채널을 바꿔서 출력할거라면 앞 코드의 6~9라인은 다음처럼 한 라인으로 작성할 수 있습니다.

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

cv2.cvtColor() 함수에 대한 더 자세한 내용은 [3.4. 색상 공간 변환]에서 설명합니다.

3절. 디지털 화상과 컬러

3.1. 이진 화상

이진 화상(Binary Image)은 화소를 이진화상은 흰색과 검은색만 표현이 되는 영상입니다. 화소를 1비트로 표현한다면 0 또는 1 값만 갖습니다. 그러나 실제로 우리가 다루는 2진 화상은 8비트로 표현하기 때문에 검은색은 0, 흰색은 255로 표현됩니다.

이진 화상은 원본 이미지를 역치(Threshold)를 이용해서 만드는 것과 이렇게 만들어진 이미지를 밀도를 조절하여 밝기를 표현한 디더링(Dithering) 화상이 있습니다. 디더링의 설명과 구현은 2장에서 합니다.

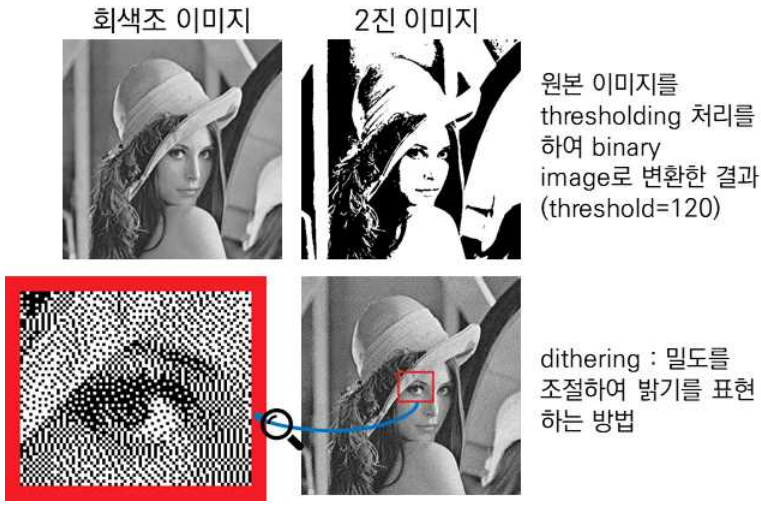


그림 9. 이진 화상의 표현

3.2. 회색조 화상

회색조(Grayscale) 화상은 화소당 8비트, 즉 256단계의 명암(빛의 세기)을 표현할 수 있는 이미지입니다.

대부분의 실습에서 사용하는 이미지가 회색조 화상입니다. '회색조'라는 단어보다 그레이스케일 이미지라고 부르는 경우가 많으므로 이 책에서도 그레이스케일 이미지라고 부르겠습니다. 그레이스케일 이미지는 모노크롬(Monochrome) 이미지라고 부르기도 합니다.

이 과정에서 가장 많이 사용하는 예제 이미지가 레나 이미지입니다. 레나 이미지는 영상처리 사례에 가장 많이 사용되는 이미지 중 하나입니다.

그림 10. Lena 이미지

3.3. 컬러 화상

컬러를 표현하기 위해 RGB(Red, Green, Blue) 모델을 사용하거나 HSV(Hue, Saturation, Value) 모델을 사용할 수 있습니다. 여기에서 모델은 색을 표현하는 방법인 색상 공간(Color space)을 의미합니다.

1) RGB 모델

RGB 모델은 Red, Green, Blue 색을 섞어서 표현합니다.

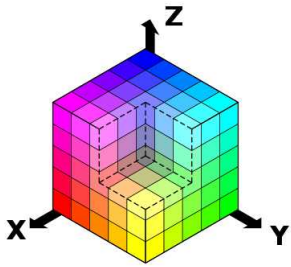


그림 11. RGB 모델

SharkD at English Wikipedia, CC BY-SA 4.0
〈https://creativecommons.org/licenses/by-sa/4.0〉, via Wikimedia Commons
https://commons.wikimedia.org/wiki/File:RGBCube_b.svg

RGB 모델에서 컬러 이미지는 화소의 색을 표현하기 위해서 RGB 각각을 8비트를 사용하므로 화소당 24비트를 사용합니다. 그래서 표현할 수 있는 색은 총 16,777,216가지입니다. 이것을 일반적으로 트루 컬러(True Color) 이미지라고 합니다.

OpenCV에서는 BGR로 표현을 하므로 파란색(Blue)는 (255,0,0)로 표시되며, 초록색(Green)은 (0,255,0), 그리고 빨간색(Red)은

(0,0,255)로 표시됩니다. 흰색(White)이라면 (255,255,255)로 표시되며 검정색(Black)은 (0,0,0)으로 표현할 수 있습니다. 컬러 이미지는 각 화소당 3바이트를 사용하기 때문에 그레이스케일 이미지보다 용량이 큽니다. 이를 해결하기 위해서 룩업테이블(Lookup Table)을 사용하여, 해당 화소에는 인덱스(Index)만 저장하기도 합니다.

2) HSV 모델

HSV 모델은 색상, 채도, 밝기를 이용해 컬러를 표현합니다.

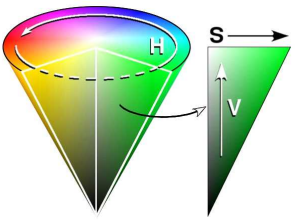


그림 12. HSV 모델

Wapcaplet, CC BY-SA 3.0
〈http://creativecommons.org/licenses/by-sa/3.0/〉, via Wikimedia Commons
https://commons.wikimedia.org/wiki/File:HSV_cone.jpg

HSV 모델에서 H, S, V의 의미는 다음과 같습니다.

- H(Hue) : 색상. 일반적인 색을 의미합니다. 원추모형에서 각도로 표현이 됩니다. (0: Red, 120도 : Green, 240: Blue)
- S(Saturation) : 채도. 색의 순수성을 의미하며 일반적으로 '짙다' 또는 '흐리다'로 표현이 됩니다. 중심에서 바깥쪽으로 이동하면 채도가 높습니다.
- V(Value) : 명도. 색의 밝고 어두운 정도를 의미합니다. 수직축의 깊이로 표현하며 '어둡다'. '밝다'로 표현이 됩니다.

3) YUV 모델

YUV는 밝기에 더 민감하고 색상에는 더 둔감한 점을 고려해서 만든 컬러 모델입니다. OpenCV의 일부 함수는 RGB 모델을 사용한 이미지 객체가 아닌 YUV 모델을 사용한 이미지 객체여야 실행되기도 합니다. Y'UV 모델에서 Y'은 루마(Luma, 밝기) 컴포넌트를 대표하며 U와 V는 2개의 색차(Chrominance, 줄여서 크로마) 컴포넌트(Chroma blue의 U와 Chroma red의 V)를 의미합니다. 휘도는 Y로 나타내며 루마는 프라임 부호(')를 포함한 Y'로 나타냅니다. 휘도는 물리적인 선형 공간 밝기를 의미하고 루마는 영상의 밝기차에 대한 정보를 의미합니다.

Y′UV, YUV, YCbCr, YPbPr 등의 용어가 모호하고 겹치는 부분이 있습니다. YUV와 Y'UV는 방송 시스템의 색 정보의 특정 "아날로그 인코딩"용으로 사용되었고, YCbCr은 비디오 및 이미지 압축과 전송(예: MPEG, JPEG)에 적합한 색 정보의 디지털 인코딩에 사용되었습니다. 지금의 YUV는 YCbCr을 사용하여 인코딩되는 파일 포맷을 기술하기 위해 컴퓨터에서 흔히 사용되고 있습니다.

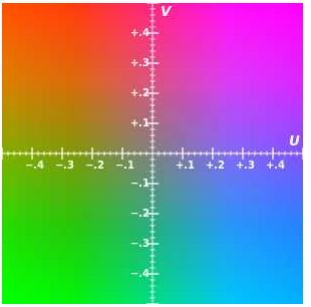


그림 13. YUV 모델에서 U-V 색 플레인

Tonyle, CC BY-SA 3.0 〈http://creativecommons.org/licenses/by-sa/3.0/〉, via Wikimedia Commons
https://commons.wikimedia.org/wiki/File:YUV_UV_plane.svg

3.4. 색상 공간 변환

1) cv2.cvtColor()

cv2.cvtColor() 함수는 색상 공간을 변환합니다. 이 함수를 이용하면 컬러 이미지를 그레이스케일 이미지로 쉽게 변환할 수 있습니다.

cv2.cvtColor(src, flag)

- src - 공간변화를 할 numpy.ndarray 타입 이미지 객체를 지정합니다.
- flag - 변환 방법을 지정합니다. 정수형이며 COLOR_XXX 플래그를 이용합니다.

OpenCV에는 150여 가지 변환 방법이 있습니다. 다음 코드를 이용하면 변환 방법을 지정하는 플래그를 볼 수 있습니다.

```
1  flags = [flag for flag in dir(cv2) if flag.startswith('COLOR_')]
2  print(flags)
```

```
['COLOR_BAYER_BG2BGR', 'COLOR_BAYER_BG2BGRA',
'COLOR_BAYER_BG2BGR_EA', 'COLOR_BAYER_BG2BGR_VNG',
... 생략
```

* 변환 플래그를 모두 출력하기에는 너무 많습니다. 그래서 출력 결과의 일부분만 표시했습니다.

2) 컬러 이미지를 그레이스케일 이미지로 변환하기

컬러 이미지를 그레이스케일 이미지로 변환할 때 (R+G+B)/3을 그레이스케일 이미지의 화소 값으로 사용할 것 같지만 그렇지 않습니다. 컬러 이미지를 그레이스케일 이미지로 변환하는 방법은 여러 가지가 있는데, 보통 밝기 정보(Y)만을 따로 분리해서 전송하고, 색깔 정보(U,V)는 따로 보내는 방법을 사용합니다.

컬러 이미지 그레이스케일 이미지로 변환하는 공식은 YPbPr, YCbCr, YIQ 등이 있습니다[2]. 이 중에서 주로 사용하는 방법은 YPrPb입니다.

다음은 YCrCb 방법과 YPrPb 방법의 변환공식입니다.

$$YCrCb: Y = Red*0.2126 + Green*0.7152 + Blue*0.0722$$

$$YPrPb: Y = Red*0.299 + Green*0.587 + Blue*0.114$$

공식에서 보듯이 R, G, B 각 성분이 밝기에 영향을 주는 정도가 다름을 알 수 있습니다.

다음 코드는 컬러 이미지를 불러와 [0, 0] 위치 화소의 R, G, B값과 YPrPb 방법, YCrCb방법 그리고 cv2.imread() 함수의 회색조 플래그(cv2.IMREAD_GRAYSCALE)에 따른 화소의 값을 출력한 것입니다. 아래 코드대로라면 cv2.imread() 함수에서 회색조로 이미지를 불러올 때의 컬러 변환 방법은 YPrPb임을 알 수 있습니다.

```
1  lena_img = cv2.imread("lena.jpg")
2  B, G, R = lena_img[0][0]
3  print("컬러 화소", lena_img[0][0])
4  print("YPrPb", 0.114*B + 0.587*G + 0.299*R)
5  print("YCrCb", 0.0722*B + 0.7152*G + 0.2116*R)
6  lena_gray = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
7  print("그레이스케일 화소", lena_gray[0][0])
```

```
컬러 화소 [127 136 226]
YPrPb 161.884
YCrCb 154.2582
그레이스케일 화소 162
```

2) HSL, HSV : http://en.wikipedia.org/wiki/HSL_and_HSV
YCrCb: http://en.wikipedia.org/wiki/YCbCr

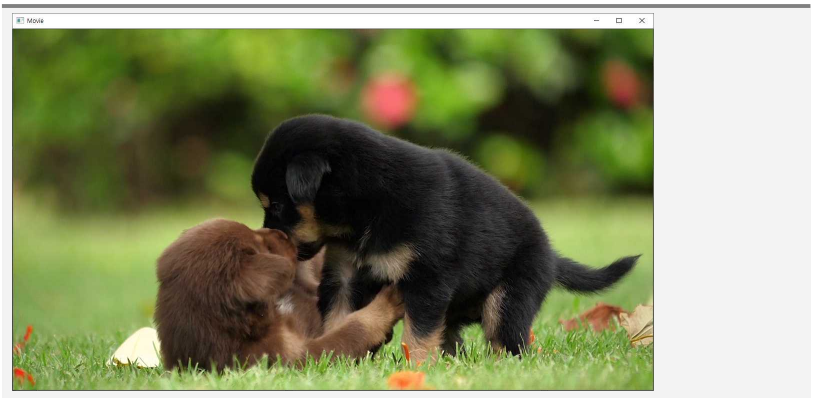
4절. 동영상 처리하기

4.1. 동영상 읽기

동영상을 읽어 처리하려면 cv2.VideoCapture 클래스를 이용합니다. 이 클래스의 생성자 인수로 동영상 파일의 경로를 추가하면 쉽게 동영상을 불러와 처리할 수 있습니다.

동영상을 정상적으로 불러오면 VideoCapture 클래스의 read() 메서드를 이용해서 동영상 내의 프레임을 읽을 수 있습니다. 만일 프레임이 읽히면 True와 해당 프레임의 이미지 객체를 반환합니다. 영상이 끝까지 실행되어 더 이상 읽을 수 있는 프레임이 없으면 read() 함수는 (False, None)을 반환합니다.

```
 1  import cv2
 2
 3  cap = cv2.VideoCapture("Puppies-HD.mp4")
 4  if cap.isOpened():
 5      delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
 6      while True:
 7          ret, img = cap.read()
 8          if ret:
 9              cv2.imshow("Movie", img)
10              if cv2.waitKey(delay) & 0xFF == 27 :
11                  print("ESC Key pressed")
12                  break
13          else:
14              print(ret, img)
15              break
16  else:
17      print("File not opened")
18  cap.release()
19  cv2.destroyAllWindows()
```



예제에서 사용한 동영상은 29.97프레임/초입니다. OpenCV를 이용해서 프레임속도(FPS, Frame Per Second)를 얻으려면 VideoCapture 클래스의 get() 메서드와 cv2.CAP_PROP_FPS 속성을 이용합니다. cap.get(cv2.CAP_PROP_FPS)의 결과는 29.97002997002997입니다.

코드에서 11라인의 waitKey() 함수의 인수는 다음 키를 누를 때까지 대기시간입니다. 대기시간은 밀리초 단위로 정수값을 사용해야 합니다. 만일 1초에 30프레임 동영상이라면 1초는 1000밀리초이므로 지연시간을 $1000 \div 30 \approx 33$으로 지정해야 원래 영상과 같은 속도로 재생됩니다. 이 값은 동영상의 프레임을 얻어 계산된 값을 사용할 수 있습니다.

- 동영상/카메라 지정 시 cv2.VideoCapture(0, cv2.CAP_DSHOW)형식처럼 캡처 개체에 대한 기본 API를 지정해야 할 수 있습니다. 자주 지정하는 API는 cv2.CAP_DSHOW(DirectShow), cv2.CAP_MSMF(Microsoft Media Foundation), CAP_QT(퀵타임) 등이 있습니다.

4.2. 카메라 영상 읽기

1) USP 카메라 영상 읽기

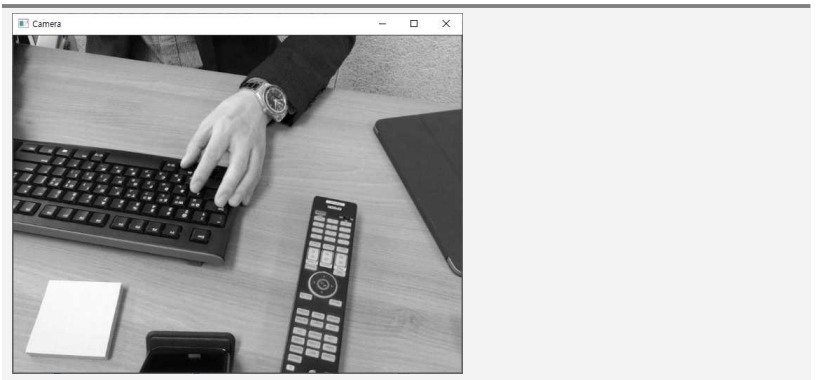
내 컴퓨터의 USP에 연결된 카메라(웹카메라) 정보를 가져오기 위해서는 연결된 카메라 번호만 알면 됩니다.

cap = cv2.VideoCapture(index)

- index : 카메라 번호를 지정합니다. 번호 0은 USB에 연결된 0번 카메라를 열겠다는 의미입니다.

다음 코드는 0번 카메라의 영상을 읽어 그레이스케일로 변환하여 화면에 출력합니다.

```
 1  import cv2
 2
 3  cap = cv2.VideoCapture(0)
 4  if cap.isOpened():
 5      delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
 6      while True:
 7          ret, img = cap.read()
 8          if ret:
 9              img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10              cv2.imshow("Camera", img_gray)
11              if cv2.waitKey(delay) & 0xFF == 27 :
12                  print("ESC Key pressed")
13                  break
14          else:
15              print(ret, img)
16              break
17  else:
18      print("Camera not opened")
19
20  cap.release()
21  cv2.destroyAllWindows()
```



Windows 10에서 웹캠 또는 카메라를 열려면 **설정** ⚙ 〉 **개인정보** 〉 **카메라**를 선택한 다음 **[앱에서 카메라에 액세스하도록 허용]**을 켭니다.

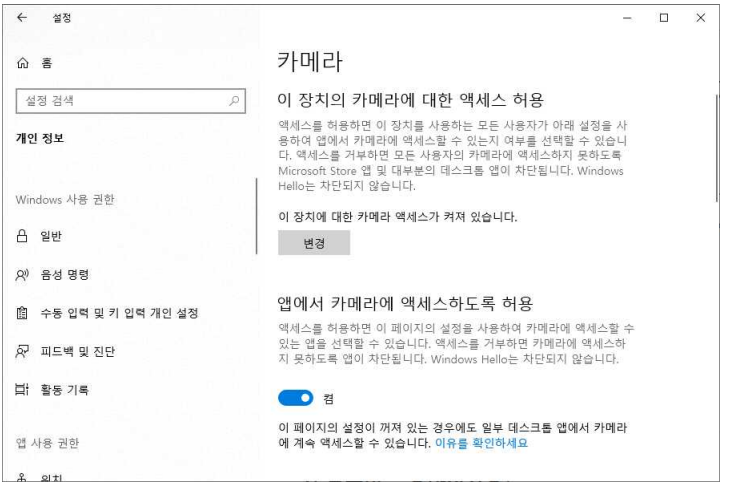


그림 14. 카메라 설정

만일 그래도 카메라 연결이 되지 않는다면 카메라 설정에서 아래로 조금 더 내려가면 [데스크톱 앱에서 카메라를 사용할 수 있도록 허용]을 켜고, Python 항목이 있는지 확인해 보세요. 그래도 연결이 안 되면 카메라에 이상이 있는지 확인해야 합니다.

2) IP 카메라 RTSP 영상 읽기

IP 카메라는 카메라가 직접 인터넷에 연결되어 IP주소를 가지며, RTSP(Real Tile Streaming Protocol) 비디오 스트림을 지원하기 때문에 사용자는 어디에서나 라이브 비디오를 볼 수 있습니다. RTSP는 스트리밍 미디어 서버를 제어하기 위해 설계된 네트워크 프로토콜입니다. 이 프로토콜을 통해 제공되는 영상은 OpenCV에서 쉽게 불러와 영상을 처리할 수 있습니다.

IP 카메라의 영상의 RTSP 비디오 스트림을 받아 처리하려면 다음 형식으로 RTSP 서버 주소를 지정하면 됩니다.

rtsp://사용자아이디:비밀번호@아이피주소:포트번호/프로파일

- 다음은 코드 예입니다.
  rtsp_addr = “rtsp://admin:admin12@111.201.13.53:553/profile1”
  cap = cv2.VideoCapture(rtsp_addr)
- admin:admin12는 RTSP 스트리밍에 연결할 사용자 아이디와 비밀번호 예시 입니다.
- 아이피 주소와 포트번호, 그리고 프로파일 이름은 다를 수 있습니다.
- RTSP 주소는 다음 형식처럼 사용할 수 있습니다.
  rtsp://192.168.0.100:554/user=admin&password=admin12&channel=1&stream=0.sdp

RTSP 카메라를 연결하는 방법은 RTSP 주소와 계정정보만 알면 되므로 코드는 생략하겠습니다.

카메라의 RTSP 주소를 설정하는 방법은 제품마다 다를 수 있으므로 제품 설명서를 참고하셔야 합니다.

4.3. 동영상 속성

cv2.CAP_PROP_로 시작하는 상수들은 동영상의 속성들에 관한 정보를 가지고 있습니다. VideoCapture 클래스의 set(id, value) 메서드와 get(id) 메서드를 이용하면 동영상 속성 정보를 변경하거나 조회할 수 있습니다.

다음 OpenCV의 VideoCapture 속성입니다.

표 11. VideoCapture 속성

동영상 속성	설명
CAP_PROP_POS_MSEC	비디오 파일의 현재 위치(밀리 초)입니다.
CAP_PROP_POS_FRAMES	다음에 디코딩/캡처할 프레임의 인덱스입니다.
CAP_PROP_POS_AVI_RATIO	비디오 파일의 상대 위치 : 0=영화 시작, 1=영화 끝.
CAP_PROP_FRAME_WIDTH	비디오 스트림의 프레임 너비입니다.
CAP_PROP_FRAME_HEIGHT	비디오 스트림의 프레임 높이입니다.
CAP_PROP_FPS	프레임 속도.
CAP_PROP_FOURCC	코덱의 4문자 코드입니다. 코덱에 대한 자세한 내용은 https://www.fourcc.org/codecs.php를 참고하세요.
CAP_PROP_FRAME_COUNT	비디오 파일의 프레임 수입니다.
CAP_PROP_FORMAT	VideoCapture.retrieve()에 의해 반환된 Mat 객체의 형식. 디코딩되지 않은 RAW 비디오 스트림을 가져오려면 값 -1을 설정합니다.
CAP_PROP_MODE	현재 캡처 모드를 나타내는 값입니다.
CAP_PROP_BRIGHTNESS	이미지의 밝기 (지원하는 카메라에만 해당).
CAP_PROP_CONTRAST	이미지의 대비 (카메라에만 해당).
CAP_PROP_SATURATION	이미지의 채도 (카메라에만 해당).
CAP_PROP_HUE	이미지의 색조 (카메라에만 해당).
CAP_PROP_GAIN	이미지 획득 (지원하는 카메라에만 해당).
CAP_PROP_EXPOSURE	노출 (지원하는 카메라에만 해당).
CAP_PROP_CONVERT_RGB	이미지를 RGB로 변환해야 하는지 여부를 나타내는 논리형 플래그입니다.

CAP_PROP_WHITE_BALANCE_BLUE_U	현재 지원되지 않습니다.
CAP_PROP_RECTIFICATION	스테레오 카메라용 정류 플래그
CAP_PROP_MONOCHROME	
CAP_PROP_SHARPNESS	
CAP_PROP_AUTO_EXPOSURE	카메라에 의한 노출 제어, 사용자는 이 기능을 사용하여 기준 레벨을 조정할 수 있습니다.
CAP_PROP_GAMMA	
CAP_PROP_TEMPERATURE	
CAP_PROP_TRIGGER	
CAP_PROP_TRIGGER_DELAY	
CAP_PROP_WHITE_BALANCE_RED_V	
CAP_PROP_ZOOM	
CAP_PROP_FOCUS	
CAP_PROP_GUID	
CAP_PROP_ISO_SPEED	
CAP_PROP_BACKLIGHT	
CAP_PROP_PAN	
CAP_PROP_TILT	
CAP_PROP_ROLL	
CAP_PROP_IRIS	
CAP_PROP_SETTINGS	비디오/카메라 필터 대화 상자 팝업(참고 : 현재 DSHOW 백엔드에서만 지원됩니다. 속성값은 무시 됨)
CAP_PROP_BUFFERSIZE	
CAP_PROP_AUTOFOCUS	
CAP_PROP_SAR_NUM	샘플 종횡비 : num/den(num)
CAP_PROP_SAR_DEN	샘플 종횡비 : num/den(den)
CAP_PROP_BACKEND	현재 백엔드(열거형 VideoCaptureAPI). 읽기 전용 속성입니다.
CAP_PROP_CHANNEL	비디오 입력 또는 채널 번호(지원하는 카메라에만 해당)

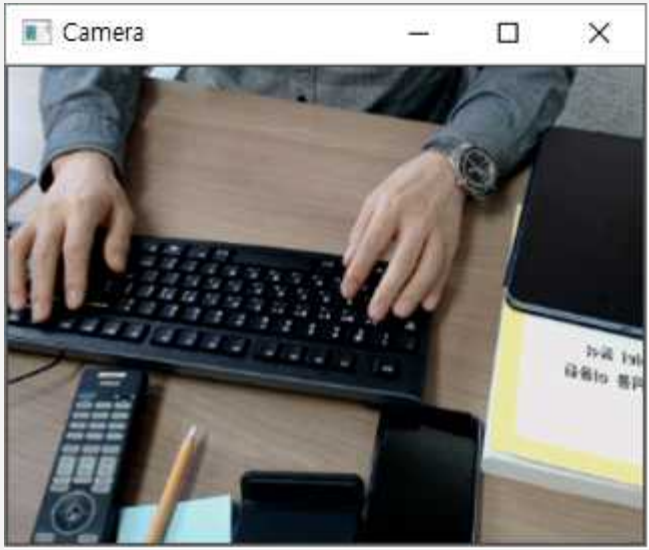
CAP_PROP_AUTO_WB	자동 화이트 밸런스 활성화 / 비활성화
CAP_PROP_WB_TEMPERATURE	화이트 밸런스 색온도
CAP_PROP_CODEC_PIXEL_FORMAT	(읽기 전용) 코덱의 픽셀 형식입니다. 4 문자 코드 cv2.VideoWriter_fourcc() 참고. AV_PIX_FMT_*의 하위 집합 또는 알 수 없는 경우 -1
CAP_PROP_BITRATE	(읽기 전용) 비디오 비트 전송률 (KBits/s)
CAP_PROP_ORIENTATION_META	(읽기 전용) 스트림 메타에 의해 정의된 프레임 회전
CAP_PROP_ORIENTATION_AUTO	True이면 비디오 파일의 메타데이터를 고려하여 CvCapture의 출력 프레임을 회전합니다.
CAP_PROP_HW_ACCELERATION	(개방 전용) 하드웨어 가속 유형. cv2.VideoCapture 생성자의 params 매개변수를 통해서만 지원되는 설정입니다.
CAP_PROP_HW_DEVICE	(개방 전용) 하드웨어 장치 색인입니다.

다음 코드는 웹 카메라의 영상을 화면에 출력할 때 프레임의 크기를 조절한 영상을 출력합니다.

```
1  import cv2
2
3  cap = cv2.VideoCapture(0)
4  print(cap.get(cv2.CAP_PROP_FRAME_WIDTH),
         cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
5  cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
6  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
7  print(cap.get(cv2.CAP_PROP_FRAME_WIDTH),
         cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
8
9  if cap.isOpened():
10     delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
11     while True:
12         ret, img = cap.read()
13         if ret:
```

```
            cv2.imshow("Camera", img)
            if cv2.waitKey(delay) & 0xFF == 27 :
                print("ESC Key pressed")
                break
        else:
            print(ret, img)
            break
else:
    print("Camera not opened")

cap.release()
cv2.destroyAllWindows()
```

```
640.0 480.0
320.0 240.0
30.0
```



4.4. 동영상 프레임 저장하기

동영상의 프레임들은 cv2.imwrite()를 이용해서 이미지 파일로 저장할 수 있습니다.

다음 코드에서는 윈도우를 마우스로 클릭하는 순간 화면을 파일에 저장합니다. 마우스 이벤트에 대해서는 다음 절에서 설명합니다. 파일 이름은 지금 시간의 마이크로초입니다. 10라인의 namedWindow() 함수는 제목을 가진 윈도우를 생성하여 이벤트 소스로 사용하기 위한 것입니다.

```
 1  import cv2
 2  import numpy as np
 3  from datetime import datetime
 4
 5  def mouseHandler(event, x, y, flags, param):
 6      if event==cv2.EVENT_LBUTTONDOWN:
 7          print(event, x, y)
 8          print(datetime.today())
 9          filename = str(datetime.today().microsecond)+".jpg"
10          cv2.imwrite(filename, img)
11
12  cv2.namedWindow("Camera")
13  cv2.setMouseCallback("Camera", mouseHandler)
14
15  cap = cv2.VideoCapture(0)
16
17  img = None
18  if cap.isOpened():
19      delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
20      while True:
21          ret, img = cap.read()
22          if ret:
23              img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
24              cv2.imshow("Camera", img_gray)
```

```
25                if cv2.waitKey(delay) & 0xFF == 27 :
26                    print("ESC Key pressed")
27                    break
28            else:
29                print(ret, img)
30                break
31  else:
32      print("Camera not opened")
33
34  cap.release()
35  cv2.destroyAllWindows()
```

```
29.97002997002997
1 644 310
2021-04-07 16:34:01.027176
* 이 예제는 마우스를 클릭하는 순간의 프레임을 파일에 저장합니다.
위의 실행결과대로라면 이 코드를 실행시킨 디렉토리에 27176.jpg
파일이 저장됩니다. ESC키를 누르면 윈도우가 종료됩니다.
```

윈도우와 관련된 함수는 cv2.namedWindow() 함수만 있는 것이 아닙니다. 윈도우 관련 함수는 다음을 참고하세요.

- cv2.namedWindow(title[, option]) : 이름을 갖는 윈도우 열기. option 매개변수는 다음을 사용할 수 있습니다.
  . cv2.WINDOW_NORMAL - 임의의 크기, 사용자가 창의 크기를 조정할 수 있음
  . cv2.WINDOW_AUTOSIZE - 이미지와 같은 크기, 윈도우의 크기 재조정 할 수 없음. 기본값.
- cv2.moveWindow(titie, x, y) : 윈도우 위치 이동
- cv2.resizeWindow(title, width, height) : 윈도우 크기 변경
- cv2.destroyWindow(title) : 지정한 title 윈도우 닫기
- cv2.destroyAllWindows() : 열린 모든 윈도우 닫기

4.5. 비디오 저장하기

카메라의 프레임들을 cv2.ViderWriter() 클래스와 write() 함수를 이용해서 비디오로 저장할 수 있습니다.

cv2.VideoWriter(outputFile, fourcc, frame, size) -> retval

- outputFile - 저장될 파일명을 지정합니다.
- fourcc - 비디오의 코덱(Codec)정보를 지정합니다. 코덱 정보는 cv2.VideoWriter_fourcc('M','J','P','G') 또는 cv2.VideoWriter_fourcc(*'MJPG) 와 같이 표현할 수 있습니다. 자주 사용하는 코덱은 DIVX, XVID, FMP4, X264, MJPG이며, OS마다 지원하는 코덱은 다릅니다. 더 많은 코덱 정보는 다음 사이트에서 확인할 수 있습니다.
  * https://www.fourcc.org/codecs.php
- frame - 초당 저장될 프레임의 수를 지정합니다.
- size - 저장될 비디오의 크기를 (width, height) 형식으로 지정합니다.

- 반환값 retval - ViderWriter 객체입니다. 이 객체의 write() 함수를 이용해서 비디오를 저장합니다.

cv2.VideoWriter.write(img)

- img - 비디오로 저장할 넘파이 배열입니다. 2차원(그레이스케일)이 아닌 3차원(컬러) 배열이어야 합니다.

다음 코드는 카메라에 보인 영상을 그레이스케일로 변환한 후 비디오 파일로 저장합니다. 저장은 ESC키를 누를 때까지 계속됩니다. 카메라가 준비되어 있지 않다면 3라인의 숫자 0 대신 다른 비디오 파일의 경로를 입력하면 테스트할 수 있습니다. out.write() 함수의 인수로 그레이스케일 이미지를 직접 넣으면 안 됩니다. 다음 코드의 19라인에서 보듯이 그레이스케일 이미지를 BGR 이미지로 변환해서 저장해야 합니다.

```python
import cv2

cap = cv2.VideoCapture(0)

if cap.isOpened():
    fourcc = cv2.VideoWriter_fourcc(*'DIVX')
    width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
    height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
    size = (int(width), int(height))
    fps = cap.get(cv2.CAP_PROP_FPS)
    out = cv2.VideoWriter("video.avi", fourcc, fps, size)

    delay = int(1000/cap.get(cv2.CAP_PROP_FPS))
    while True:
        ret, img = cap.read()
        if ret:
            gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
            cv2.imshow("Puppy", gray)
            out.write(cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR))
            if cv2.waitKey(delay) & 0xFF == 27:
                print("윈도우 종료")
                break
        else:
            print(ret, img)
            break
else:
    print("비디오 안열림")

out.release()
cap.release()
cv2.destroyAllWindows()
```

5절. 이벤트 처리하기

5.1. 마우스 이벤트

OpenCV에는 이미지 처리를 위해서 자체 마우스 이벤트 체계를 가지고 있습니다. OpenCV의 마우스 이벤트를 이해하기 위해서는 이벤트를 발생시킬 이벤트 소스와 이벤트가 발생했을 때 처리할 이벤트 핸들러를 이해해야 합니다.

그림 15. OpenCV 마우스 이벤트

만일 imshow() 함수를 이용해서 이미지를 보이는 윈도우에서 마우스 이벤트를 발생시킨다면 그 윈도우가 이벤트 소스가 됩니다. 그리고 마우스를 클릭하는 등의 조작을 했을 경우 이벤트 처리를 하는 이벤트 핸들러를 콜백(Callback) 함수라고 합니다.

이벤트 소스와 이벤트 핸들러만 있다고 이벤트를 처리할 수 있는 것은 아닙니다. setMouseCallback() 함수를 이용해서 이벤트 소스와 이벤트 핸들러를 연결해 줘야 해당 윈도우에서 이벤트가 발생했을 경우 그 이벤트를 처리할 수 있습니다. 이 함수의 첫 번째 인수는 이벤

트 소스에 해당하는 윈도우의 제목(Title)을 지정하고, 두 번째 인수는 이벤트 핸들러에 해당하는 콜백 함수를 지정합니다.

cv2.setMouseCallback(windowName, callback, param=None)

- windowName – 윈도우의 이름(Title)을 지정합니다.
- callback – 이벤트를 처리할 함수(콜백함수에는 (event, x, y, flags, param)이 전달됩니다.
- param – 콜백함수에 전달되는 데이터입니다.

다음 구문은 콜백 함수입니다. 콜백 함수의 이름은 사용자가 원하는 이름으로 지정해서 사용할 수 있습니다.

***callback*(event, x, y, flags, param)**

- event – cv2.EVENT_로 시작하는 마우스 이벤트 속성이며 [표 16. 마우스 이벤트 속성]을 참고하세요.
- x.y – 마우스 이벤트가 발생된 위치입니다.
- flags – 이벤트가 발생할 때 키보드나 마우스의 추가적인 상태를 알려줍니다. cv2.EVENT_FLAG_로 시작하는 상수 6가지가 있으며 이들의 조합에 따른 결과값이 전달됩니다. 만일 컨트롤키와 쉬프트키를 누른 상태에서 마우스 오른쪽 버튼을 클릭한다면 8+16+1=25가 전달됩니다.

  표 15. 이벤트 플래그

이름	값	의미
EVENT_FLAG_LBUTTON	1	마우스 왼쪽 버튼 누름
EVENT_FLAG_RBUTTON	2	마우스 오른쪽 버튼 누름
EVENT_FLAG_MBUTTON	4	마우스 가운데 버튼 누름
EVENT_FLAG_CTRLKEY	8	컨트롤(Ctrl) 키 누름
EVENT_FLAG_SHIFTKEY	16	쉬프트(Shift) 키 누름
EVENT_FLAG_ALTKEY	32	알트(Alt) 키 누름

  flags 속성을 이용해서 컨트롤키가 눌려진 상태에서 이벤트가 발생됐는지 알려면 다음 코드처럼 비트 & 연산자를 이용합니다.

  if flags & cv2.EVENT_FLAG_CTRLKEY :
- param – 함수에 전달되는 데이터입니다.

다음 표는 마우스 이벤트가 발생했을 경우 콜백 함수에 전달되는 event 속성입니다. 콜백 함수에서 event 변수에 전달된 값과 아래의 표에 있는 이벤트 속성과 비교해서 해당 이벤트를 처리합니다.

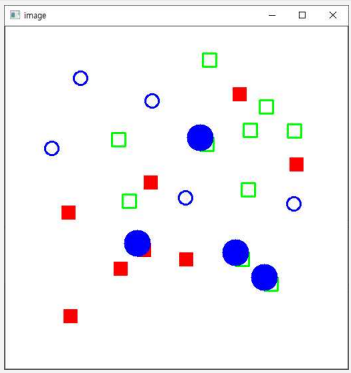
표 16. 마우스 이벤트 속성

속성	의미
EVENT_MOUSEMOVE	마우스 포인터가 윈도우 위에서 움직일 때
EVENT_LBUTTONDOWN	마우스 왼쪽 버튼을 누를 때
EVENT_MBUTTONDOWN	마우스 가운데 버튼을 누를 때
EVENT_RBUTTONDOWN	마우스 오른쪽 버튼을 누를 때
EVENT_LBUTTONUP	마우스 왼쪽 버튼을 뗄 때
EVENT_MBUTTONUP	마우스 가운데 버튼을 뗄 때
EVENT_RBUTTONUP	마우스 오른쪽 버튼을 뗄 때
EVENT_LBUTTONDBLCLK	마우스 왼쪽 버튼을 더블 클릭할 때
EVENT_MBUTTONDBLCLK	마우스 가운데 버튼을 더블 클릭할 때
EVENT_RBUTTONDBLCLK	마우스 오른쪽 버튼을 더블 클릭할 때
EVENT_MOUSEWHEEL	마우스 상하 스크롤을 사용할 때
EVENT_MOUSEHWHEEL	마우스 좌우 스크롤을 사용할 때

다음 코드는 흰색 배경이 있는 윈도우에 마우스 이벤트를 테스트하는 예입니다. 마우스 왼쪽 버튼을 클릭하면 20x20픽셀 크기 초록색 사각형을 그리고, 오른쪽 버튼을 클릭하면 반지름 10픽셀 크기의 파란색 원을 그립니다. 마우스 왼쪽 버튼을 더블클릭하면 안이 채워진 반지름 20픽셀 크기의 파란색 원을 그립니다. 만일 컨트롤키가 눌러진 상태에서 마우스 왼쪽 버튼을 클릭하면 빨간색 속이 채워진 네모를 그립니다.

```
1  import cv2
2  import numpy as np
```

```
3
4  def mouseHandler(event, x, y, flags, param):
5      if event == cv2.EVENT_LBUTTONDOWN:
6          if event & cv2.EVENT_FLAG_CTRLKEY:
7              cv2.rectangle(img, (x,y), (x+20,y+20), (0,0,255),-1)
8          else:
9              cv2.rectangle(img, (x,y), (x+20,y+20), (0,255,0), 2)
10     elif event == cv2.EVENT_RBUTTONDOWN:
11         cv2.circle(img, (x,y), 10, (255,0,0), 2)
12     elif event == cv2.EVENT_LBUTTONDBLCLK:
13         cv2.circle(img, (x,y), 20, (255,0,0), -1)
14
15 img = np.full((512,512,3), 255, dtype=np.uint8)
16 cv2.namedWindow('image')
17 cv2.setMouseCallback('image', mouseHandler)
18
19 while(1):
20     cv2.imshow('image',img)
21     if cv2.waitKey(20) & 0xFF == 27:
22         break
23
24 cv2.destroyAllWindows()
```



5.2. 키보드 이벤트

키보드 이벤트를 처리하기 위해서는 cv2.waitKey() 함수를 사용합니다. 이 함수는 키보드 이벤트에 대해 지정된 밀리 초 동안 기다립니다. 이때 아무 키나 누르면 프로그램이 계속됩니다.

cv2.waitKey(delay=None) -> retval

- delay - 밀리초(ms) 단위 대기시간입니다. 1초=1000밀리초입니다. 만일 delay 값이 기본값(None)이거나 0보다 작거나 같으면(delay<=0) 무한대기합니다.

- 반환값 retval - 이 함수가 반환하는 값은 키보드로 입력한 키의 아스키코드[3]입니다. 만일 입력 대기시간 동안 아무 키도 눌리지 않으면 -1을 반환합니다. 아스키코드 중에서 10진수 27(16진수 1B)은 ESC 키를 의미합니다. 0은 48, A는 65, a는 97에 해당하는 아스키코드값을 가지고 있습니다.

다음 코드는 키가 눌리는 것을 테스트합니다.

```
1  import cv2
2  import numpy as np
3
4  image = np.full((100, 100, 3), 255, np.uint8)
5  while(1):
6      cv2.imshow("Key test", image)
7      key = cv2.waitKey()
8      print(f'Code: {key}, Char: {chr(key)}')
9      if key & 0xFF == 27:
10         break
11
12 cv2.destroyAllWindows()
```

```
Code: 97, Char: a
Code: 98, Char: b
Code: 48, Char: 0
Code: 27, Char:
```

3) http://www.asciitable.com

5.3. 트랙 바 사용하기

트랙바(Trackbar)를 이용하면 값을 쉽게 입력받을 수 있습니다. cv2.createTrackbar()는 트랙바를 만들고, cv2.getTrackbarPos()는 트랙바의 값을 받습니다.

cv2.createTrackbar(trackbarName, windowName, value, count, onChange)

- trackbarName – 트랙바 이름입니다.
- windowName – 트랙바가 등록될 윈도우 이름입니다.
- value – int 타입이며, 트랙바가 생성될 때 초깃값입니다.
- count – 트랙바의 최댓값입니다. 최솟값은 항상 0입니다.
- onChange – 트랙바의 슬라이드가 변경될 때 호출되는 콜백 함수입니다. 전달되는 파라미터는 트랙바의 위치입니다.

cv2.getTrackbarPos(trackbarName, windowName) -> retval

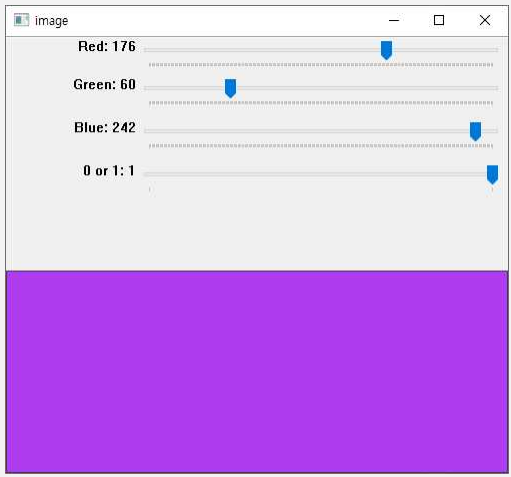
- trackbarName – 트랙바 이름입니다.
- windowName – 트랙바가 등록된 윈도우 이름입니다.

- 반환값 retval – 트랙바의 위치를 반환합니다.

다음 코드는 트랙바를 테스트합니다. 트랙바가 생성될 때는 아무것도 하지 않습니다. 트랙바가 만들어져 윈도우에 등록된 후 ESC키가 눌러질 때까지 트랙바의 값을 가져와 화면에 색을 채웁니다.

```
1  import cv2
2  import numpy as np
3
4  img = np.zeros((200,500,3), np.uint8)
5  cv2.namedWindow('image')
6
7  def set_background(x):
8      global img
9      r = cv2.getTrackbarPos('Red', 'image')
```

```
10      g = cv2.getTrackbarPos('Green', 'image')
11      b = cv2.getTrackbarPos('Blue', 'image')
12      s = cv2.getTrackbarPos('0 or 1', 'image')
13      if s == 0:
14          img[:] = 0
15      else:
16          img[:] = [b,g,r]
17
18
19  cv2.createTrackbar('Red', 'image', 0, 255, set_background)
20  cv2.createTrackbar('Green', 'image', 0, 255, set_background)
21  cv2.createTrackbar('Blue', 'image', 0, 255, set_background)
22  cv2.createTrackbar('0 or 1', 'image', 0, 1, set_background)
23
24  while(1):
25      cv2.imshow('image', img)
26      k = cv2.waitKey(1) & 0xFF
27      if k == 27:
28          break
29
30  cv2.destroyAllWindows()
```



6절. 도형 그리기

OpenCV는 선, 사각형, 원, 타원, 다각형 등 도형을 그릴 수 있는 함수와 이미지에 텍스트를 표현할 수 있는 함수를 제공합니다.

6.1. 선 그리기

cv2.line() 함수는 선을 그려줍니다.

cv2.line(img, start, end, color, thickness=1)

- img – 도형을 그릴 이미지 객체입니다.
- start – 선의 시작 좌표입니다. (x, y) 형식으로 지정합니다.
- end – 선의 끝 좌표입니다. (x, y) 형식으로 지정합니다.
- color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.
- thickness – 선의 두께를 픽셀단위 정수로 지정합니다

다음 코드는 선 그리기 예입니다.

```
1  import cv2
2  import numpy as np
3
4  img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6  cv2.line(img, (50, 10), (200, 10), (255, 0, 0), 1)
7  cv2.line(img, (50, 60), (200, 60), (0, 255, 0), 2)
8  cv2.line(img, (50, 110), (200, 110), (0, 0, 255), 3)
9  cv2.line(img, (50, 160), (200, 160), (0, 0, 0), 4)
10 cv2.line(img, (50, 210), (200, 210), (255, 0, 255), 5)
11
12 cv2.imshow('Lines',img)
13 cv2.waitKey()
14 cv2.destroyAllWindows()
```

6.2. 사각형 그리기

cv2.rectangle() 함수는 사각형을 그려줍니다.

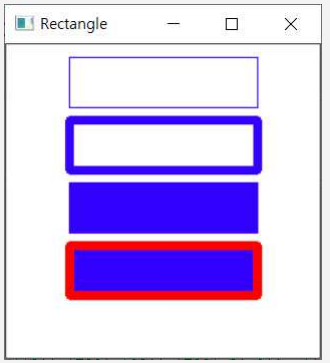
cv2.rectangle(img, start, end, color, thickness=1)

- img – 도형을 그릴 이미지 객체입니다.
- start – 사각형의 왼쪽 위 좌표입니다. (x, y) 형식으로 지정합니다.
- end – 사각형의 오른쪽 아래 좌표입니다. (x, y) 형식으로 지정합니다.
- color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.
- thickness – 선의 두께를 픽셀단위 정수로 지정합니다. -1이면 사각형의 안쪽을 채웁니다.

다음 코드는 사각형을 그리는 예입니다.

```
1  import cv2
2  import numpy as np
3
4  img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6  cv2.rectangle(img, (50, 10), (200, 50), (255, 0, 0))
```

```
 7  cv2.rectangle(img, (50, 60), (200, 100), (255, 0, 0), 5)
 8  cv2.rectangle(img, (50, 110), (200, 150), (255, 0, 0), -1)
 9  cv2.rectangle(img, (50, 160), (200, 200), (255, 0, 0), -1)
10  cv2.rectangle(img, (50, 160), (200, 200), (0, 0, 255), 5)
11
12  cv2.imshow('Rectangle',img)
13  cv2.waitKey()
14  cv2.destroyAllWindows()
```



6.3. 원 그리기

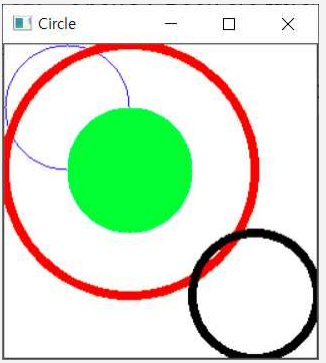
cv2.circle() 함수는 원을 그려줍니다.

cv2.circle(img, center, radius, color, thickness=1)

- img – 도형을 그릴 이미지 객체입니다.
- center – 원의 중심 좌표입니다. (x, y) 형식으로 지정합니다.
- radius – 반지름을 픽셀단위 정수로 지정합니다.
- color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.
- thickness – 선의 두께를 픽셀단위 정수로 지정합니다. -1이면 원의 안쪽을 채웁니다.

다음 코드는 원을 그리는 예입니다.

```
1  import cv2
2  import numpy as np
3
4  img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6  cv2.circle(img, (50, 50), 50, (255, 0, 0))
7  cv2.circle(img, (100, 100), 50, (0, 255, 0), -1)
8  cv2.circle(img, (100, 100), 100, (0, 0, 255), 5)
9  cv2.circle(img, (200, 200), 50, (0, 0, 0), 5)
10
11 cv2.imshow('Circle',img)
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```



원을 그리거나 사각형을 그릴 때 코드를 작성하고 실행결과를 확인한 후 더 많은 원과 사각형을 그려보면서 좌표에 대한 이해를 높일 수 있으므로 코드를 수정하거나 새로운 원과 사각형을 그려보는 실습을 하는 것을 권합니다.

6.4. 타원과 호 그리기

cv2.ellipse() 함수는 타원과 호를 그려줍니다.

cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color, thickness=1)

- img – 도형을 그릴 이미지 객체입니다.
- center – 타원의 중심 좌표입니다. (x, y) 형식으로 지정합니다.
- axes – 타원의 중심에서 가장 큰 거리와 작은 거리를 픽셀단위 정수로 지정합니다. (max, min) 형식으로 지정합니다.
- angle – 타원의 기울기 각도입니다.
- startAngle – 호의 시작 각도입니다. 아래 그림을 참고하세요.
- endAngle – 호의 끝 각도입니다. 아래 그림을 참고하세요.
- color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.
- thickness – 선의 두께를 픽셀단위 정수로 지정합니다. -1이면 타원의 안쪽을 채웁니다.

다음 그림은 ellipse() 함수의 파라미터들에 대한 이해를 도와줍니다.

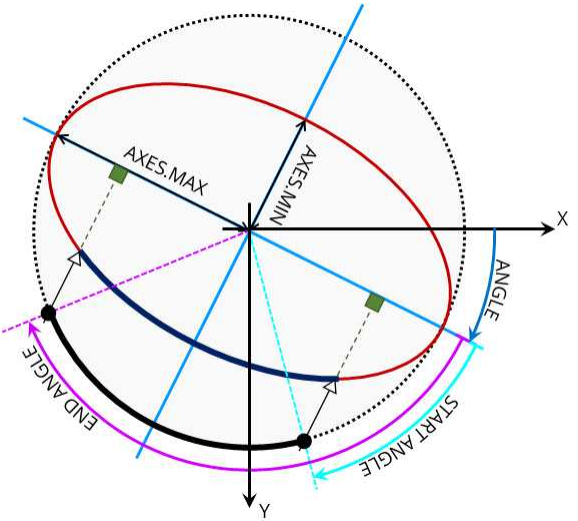
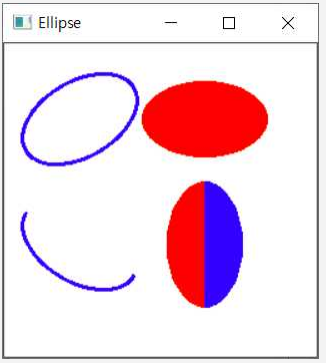


그림 17. cv2.ellipse() 함수의 파라미터

타원의 시작 각도(startAngle)가 0이고, 끝 각도(endAngle)가 360이면 타원이 되고 그렇지 않으면 호가 만들어집니다. ellipse() 함수는 다른 함수에 비해 많은 파라미터를 설정해야 하므로 매개변수들에 대해 어디에 어떻게 적용되는지 예제를 실행시켜보고 이해하시기 바랍니다.

다음 코드는 타원과 호를 그리는 예입니다. 코드에서 각도와 축의 크기 등을 바꿔보고 도형이 어떻게 그려지는지 확인해 보세요.

```
1  import cv2
2  import numpy as np
3
4  img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6  cv2.ellipse(img, (60,60), (50,30), -30, 0, 360, (255,0,0), 2)
7  cv2.ellipse(img, (160,60), (50,30), 0, 0, 360, (0,0,255), -1)
8  cv2.ellipse(img, (60,160), (50,30), 30, 0, 180, (255,0,0), 2)
9  cv2.ellipse(img, (160,160), (50,30), 90, 0, 180, (0,0,255), -1)
10 cv2.ellipse(img, (160,160), (50,30), 90, 180, 360, (255,0,0), -1)
11
12 cv2.imshow('Ellipse',img)
13 cv2.waitKey()
14 cv2.destroyAllWindows()
```



6.5. 다각형 그리기

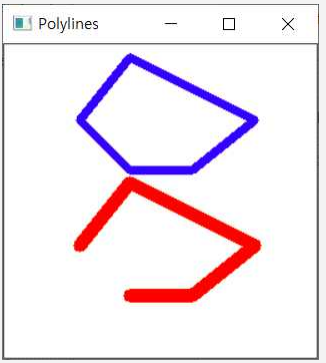
cv2.polylines() 함수는 다각형을 그려줍니다. isClosed 속성값에 따라 닫힌 다각형을 그려줍니다.

cv2.polylines(img, points, isClosed, color, thickness=1)

- img – 도형을 그릴 이미지 객체입니다.
- points – 다각형 꼭지점의 좌표들입니다. [[[x1,y1], [x2,y2]....]]] 형식으로 지정합니다.
- isClosed – True이면 닫힌 도형이 됩니다.
- color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.
- thickness – 선의 두께를 픽셀단위 정수로 지정합니다. 기본값은 1입니다. 음수는 사용할 수 없습니다. 그러므로 속이 채워진 다각형은 그릴 수 없습니다.

다음 코드는 다각형을 그리는 예입니다. 8라인의 points + [0, 100]은 points 점들의 y값만 100 증가시킵니다.

```
1  import cv2
2  import numpy as np
3
4  img = np.full((250, 250, 3), 255, dtype=np.uint8)
5  points = np.array([[60,60], [100,10], [200,60], [150,100],
                      [100, 100]], dtype=np.int32)
6
7  cv2.polylines(img, [points], True, (255, 0, 0), 5)
8  cv2.polylines(img, [points+[0,100]], False, (0, 0, 255), 10)
9
10 cv2.imshow('Polylines',img)
11 cv2.waitKey()
12 cv2.destroyAllWindows()
```



6.6. 문자열 표시하기

cv2.putText() 함수는 문자열을 표시해줍니다.

cv2.putText(img, text, org, font, fontScale, color)

- img – 도형을 그릴 이미지 객체입니다.
- text – 표시할 문자열입니다.
- org – 문자열이 표시될 위치입니다. 문자열의 왼쪽 아래 모서리 지점이 기준입니다.
- font – 폰트를 지정합니다. cv2.FONT_XXX 형식의 속성으로 지정합니다.
- fontScale – 폰트의 크기를 지정합니다.
- color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.

다음 코드는 문자열을 표시하는 예입니다.

```
1  import cv2
2  import numpy as np
3
4  img = np.full((250, 250, 3), 255, dtype=np.uint8)
```

```
5
6  cv2.putText(img, "Hello Python", (10,50),
               cv2.FONT_ITALIC, 1, (255,0,0))
7  cv2.putText(img, "Hello Python", (10,80),
               cv2.FONT_ITALIC, 0.5, (0,0,255))
8  cv2.putText(img, "Hello Python", (10,150),
               cv2.FONT_HERSHEY_DUPLEX, 2, (0,0,0))
9  cv2.putText(img, "Hello Python", (10,200),
               cv2.FONT_HERSHEY_TRIPLEX, 1, (0,0,0))
10
11 cv2.imshow('Text',img)
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```

2장. 영상 기본 연산

1절. 영상 기본 정보

1.1. 영상 기본 속성

OpenCV의 imread() 함수를 이용해서 이미지를 읽으면 넘파이 배열로 반환합니다. 영상처리를 잘 하려면 배열을 잘 다뤄야 하는데 이것은 넘파이 패키지를 잘 다뤄야 한다는 것과 같습니다.

다음 코드의 shape, size, dtype 속성은 각각 배열의 모양, 배열의 모든 요소의 수, 그리고 자료형을 출력합니다.

```
1  import cv2
2
3  img = cv2.imread("lena.jpg")
4  print(img.shape)
```

```
(512, 512, 3)
```

```
1  print(img[0,0])
```

```
[127 136 226]
```

```
1  print(img.size)
```

```
786432
```

```
1  print(img.dtype)
```

```
unit8
```

영상처리에 사용하는 N-차원 배열은 한 화소의 값이 0~255 값만 가지면 되므로 부호(음수)를 갖지 않는 정수형(Unsigned int)으로 저장합니다. 넘파이에서는 이러한 타입으로 np.uint8 자료형을 정의해 놓았습니다. np.uint8 타입은 8비트이지만 음수를 사용하지 않으므로 가질 수 있는 값은 0~255입니다.

1.2. 영상 화소 처리하기

그레이스케일 이미지일 경우 shape 속성은 이미지의 행의 수와 열의 수를 반환합니다. 행이 수는 이미지의 높이(height)가 되며, 열의 수는 이미지의 폭(width)이 됩니다. 그리고 이미지의 가장 왼쪽 위가 기준점 (0, 0)에 해당하면 오른쪽으로 갈수록 화소의 x 인덱스가 증가하며, 아래쪽으로 갈수록 화소의 y 인덱스가 증가합니다. 그래서 이미지에서 (x, y) 좌표 위치의 화소를 넘파이 배열을 이용해 참조하려면 img[y, x] 형식으로 인덱싱합니다.

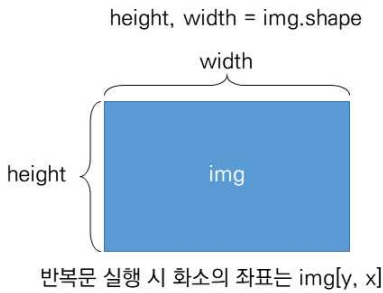


그림 1. 이미지 폭과 높이

다음은 레나 영상의 짝수행을 검은 선으로 처리하는 반복문을 작성해서 이미지의 화소를 어떻게 처리하는지 보여줍니다.

```
1  import cv2
2
3  img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
4  height, width = img.shape
5
6  for y in range(height) :
7      for x in range(width) :
8          if(y%2==0) :
9              img[y, x] = 0
```

영상처리 시 모든 화소를 비교하면서 반복문을 실행시킬 필요는 없습니다. 다음 코드는 앞의 방법과 같은 결과를 얻습니다. 그러나 실행시간은 다릅니다.

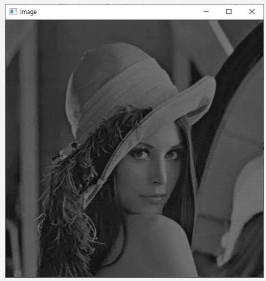
```
for y in range(height) :
    if(y%2==0) :
        for x in range(width) :
            img[y, x] = 0
```

조건에 따라 특정 행을 반복 처리하지 않을 수 있으며, 때로는 인덱싱이 아닌 슬라이싱으로 값을 할당할 수도 있습니다. 다음 코드는 앞의 두 코드와 같은 결과를 출력합니다. 그러나 실행시간은 이 코드가 더 빠릅니다. 이처럼 같은 결과를 얻는 코드이지만 작성하는 사람에 따라 다르게 작성할 수 있고 성능 또한 다를 수 있습니다.

```
for y in range(height) :
    if(y%2==0) :
        img[y, :] = 0
```

다음은 레나 영상의 짝수 번째 행을 0으로 처리한 영상을 출력합니다.

```
cv2.imshow("Lena", img)
cv2.waitKey()
cv2.destroyAllWindows()
```



OpenCV의 함수는 C++로 구현되어 있으므로 파이썬 언어를 이용해서 구현하는 것보다 OpenCV의 함수를 사용하는 것이 더 빠르고 편합니다. 그러나 영상처리 알고리즘을 배우기 위한 목적으로 직접 구현해 사용할 수 있을 것입니다. 그럴 때마다 구현한 코드를 함수로 만들어 두면 코드의 재사용성을 높일 수 있습니다.

1.3. 함수 만들어 사용하기

영상처리를 위해 학습된 내용은 함수로 만들어 두면 언제든지 재사용하기 편합니다. 다음 코드는 이미지의 짝수 행을 검은색 선으로 만드는 함수입니다.

```
import numpy as np

def black_lines(img):
    height, width = img.shape
    img_ = np.zeros(img.shape, dtype=np.uint8)

    for y in range(height):
        if(y % 2 == 0):
            img_[y, :] = 0
        else:
            img_[y, :] = img[y, :]
    return img_
```

다음 코드는 이미지 처리 함수와 이미지 객체를 인수로 받아 화면에 출력하는 코드입니다.

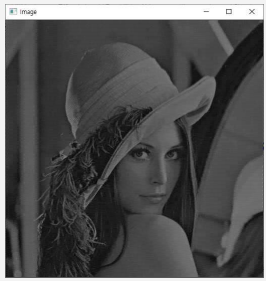
```
1  def img_pro(func, img, *args, show=True, **kwargs):
2      img_ = func(img, *args, **kwargs)
3      if show:
4          cv2.imshow("Image", img_)
5          cv2.waitKey()
6          cv2.destroyAllWindows()
7      else:
8          return img_
```

다음은 앞에서 정의한 img_pro() 함수의 매개변수들에 대한 설명입니다.

- func : 영상처리를 구현한 함수 이름입니다.
- img : 영상처리에 사용할 이미지 객체입니다.
- *args : func() 함수의 인수로 전달될 튜플 매개변수입니다.
- show : True일 경우 윈도우에 결과를 보여주며, False일 경우 그 결과를 반환합니다.
- kwargs : func() 함수의 인수로 전달될 딕셔너리 매개변수입니다.

다음은 함수를 사용하는 코드입니다. img_pro() 함수의 show 매개변수를 기본값(True)으로 하면 윈도우에 이미지를 보여줍니다.

```
1  img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
2  img_pro(black_lines, img)
```



만일 이미지에 함수를 적용한 결과를 받으려면 다음처럼 show 매개변수를 False를 지정하세요.

```
1  img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
2  t = img_pro(black_lines, img)
```

만일 여러분이 맷플롯립(Matplotlib) 패키지를 이용해서 시각화하길 원하면 다음처럼 맷플롯립의 imshow() 함수를 이용해서 이미지를 출력하는 함수를 만들어 사용할 수도 있습니다.

```
1  import cv2
2  import matplotlib.pyplot as plt
3
4  def img_pro2(func, img, *args, show=True, win=False, **kwargs):
5      img_ = func(img, *args, **kwargs)
6      if show:
7          if win:
8              cv2.imshow("Image", img_)
9              cv2.waitKey()
10             cv2.destroyAllWindows()
11         else:
12             fig, axes = plt.subplots(1,2)
13             axes[0].imshow(img, cmap="gray")
14             axes[0].axis("off")
15             axes[0].set_title("origin")
16             axes[1].imshow(img_, cmap="gray", interpolation=None)
```

```
            axes[1].axis("off")
            axes[1].set_title("target")
    else:
        return img_
```

```
img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
img_pro2(black_lines, img)
```

● 이 함수가 반드시 필요한 것은 아닙니다. 필요에 따라 작성해서 사용하세요.

1.4. ROI

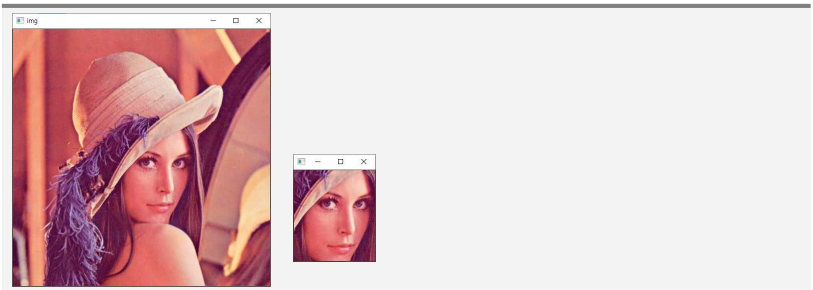
이미지 작업 시 전체 영역의 화소 단위 처리보다 특정 영역 단위로 작업을 하게 됩니다. 원본 이미지에서 처리할 관심 영역을 ROI(Region of Interest)라고 합니다. ROI 설정은 넘파이의 인덱싱(Indexing) 방법을 사용합니다.

```
img[y_from:y_to, x_from:x_to]
```

from 인덱스부터 to 인덱스까지 부분 배열을 선택합니다. 이 구문은 from에 해당하는 인덱스는 포함하지만, to 인덱스는 포함하지 않습니다.

다음 코드는 원본 영상의 일부 영역을 넘파이 배열의 슬라이싱을 이용해 분리한 후 화면에 출력한 예입니다.

```
import cv2

img = cv2.imread('lena.jpg')
temp = img[200:380, 200:360]  # img[y1:y2, x1:x2] 형식임

cv2.imshow('img', img)
cv2.imshow('roi_img', temp)
cv2.waitKey()
cv2.destroyAllWindows()
```



2절. 기본 연산

2.1. 이미지 채널

컬러 이미지는 3개의 채널 B(Blue), G(Green), R(Red)로 구성이 되어 있습니다. 이것을 채널별로 분리할 수 있습니다. 다음 코드는 cv2.split() 함수를 이용해서 채널을 분리합니다. 그리고 각 채널의 평균값을 출력합니다.

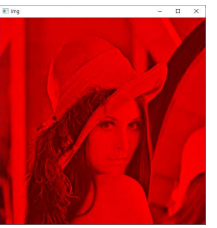
```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread('lena.jpg')
5  b, g, r= cv2.split(img)
6  print(np.mean(b), np.mean(g), np.mean(r), sep=", ")
```

```
105.54556274414062, 98.9676513671875, 180.31714248657227
```

위 코드의 결과에서 Red 채널의 평균값이 가장 크다는 것은 전체적으로 이미지 화소들의 Red 값이 전체적으로 큰 값임을 알 수 있고 이것은 이미지가 대체로 붉은색 성분이 많이 있다는 것을 의미입니다.

cv2.split() 함수를 이용하는 것보다 넘파이의 인덱싱 또는 슬라이싱을 사용하는 것이 더 효율적입니다. 다음 코드는 레나 영상에서 Blue 채널과 Green 채널을 0으로 변경합니다. 이렇게 하면 남는 것은 Red 채널입니다. 이것을 화면에 출력해 봅니다.

```
1  img = cv2.imread('lena.jpg')
2  img[:, :, [0,1]] = 0
3
4  cv2.imshow('img', img)
5  cv2.waitKey()
6  cv2.destroyAllWindows()
```



2.2. 이미지 합치기

1) cv2.add()와 numpy.add()

두 이미지를 합해서 하나의 이미지로 만들기 위해서 cv2.add() 함수와 numpy.add() 함수를 이용할 수 있습니다. 둘 다 각 영상의 화소값을 더하는 것은 같지만 더한 화소의 값이 255보다 크면 cv2.add()와 numpy.add()의 실행 결과는 다릅니다. OpenCV의 add()는 Saturation 연산을 하고, 넘파이의 add()는 Modulo 연산을 합니다.

Saturation 연산은 한계값을 정하고 그 값을 벗어나는 경우는 모두 특정 값으로 계산합니다. 영상처리에서 Saturation은 화소의 값이 0 이하는 모두 0, 255 이상은 모두 255로 표현하는 것입니다.

Modulo 연산은 나머지 연산을 적용합니다. 그래서 넘파이의 add()는 이미지에서는 연산의 결과가 256보다 크거나 같은 경우는 256으로 나눈 나머지 값을 화소의 값으로 결정합니다.

그림 2. Saturation과 Modulo

다음 코드는 OpenCV의 add() 함수와 넘파이의 add() 함수의 실행 결과를 비교합니다.

```
1  import cv2
2  import numpy as np
3
4  img1 = cv2.imread('flower1.jpg')
5  img2 = cv2.imread('flower2.jpg')
6
7  cv2.imshow('Image1', img1)
8  cv2.imshow('Image2', img2)
9  cv2.imshow('cv2.add', cv2.add(img1, img2))
10 cv2.imshow('np.add', np.add(img1, img2))
11 cv2.waitKey()
12 cv2.destroyAllWindows()
```

Flower 1 Flower 2

Saturation(cv2.add()) Modulo(numpy.add())

2) 이미지 블랜딩

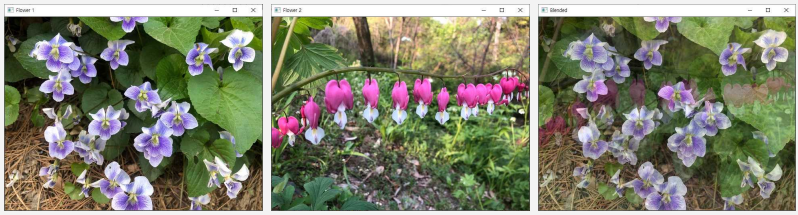
OpenCV의 add()와 넘파이의 add()는 두 이미지를 하나의 이미지로 합치면 오히려 원본 이미지를 알아보지 못할 경우가 많습니다. 두 이미지를 하나의 이미지로 합칠 때 두 이미지에 가중치를 두어 합치면 가중치가 더 큰 이미지가 상대적으로 더 뚜렷하게 보이도록 할 수 있습니다. 이것을 이미지 블랜딩(Blending)이라고 합니다.

다음 코드는 두 이미지에 가중치를 부여해서 합치는 함수입니다.

```
import numpy as np

def weighted_blending(img1, img2, img1_weight=0.5):
    output = np.zeros(img1.shape, dtype=np.uint8)
    height, width = img1.shape[0:2]
    if len(img1.shape)==2:
        for y in range(height):
            for x in range(width):
                output[y,x] = int(img1[y,x]*img1_weight +
                                  img2[y,x]*(1-img1_weight))
    elif len(img1.shape)==3:
        for y in range(height):
            for x in range(width):
                blended = img1[y,x]*img1_weight +
                          img2[y,x]*(1-img1_weight)
                output[y,x] = blended.astype(np.uint8)

    return output
```

다음 코드는 이미지 파일 두 개를 불러와 가중치를 부여해서 합칩니다. 이 예에서는 첫 번째 그림에 가중치 70%를 주었습니다. 그러면 두 번째 이미지에는 가중치가 30% 주어집니다.

```python
img1 = cv2.imread("flower1.jpg")
img2 = cv2.imread("flower2.jpg")

new_img = weighted_blending(img1, img2, img1_weight=0.7)

cv2.imshow('Image 1', img1)
cv2.imshow('Image 2', img2)
cv2.imshow('Blended', new_img)
cv2.waitKey()
cv2.destroyAllWindows()
```



이미지 블랜딩은 넘파이의 브로드캐스팅 규칙을 적용하면 더 쉽게 할 수 있습니다. 다음 코드는 브로드캐스팅 규칙을 이용해서 가중치를 부여한 이미지 블랜딩 예입니다.

```python
import numpy as np

def ez_blending(img1, img2, img1_weight=0.5):
    output = img1*img1_weight + img2*(1-img1_weight)
    return output.astype(np.uint8)
```

넘파이의 브로드캐스팅 규칙은 두 배열을 이용해 연산을 수행할 때 두 배열의 크기가 같지 않아도 어느 한 배열의 축의 크기가 1일 경우에 나머지 배열의 축의 크기만큼 배열의 요소들이 복재되어 연산되는 것을 말합니다. 예를 들면 아래처럼 4x3 배열과 1x3 배열을 더할 때 1x3 배열의 행들이 4x3배열의 크기만큼 행이 복제되어 연산되는 것을 의미합니다.

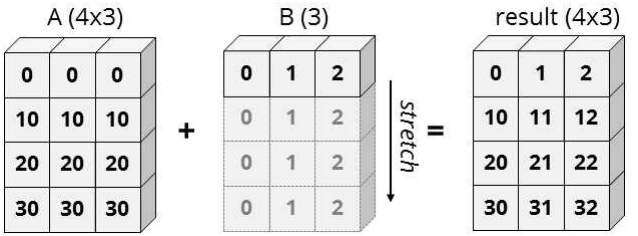


그림 3. 브로드캐스팅 규칙

브로드캐스팅 규칙은 '연산의 두 배열에 대한 후미 축의 크기가 같거나 둘 중 하나가 1이어야 한다.'입니다.

3) 비트연산

비트 연산을 통해 두 이미지를 합칠 수 있습니다. 비트 and, 비트 or, 비트 not, 비트 xor 연산을 통해 특정 영역을 삭제하거나 반전시키는 등의 작업을 할 수 있습니다.

표 22. 비트연산

p1	p2	bitwise_and (p1, p2)	bitwise_or (p1, p2)	bitwise_not (p1)	bitwise_xor (p1, p2)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

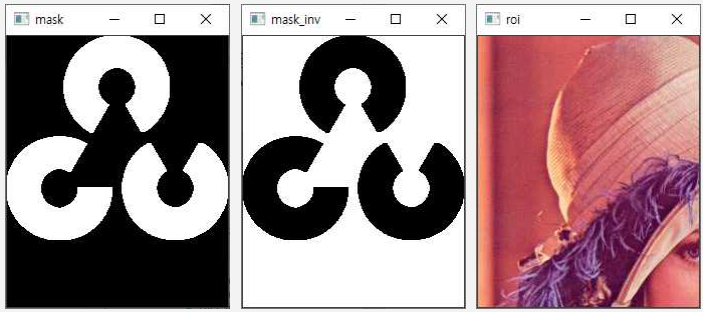
OpenCV 로고 이미지와 레나 영상을 비트 연산을 이용해서 합치는 예를 설명하겠습니다. 다음 코드는 삽입할 이미지의 row, col, channel 정보를 추출하고, 대상 이미지에서 삽입할 이미지의 영역을 추출합니다. 그리고 마스크(mask) 영역을 만들기 위해서 회색조 이미지로 바꾼 다음 이진 이미지로 만듭니다. mask 변수는 로고 부분이

흰색(255)이며 바탕은 검은색(0)입니다. 그리고 mask_inv 변수는 로고 부분이 검은색(0), 바탕은 흰색(255)입니다.

```
import cv2

logo = cv2.imread('logo.png')
lena = cv2.imread('lena.jpg')

rows, cols, channels = logo.shape
roi = lena[50:rows+50, 50:cols+50]

logogray = cv2.cvtColor(logo, cv2.COLOR_BGR2GRAY)
_, mask = cv2.threshold(logogray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)
```

비트 반전에 사용하는 함수가 cv2.bitwise_not()입니다. 다음 코드는 mask 변수와 mask_inv 이미지의 차이점과 레나 영상의 ROI 영역을 확인하기 위해 화면에 출력하는 예입니다.

```
cv2.imshow('mask', mask)
cv2.imshow('mask_inv', mask_inv)
cv2.imshow('ROI, roi)
cv2.waitKey()
cv2.destroyAllWindows()
```



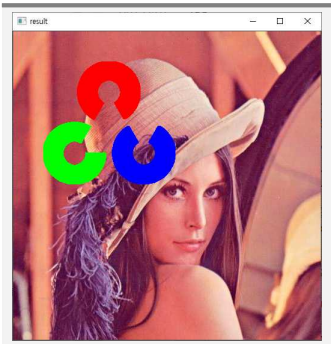
검은색 영역은 화소의 값이 0이기 때문에 이 영역과 비트 and 연산되는 영역은 모두 0이 되어 화소 정보가 사라지게 됩니다.

bitwise_and 연산자는 둘 다 0이 아닌 경우만 값을 통과시킵니다. 즉 마스크가 검은색이 아닌 경우만 통과가 되기 때문에 마스크 영역 이외는 모두 제거됩니다.

아래 코드에서 logo_fg는 배경이 제거되고 로고 영역만 남게 되며, lena_bg는 레나 영상의 ROI에서 OpenCV 로고 파일의 로고 영역에 해당하는 곳이 제거되고 OpenCV의 배경 영역만 남게 됩니다. 2개의 이미지를 합치면 바탕은 제거되고 로고만 레나영상에 합쳐집니다.

이렇게 합쳐진 이미지를 레나 영상의 ROI 영역에 매핑시킵니다.

```
logo_fg = cv2.bitwise_and(logo, logo, mask=mask)
lena_bg = cv2.bitwise_and(roi, roi, mask=mask_inv)
dst = cv2.add(logo_fg, lena_bg)
lena[50:rows+50, 50:cols+50] = dst
cv2.imshow('result', lena)
cv2.waitKey()
cv2.destroyAllWindows()
```



4) cv2.seamlessClone()

만일 두 이미지가 조화되어 합쳐지길 원한다면 비트연산을 이용하는 것보다 cv2.seamlessClone() 함수를 이용하는 것이 좋습니다.

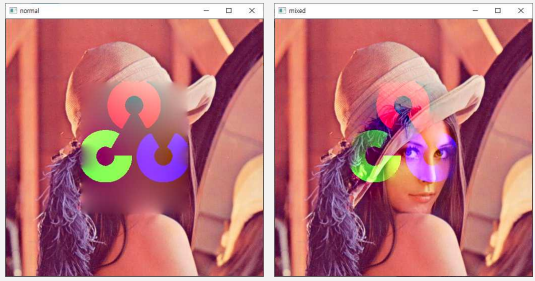
cv2.seamlessClone (src, dst, mask, point, flags)

- src – 대상 이미지로 복제될 소스 이미지입니다. 다음 예에서는 로고(logo.jpg)입니다.
- dst – 소스 이미지가 복제될 대상 이미지입니다. 다음 예에서는 레냐(lena.jpg) 이미지입니다.
- mask – 복제하려는 개체 주변의 마스크입니다. 정확한 마스크 영역을 생성할 필요는 없습니다. 대략적인 영역을 선택해도 됩니다.
- point – 대상 이미지에서 소스 이미지의 중앙 위치입니다.
- flags – cv2.NORMAL_CLONE 또는 cv2.MIXED_CLONE을 지정할 수 있습니다. 두 플래그의 차이점은 실행 결과를 통해 알 수 있습니다.

다음 코드는 cv2.seamlessClone() 함수를 이용해서 두 이미지를 합칩니다. 결과에서 cv2.NORMAL_CLONE과 cv2.MIXED_CLONE 플래그의 차이점을 구분할 수 있습니다.

```
1  import cv2
2  import numpy as np
3
4  logo = cv2.imread('logo.png')
5  lena = cv2.imread('lena.jpg')
6  mask = np.full_like(logo, 255)
7  height, width = lena.shape[:2]
8  center = (width//2, height//2)
9
10 normal = cv2.seamlessClone(logo, lena, mask, center,
                              cv2.NORMAL_CLONE)
11 mixed = cv2.seamlessClone(logo, lena, mask, center,
                             cv2.MIXED_CLONE)
12 cv2.imshow('normal', normal)
13 cv2.imshow('mixed', mixed)
```

```
cv2.waitKey()
cv2.destroyAllWindows()
```



NORMAL_CLONE MIXED_CLONE

cv2.seamlessClone() 함수를 사용하면 한 이미지에서 개체를 복사하여 다른 이미지에 붙여넣어 매끄럽고 자연스러운 이미지를 만들 수 있습니다. 만일 다음 두 이미지 red_sky.jpg와 airplane2.jpg를 합친다고 생각해 보세요.

그림 4. red_sky.jpg

epSos.de, CC BY 2.0
〈https://creativecommons.org/licenses/by/2.0〉, via Wikimedia Commons
https://commons.wikimedia.org/wiki/File:Big_Tree_with_Red_Sky_in_the_Winter_Night.jpg

그림 5. airplane2.jpg

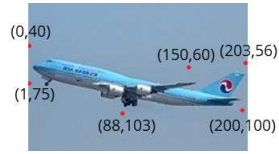
By N509FZ - 자작, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=73763268
https://commons.wikimedia.org/wiki/File:HL7637@HKG_(20181021143558).jpg

이 두 이미지를 하나로 합치면 다음과 같은 이미지를 얻을 수 있습니다. 하늘에 있는 비행기는 누가 보더라도 어색하게 보입니다.

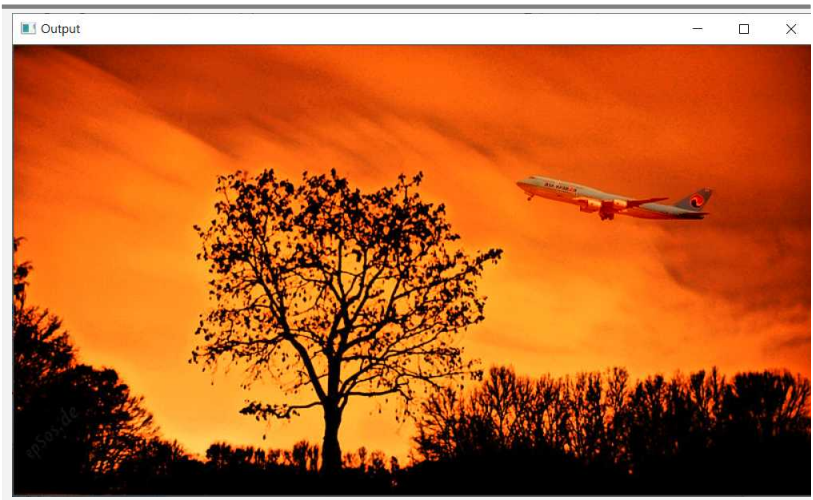
그림 6. 비트 연산을 이용해 합친 이미지

물론 여러분이 포토샵 전문가라면 비행기 이미지(airplane.jpg)를 조심스럽게 마스킹하고 두 이미지가 좀 더 부드럽게 합쳐진 이미지를 얻을 수 있을 것입니다. 그런데 포토샵을 사용하지 않아도 몇 줄의 코드만 있으면 이 두 이미지가 자연스럽게 그리고 하늘빛이 비행기와 어울리도록 만들 수 있습니다. 이것을 가능하게 해 주는 것이 cv2.seamlessClone() 함수입니다.

다음 코드에서 poly 변수는 비행기 이미지에서 비행이 영역을 대충 지정한 다각형입니다. 비행기의 외곽선을 완벽하게 지정하지 않아도 됩니다. center = (600,150)은 비행기(airplane2.jpg)의 중심이 위치 되어야 할 하늘이미지(red_sky.jpg)의 좌표 (x, y)입니다.




```
1  import cv2
2  import numpy as np
3
4  src = cv2.imread("airplane2.jpg")
5  dst = cv2.imread("red_sky.jpg")
6  src_mask = np.zeros(src.shape, src.dtype)
7  poly = np.array([ (0,40), (150,60), (203,56), (200,100),
                     (88,103), (1,75) ], np.int32)
8  cv2.fillPoly(src_mask, [poly], (255, 255, 255))
9  center = (600,150)
10 output = cv2.seamlessClone(src, dst, src_mask, center,
                              cv2.NORMAL_CLONE)
11
12 cv2.imshow('Output', output)
13 cv2.waitKey()
14 cv2.destroyAllWindows()
```



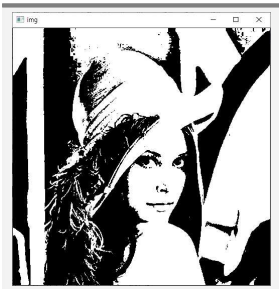
2.3. 이진화

임계값(Threshold)을 이용해서 이미지를 이진화하는 것은 영상처리에서 자주 사용되는 기술입니다.

1) 이진화

배열의 브로드캐스팅 규칙과 True/False가 1과 0으로 계산될 수 있다는 것을 이용해서 이미지 임계처리를 쉽게 할 수 있습니다.

```
1  import cv2
2  import numpy as np
3
4  def two_tone(img, threshold=128):
5      output = (img > threshold) * 255
6      return output.astype(np.uint8)
7
8  img = cv2.imread("lena.jpg", 0)
9  new_img = two_tone(img, threshold=120)
10
11 cv2.imshow('img', new_img)
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```



2) cv2.threshold()

cv2.threshold() 함수를 이용하면 그레이스케일 이미지를 쉽게 이진화할 수 있습니다.

cv2.threshold(src, thresh, maxValue, thresholdType) -> ret, dst

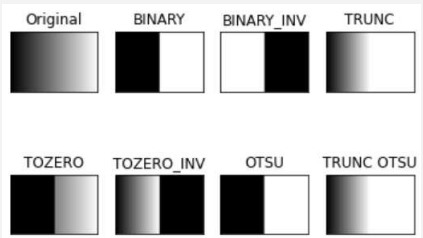
- src – 단일 채널(그레이스케일) 입력 이미지입니다.
- thresh – 임계값입니다.
- maxValue – 임계값을 넘었을 경우 적용할 값입니다.
- thresholdType – 입계값을 적용할 타입입니다.(Thresholding Type)
  사용 가능한 타입 아래와 같습니다.
  cv2.THRESH_BINARY
  cv2.THRESH_BINARY_INV
  cv2.THRESH_TRUNC
  cv2.THRESH_TOZERO
  cv2.THRESH_TOZERO_INV
  cv2.THRESH_OTSU : 1979년 오츠 노부유키(Nobuyuki Otsu)는 임계값을 찾는 알고리즘을 제안했는데 이 알고리즘을 적용하는 것은 thresholdType 매개변수에 cv2.THRESH_OTSU를 추가하면 됩니다. 이때 임계값은 -1로 설정하면 됩니다. 오츠의 알고리즘은 아래 주소를 참고하세요.
  - https://en.wikipedia.org/wiki/Otsu's_method

- 반환값 ret, dst – 이 함수가 반환하는 값은 임계값과 결과이미지입니다.

다음 코드는 임계 타입에 따른 결과를 보여주는 예입니다.

```
1  import cv2
2  from matplotlib import pyplot as plt
3
4  img = cv2.imread('gradation.png',0)
5  _, bin1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
6  _, bin2 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
7  _, bin3 = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC)
8  _, bin4 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO)
9  _, bin5 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV)
10 _, bin6 = cv2.threshold(img, -1, 255, cv2.THRESH_OTSU)
```

```
_, bin7 = cv2.threshold(img, -1, 255,
                        cv2.THRESH_TRUNC | cv2.THRESH_OTSU)
titles = ['Original', 'BINARY', 'BINARY_INV', 'TRUNC',
'TOZERO', 'TOZERO_INV', 'OTSU', 'TRUNC OTSU']
images = [img, bin1, bin2, bin3, bin4, bin5, bin6, bin7]
for i in range(8):
    plt.subplot(2, 4, i+1), plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```



3) 적응 임계처리

임계값을 이미지 전체에 적용하여 처리하기 때문에 하나의 이미지에 음영이 다르면 일부 영역이 모두 흰색 또는 검은색으로 보이게 됩니다. 이런 문제를 해결하기 위해서 이미지의 작은 영역별로 나눠서 하는 임계처리를 적응 임계처리(Adaptive Threshold)라고 합니다. 이때 사용하는 함수가 cv2.adaptiveThreshold() 입니다.

cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C) -> dst

- ▶ src – 단일 채널 입력 이미지(그레이스케일 이미지)입니다.
- ▶ maxValue – 임계값을 넘었을 때 적용할 값입니다.
- ▶ adaptiveMethod – 임계값을 결정하는 계산 방법을 지정합니다.
  cv2.ADAPTIVE_THRESH_MEAN_C : blockSize*blockSize 안에 있

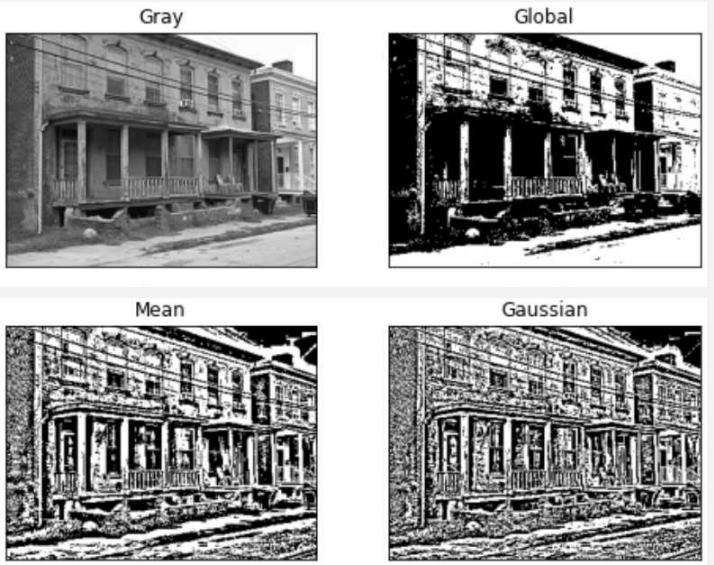
는 화소 값의 평균에서 C값을 뺀 값을 임계치로 합니다. cv2.ADAPTIVE_THRESH_GAUSSIAN_C : blockSize*blockSize 안의 가우시안 윈도우 기반 가중치들의 합에서 C값을 뺀 값을 임계치로 합니다.

- thresholdType – 임계치를 적용할 타입(Thresholding Type)을 지정합니다. cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2.THRESH_TOZERO, cv2.THRESH_TOZERO_IN을 지정할 수 있습니다.
- blockSize – 임계치를 적용할 영역의 크기(가로/세로 화소 수)입니다.
- C – 평균이나 가중평균에서 차감할 값입니다.

- 반환값 dst – 적응 임계처리된 이미지 객체입니다.

다음 코드는 적응 임계처리를 사용한 예입니다.

```
1  import cv2
2  from matplotlib import pyplot as plt
3
4  img_gray = cv2.imread('house-320x240.jpg', 0)
5
6  _, img_bin = cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY)
7  adt_mean = cv2.adaptiveThreshold(img_gray, 255,
                                    cv2.ADAPTIVE_THRESH_MEAN_C,
                                    cv2.THRESH_BINARY, 15, 0)
8  adt_gaus = cv2.adaptiveThreshold(img_gray, 255,
                                cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                cv2.THRESH_BINARY, 15, 0)
9
10 titles = ['Gray', 'Global', 'Mean', 'Gaussian']
11 images = [img_gray, img_bin, adt_mean, adt_gaus]
12
13 plt.figure(figsize=(8,6))
14 for i in range(4):
15     plt.subplot(2,2,i+1), plt.imshow(images[i],'gray')
16     plt.title(titles[i])
17     plt.xticks([])
18     plt.yticks([])
19
20 plt.show()
```



Taber Andrew Bain from Richmond, VA, USA, CC BY 2.0
〈https://creativecommons.org/licenses/by/2.0〉, via Wikimedia Commons
https://commons.wikimedia.org/wiki/File:Gradation_(2476724304).jpg

원본 이미지를 그레이스케일 이미지로 불러온 후 일반적인 임계치 처리를 한 결과(Global)와 적응 임계처리한 결과(Mean, Gaussian) 이미지를 비교해 보면 적응 임계처리가 이미지의 특징을 더 잘 살릴 수 있음을 알 수 있습니다.

적응 임계처리에 사용하는 블록의 크기는 1보다 큰 홀수여야 합니다. 블록의 크기가 짝수이면 현재 주목 화소의 왼쪽과 오른쪽, 그리고 위와 아래쪽의 화소수가 비례하지 않으므로 짝수는 사용할 수 없습니다. 예제에 사용한 이미지는 320x240 크기의 이미지입니다. 블록의 크기에 따라 임계처리의 결과가 달라질 수 있습니다. 이미지의 크기에 비해 블록의 크기가 작으면 더 섬세한 임계처리가 됩니다.

4) 디더링

디더링(Dithering)은 제한된 색을 이용하여 음영이나 색을 나타내는 기법입니다. 디더링을 이용하면 그레이스케일 영상을 검정색 또는 흰색 점의 밀조 조합을 이용해서 2진화할 수 있습니다.

다음 그림의 오른쪽은 그레이스케일 영상처럼 보이지만 확대해 보면 왼쪽의 그림처럼 검은점과 흰점의 밀도로 표현되는 것을 알 수 있습니다.

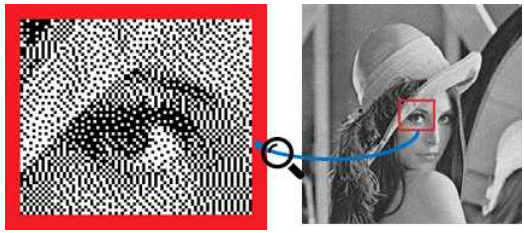


그림 7. 디더링

영상 편집 소프트웨어에서 널리 쓰이는 알고리즘으로 플로이드-스테인버그(Floyd-Steinberg dithering)[4] 디더링, 평균 디더링(Average dithering), 순서 디더링(Ordered dithering), 임의 디더링(Random dithering) 등이 있습니다.

이 책은 플로이드-스테이버그 디더링에 대해서만 설명합니다. 다른 디더링 방법들에 대해 알고 싶다면 다음 주소를 참고하세요. 아래 주소에서 플로이드-스테인버그 디더링에 대한 더 자세한 정보도 볼 수 있습니다.

- https://en.wikipedia.org/wiki/Dither

4) 1976년 로버트 플로이드와 루이스 스타인버그가 고안한 영상 디더링 알고리즘입니다. https://en.wikipedia.org/wiki/Floyd-Steinberg_dithering

플로이드-스테인버그 알고리즘은 화소의 양자화 오류를 주위의 화소로 분산시킴으로써 디더링을 수행하며, 그 분포는 다음과 같습니다.

$$\begin{bmatrix} & & * & \frac{7}{16} & \cdots \\ \cdots & \frac{3}{16} & \frac{5}{16} & \frac{1}{16} & \cdots \end{bmatrix}$$

앞의 디더링 행렬에서 *는 현재 주목 화소를 나타내고 숫자(가중치)는 해당 위치의 화소에 분포된 오류의 비율을 나타냅니다. 여기에서 바로 오른쪽에 있는 화소는 오류의 7/16을 얻습니다. 바로 아래에 있는 화소는 오류의 5/16를 얻으며. 대각선으로 인접한 화소는 왼쪽이 3/16을 얻고 오른쪽이 1/16을 얻습니다.

알고리즘은 영상을 왼쪽에서 오른쪽, 위에서 아래로 따라가면서 각각의 화소를 양자화합니다. 양자화 오류(Quant Error, 소스코드에서는 error)는 주위의 화소로 이전되지만, 이미 양자화된 화소는 영향을 받지 않습니다. 또한, 화소 값이 반올림한 결과 내림이 된 화소가 많을수록 다음 화소는 올림이 될 가능성이 커지며, 이로써 평균적인 양자화 오류는 0에 가까워집니다.

다음 코드는 그레이스케일 영상을 디더링 기법을 이용하여 2진 영상으로 만드는 dithering() 함수와 minmax() 함수입니다. ditering() 함수에서 사용하는 minmax() 함수는 계산된 화소의 값이 0보다 작으면 0, 255보다 크면 255가 되도록 합니다.

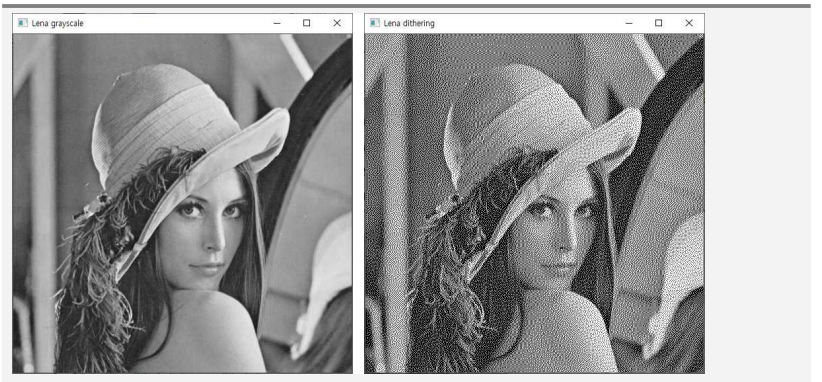
```
1  def minmax(pixel):
2      if pixel > 255:
3          pixel = 255
4      if pixel < 0:
5          pixel = 0
6      return pixel
```

다음 dithering() 함수는 플로이드-스테인버그 알고리즘을 이용해서 그레이스케일 이미지를 0과 255화소만 갖는 디더링 이미지로 변환합니다. 모든 화소에 대해 현재 주목 화소의 오른쪽, 왼쪽 아래, 아래 그리고 오른쪽 아래 화소에 대해 디더링된 현재 화소와의 차이에 가중치를 부여합니다. 이렇게 가중치가 부여된 화소의 값은 255로 나눠 반올림한 값을 이용해서 0 또는 255화소 값으로 변환합니다.

```
1  import numpy as np
2
3  def dithering(img):
4      height, width = img.shape
5      for y in range(0, height-1):
6          for x in range(1, width-1):
7              p = img[y, x]
8              new_p = np.round(p/255) * 255
9              img[y, x] = new_p
10             error = p - new_p
11             img[y  , x+1] = minmax(img[y  , x+1] + error*7/16)
12             img[y+1, x-1] = minmax(img[y+1, x-1] + error*3/16)
13             img[y+1, x  ] = minmax(img[y+1, x  ] + error*5/16)
14             img[y+1, x+1] = minmax(img[y+1, x+1] + error*1/16)
15     return img
```

다음 코드는 위 함수를 실행시켜 레나 영상을 디더링 기법을 이용해 2진화합니다. 두 결과 이미지가 비슷해 보이지만 분명 차이가 있습니다.

```
1  lena = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
2  lena_dithering = dithering(lena.copy())
3  cv2.imshow('Lena grayscale', lena)
4  cv2.imshow('Lena dithering', lena_dithering)
5  cv2.waitKey()
6  cv2.destroyAllWindows()
```



출력 결과에서 왼쪽은 그레이스케일 이미지이고, 오른쪽은 디더링한 이미지입니다.

Instructor Note : 다음 코드를 넘파이 함수를 이용하여 바꿀 수 있나요?

```
_, bin_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
```

위 코드는 img에 스레스홀드 127을 적용하여 이진화(반전)합니다. 이것은 넘파이의 함수와 연산을 이용한다면 다음처럼 바꿀 수 있습니다.

```
bin_img = ((img < 127) * 255).astype(np.uint8)
```

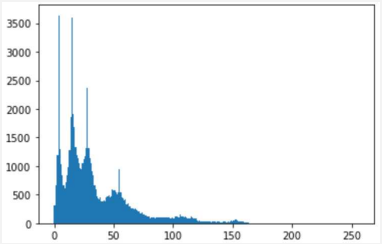
3절. 히스토그램

3.1. 이미지 히스토그램

이미지 히스토그램은 가로축에 화소의 값을, 세로축에 화소의 수를 이용해 막대그래프로 표현한 것입니다. 화소 값의 범위는 0~255이지만 실제 화소들이 분포된 범위는 다를 수 있습니다. 히스토그램을 이용하면 화소들이 어떤 값에 더 많이 분포되어 있는지 알 수 있습니다.

다음 코드는 히스토그램을 계산한 후 맷플롯립(Matplotlib)의 막대그래프(bar)로 표현하는 예입니다. 코드에서 사용한 이미지는 전체적으로 어두운 이미지입니다. 히스토그램에서 0~100 사이의 화소들이 많이 있는 것을 볼 수 있습니다.

```
 1  import cv2
 2  img = cv2.imread("couple256.jpg", flags=cv2.IMREAD_GRAYSCALE)
 3
 4  import numpy as np
 5  hist = np.zeros((256))
 6
 7  height, width = img.shape
 8
 9  for y in range(height):
10      for x in range(width):
11          hist[img[y,x]] = hist[img[y,x]] + 1
12
13  import matplotlib.pyplot as plt
14  plt.bar(x=range(256), height=hist, width=1)
15  plt.show()
```



3.2. 히스토그램 함수 구현

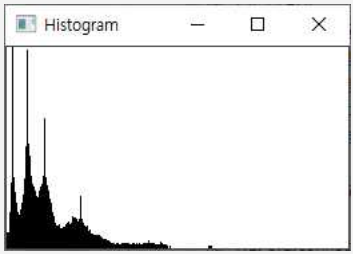
다음 함수는 그레이스케일 이미지를 입력받아 히스토그램을 계산하여 반환하는 함수입니다.

```
import numpy as np

def histogram(img):
    height, width = img.shape
    hist_ = np.zeros(256)

    for y in range(height):
        for x in range(width):
            hist_[img[y,x]] = hist_[img[y,x]] + 1

    return hist_
```

다음 함수는 히스토그램 배열을 입력받아 히스토그램을 이미지 객체로 만들어 반환하는 함수입니다. 맷플롯립을 사용하지 않고 윈도우를 이용해서 히스토그램을 표시해 줄 수 있습니다. 히스토그램을 이미지의 크기는 150x256입니다. 이미지의 높이를 바꾸고 싶으면 코드에서 hist_height 변수의 값을 바꾸면 됩니다.

```python
import numpy as np

def histogram_img(hist):
    hist_height = 150
    hist_img = np.full((hist_height,256), 255, dtype=np.uint8)
    hist = hist.astype(int)
    hist_max = max(hist)

    for i, value in enumerate(hist):
        height = -int(value * (hist_height/hist_max))
        if height<-1:
            hist_img[height:,i] = 0

    return hist_img
```

```python
import cv2

img = cv2.imread("couple256.jpg", flags=cv2.IMREAD_GRAYSCALE)
hist = histogram(img)
hist_img = histogram_img(hist)

cv2.imshow('Histogram', hist_img)
cv2.waitKey()
cv2.destroyAllWindows()
```



3.3. cv2.calcHist()

cv2.calcHist() 함수를 이용하여 히스토그램을 계산할 수 있습니다.

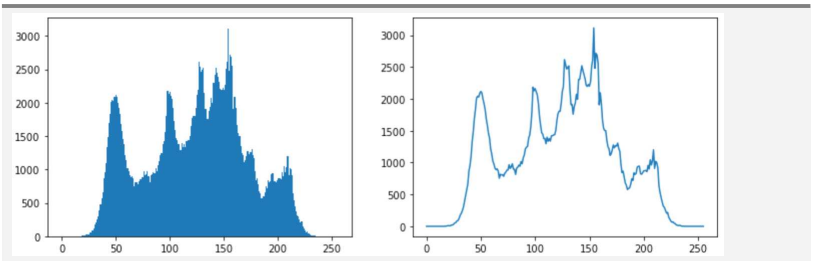
cv2.calcHist(images, channels, mask, histSize, ranges) -> histo

- image – 분석대상 이미지입니다.
- channels – 분석 채널(X축의 대상)입니다. 이미지가 그레이스케일이면 [0], 컬러이면 [0],[1] 형태(0 : Blue, 1: Green, 2: Red)입니다.
- mask – 이미지의 분석영역입니다. None이면 전체 영역을 의미합니다.
- histSize – BINS(히스토그램을 그릴 구간의 수)입니다. 히스토그램을 그리려면 [256]을 사용합니다.
- ranges – 히스토그램을 그릴 범위입니다. [0,256]을 지정합니다.

- 반환값 histo – 계산된 히스토그램입니다.

다음 코드는 앞에서 정의한 histogram() 함수(실행결과 왼쪽)와 cv2.calcHist() 함수(실행결과 오른쪽)를 이용해서 히스토그램을 계산하고 선 그래프로 표시한 예입니다.

```
1  import cv2
2  import matplotlib.pyplot as plt
3
4  lena_gray = cv2.imread("lena.jpg", 0)
5
6  plt.figure(figsize=(12,4))
7  plt.subplot(121)
8  plt.bar(x=range(256), height=histogram(lena_gray), width=1)
9
10 plt.subplot(122)
11 hist = cv2.calcHist(images=[lena_gray], channels=[0],
                       mask=None, histSize=[256], ranges=[0, 256])
12 plt.plot(hist.flatten())
13 plt.show()
```



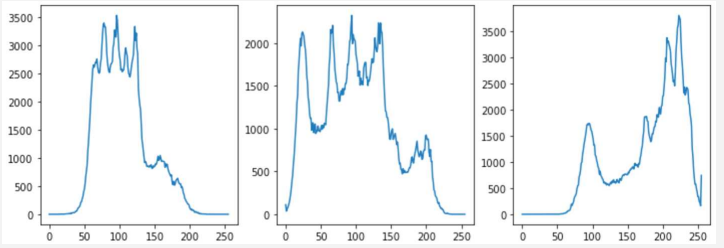
3.4. 채널

영상처리에서 채널(channels)은 한 개 화소가 가지고 있는 정보를 의미합니다. 그레이스케일 이미지의 경우 채널이 1개이고, RGB 정보를 가지고 있는 컬러 이미지의 경우 채널이 3개입니다. cv2.calcHist() 함수는 채널별로 히스토그램을 계산할 수 있습니다. 컬러 영상의 경우 채널이 0, 1, 2이며, 0이면 Blue, 1이면 Green, 2이면 Red입니다.

다음 코드는 레나 컬러 영상의 채널별로 히스토그램을 그립니다. 결과 히스토그램은 Blue, Green, Red 순입니다.

```
lena = cv2.imread("lena.jpg")

plt.figure(figsize=(12,4))
plt.subplot(131)
hist1 = cv2.calcHist(images=[lena], channels=[0], mask=None,
                     histSize=[256], ranges=[0,256])
plt.plot(hist1.flatten())

plt.subplot(132)
hist1 = cv2.calcHist(images=[lena], channels=[1], mask=None,
                     histSize=[256], ranges=[0,256])
plt.plot(hist1.flatten())

```

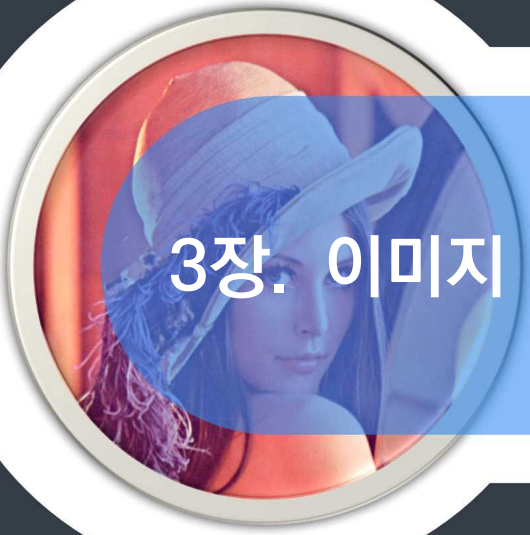
```
12  plt.subplot(133)
13  hist1 = cv2.calcHist(images=[lena], channels=[2], mask=None,
                         histSize=[256], ranges=[0,256])
14  plt.plot(hist1.flatten())
15
16  plt.show()
```



앞의 코드는 다음과 같이 반복문을 이용해서 작성할 수 있습니다.

```
1  lena = cv2.imread("lena.jpg")
2  plt.figure(figsize=(12,4))
3
4  for i in range(3):
5      plt.subplot(1,3,i+1)
6      hist = cv2.calcHist(images=[lena],
                           channels=[i], mask=None,
                           histSize=[256], ranges=[0,256])
7      plt.plot(hist.flatten())
8
9  plt.show()
```

* 결과는 반복문을 사용하지 않은 코드와 같습니다.



3장. 이미지 변환

1절. 기하 변환

1.1. 변환

1) 변환의 종류

이미지를 다른 형태로 바꾸는 것을 변환(Transformation)이라 합니다. 변환의 종류에는 몇 가지 분류가 있습니다.

- 유사(Similarity) 변환 : 물체의 크기는 변하지만, 각도는 보존되는 변환을 의미합니다. 유사변환의 한 종류로 크기변환(Scaling)이 있습니다.
- 강체(Rigid-Body) 변환 : 크기 및 각도가 보존되는 변환입니다. 이동(Translation)과 회전(Rotation)이 있습니다.
- 아핀(Affine) 변환 : 아핀 변환은 선형 변환과 이동 변환을 포함하는 변환입니다. 선의 수평성은 유지합니다. 사각형을 평행사변형 모형으로 변환하는 것이 있습니다.
- 원근법(Perspective) 변환 : 원근법 변환은 수평성도 유지되지 않는 아핀 변환에 해당합니다. 가까이 있는 물체는 크게 보이고, 멀리 있는 물체는 작게 보이도록 하는 변환입니다.
- 푸리에((Fourier) 변환 : 주파수를 이용한 변환입니다.

이 장에서는 크기(Scaling) 변환, 이동(Translation), 회전(Rotation), 아핀(Affine) 변환, 그리고 원근법(Perspective) 변환, 그리고 주파수를 이용한 푸리에 변환을 설명합니다.

2) 기준 좌표 설정 방법

영상의 좌표를 표현하는 방법은 화상의 좌측 위를 원점으로 한 경우와 화상의 중앙을 원점으로 하는 경우가 있습니다.

이동 변환의 경우 어디를 원점으로 두어도 크게 문제가 될 것은 없습니다. 그러나 회전 등 일부 변환의 경우에는 원점이 어디냐에 다라 전혀 다른 결과가 될 수 있습니다.

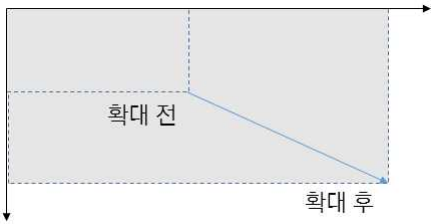


그림 1. 화상의 좌측 위를 원점으로 한 경우

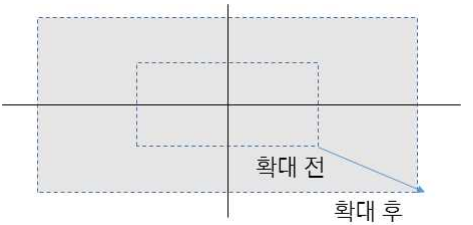


그림 2. 화상의 중앙을 원점으로 한 경우

원점을 왼쪽상단으로 했을 경우 기준점 (x, y)의 위치는 (0, 0)이 되지만 이미지의 가운데로 했을 경우 기준점 (x, y)의 위치는 (width/2, height/2)가 됩니다.

2절. 크기변환

2.1. 크기변환

크기변환은 말 그대로 이미지의 크기를 변환하는 것을 의미합니다. 이미지의 크기변환 시 입력 화상의 좌표는 출력 화상의 새로운 좌표에 매핑됩니다. 만일 어떤 점 (x, y)가 x 방향으로 a배, y 방향으로 b배 확대 또는 축소되어 (X, Y)로 위치를 바꾼다고 하면 변환공식은 다음과 같습니다.

$$X = a*x, \qquad Y = b*y$$

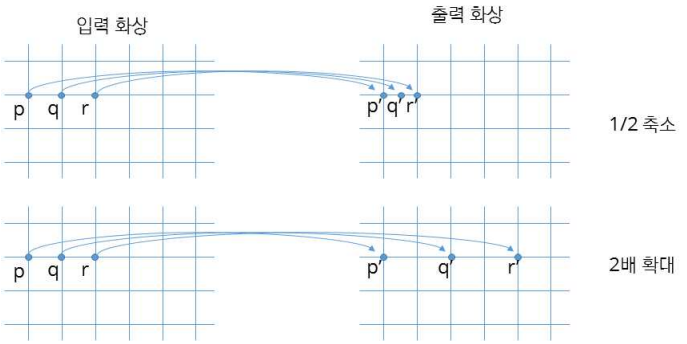


그림 3. 크기 변경

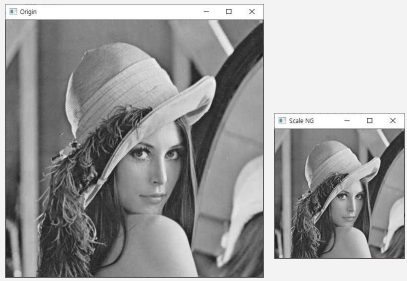
크기 변경을 위한 반복문은 입력 영상을 기준으로 실행시키는 방법과 출력 영상(크기 변경 후 영상)을 기준으로 실행시키는 방법이 있습니다.

영상을 (scale_x, scale_y)배 확대하려면 입력 영상의 (x, y) 위치 화소 값은 출력 영상의 (x*scale_x, y*scale_y) 화소에 매핑시켜야 합니다. 다음 코드는 입력 영상을 기준으로 반복문을 실행시킵니다.

```python
import numpy as np

def scale_nogood(img, scale_x=1, scale_y=1):
    height, width = img.shape
    height_n, width_n = int(height*scale_y), int(width*scale_x)
    img_ = np.zeros((height_n, width_n), dtype=np.uint8)

    for y in range(height):
        for x in range(width):
            img_[int(y*scale_y), int(x*scale_x)] = img[y,x]

    return img_
```

다음 코드는 가로의 크기는 50%, 세로의 크기는 50%로 크기 변경한 이미지를 출력합니다.

```python
import cv2

img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)

result = scale_nogood(img, 0.5, 0.5)

cv2.imshow('Origin', img)
cv2.imshow('Scale NG', result)
cv2.waitKey()
cv2.destroyAllWindows()
```



이미지의 크기를 줄이는 경우 큰 문제가 없어 보입니다. 그런데 이미지를 원본 이미지보다 크게 하는 경우는 문제가 있습니다. 이미지의 크기변환 시 이미지 크기를 크게 하는 경우 다음처럼 화소가 매핑되지 않는 화소 영역이 발생할 수 있습니다.

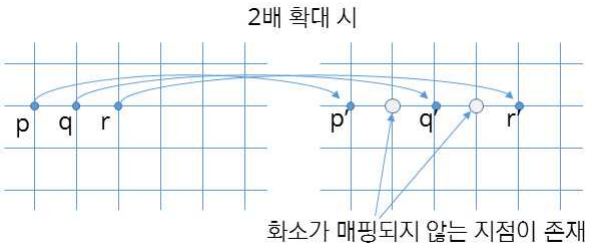
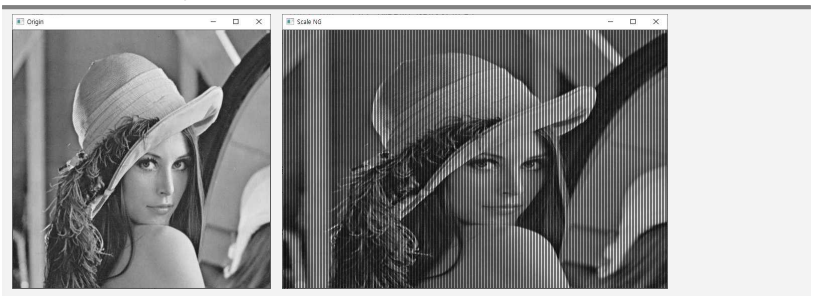


그림 4. 이미지 확대 시 문제점

다음 코드는 가로의 크기를 120%로 원본 이미지보다 조금 더 크게 변경합니다. 결과 이미지에 세로 검은색 선이 생기는 이유는 확대 후 출력 영상에 매핑되는 입력 영상의 위치를 찾을 수 없기 때문입니다.

```
1  import cv2
2
3  cv2.imshow('Origin', img)
4  cv2.imshow('Scale NG', scale_nogood(img, 1.5, 1.0))
5  cv2.waitKey()
6  cv2.destroyAllWindows()
```



2.2. 출력 영상을 기준으로 처리

출력 영상을 기준으로 반복문을 실행시키려면 (x, y) 위치의 화소를 (scale_x, scale_y)배 확대할 경우 다음처럼 역변환 공식이 필요합니다.

$$x = X/a, \quad y = Y/b$$

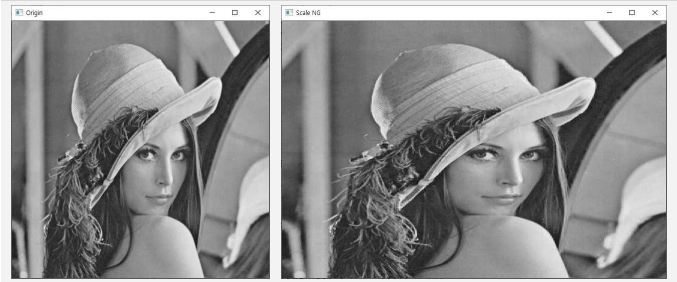
출력 영상을 기준으로 처리하면 출력 영상의 모든 화소에 매핑되는 화소를 찾을 수 있습니다. 다음 코드는 출력 영상을 기준으로 크기변환 코드를 실행시킵니다.

```
import numpy as np

def scale_nogood2(img, scale_x=1, scale_y=1):
    height, width = img.shape
    height_n, width_n = int(height*scale_y), int(width*scale_x)
    img_ = np.zeros((height_n, width_n), dtype=np.uint8)
    for y in range(height_n):
        for x in range(width_n):
            img_[y,x] = img[int(y/scale_y), int(x/scale_x)]

    return img_
```

영상을 (scale_x, scale_y)배 확대할 경우 출력 영상의 (X, Y) 위치의 화소는 입력 영상에서 (x/scale_x, y/scale_y) 위치의 화소값을 가집니다. 다만 x/scale_x, y/scale_y의 결과가 정수가 아닐 경우 매핑되는 지점을 찾을 수 없으므로 int 자료형으로 형변환 해줘야 합니다.

다음은 함수 실행을 확인하기 위한 코드입니다.

```
import cv2

img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)

result2 = scale_nogood2(img, 1.5, 1.0)

cv2.imshow('Origin', img)
cv2.imshow('Scale NG', result2)
cv2.waitKey()
cv2.destroyAllWindows()
```



이 방법이 매핑되지 않는 화소의 값을 대체하는 가장 좋은 방법은 아닙니다. 크기변환을 시도할 경우 출력 영상의 좌표와 매핑될 입력 영상의 좌표를 찾는 방법 또는 화소의 값을 결정하는 방법들이 있습니다. 이것을 보간법이라고 합니다.

보간법은 변환 후 이미지의 화소가 원본 이미지의 화소를 찾지 못할 경우 수행하는 방법입니다. 그래서 보간법은 크기변환에만 사용하는 것이 아닙니다. 이미지의 이동, 회전 등 다른 변환에도 보간법이 사용됩니다.

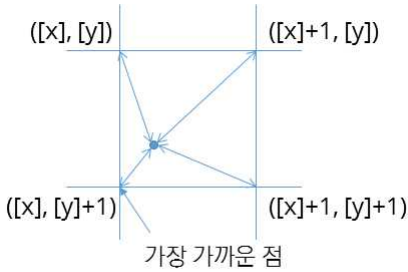
다음은 보간법에 대하여 설명합니다,

2.3. 보간법

변환 후 이미지는 원본 이미지의 화소를 가져와야 하는데 원본 이미지에서 가져와야 할 화소의 좌표가 정수가 아닌 실수로 계산되는 경우 변환 후 화소의 값을 지정할 방법이 필요합니다. 이를 보간법이라 하며, 보간법의 종류에는 최근방법, 선형방법 등이 있습니다.

1) 최근방법

최근방법(Nearest)은 가장 가까운 좌표의 값을 이용해서 지정하는 방법입니다. 예를 들면 변환 후 화소의 값이 원본 화소의 (5.3, 7.8) 위치에 해당하는 값이어야 한다면 이 값을 반올림한 (5, 8) 위치의 화소 값을 변환 후 화소 값으로 사용합니다.



[x], [y]는 각각 x, y를 넘지 않는 정수

그림 5. 최근방법

다음 코드는 최근방법을 구현한 예입니다.

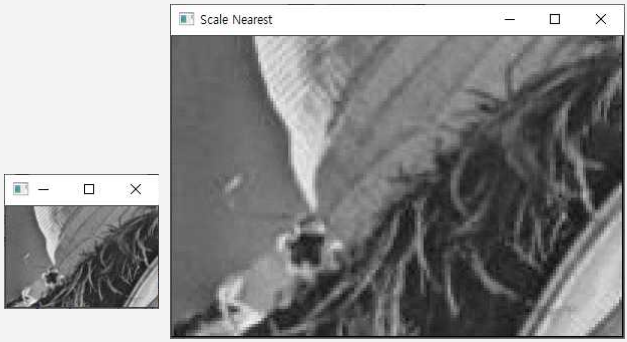
```
1  import numpy as np
2
3  def scale_nearest(img, scale_x=1, scale_y=1):
4      height, width = img.shape
```

```
    img_ = np.zeros((int(height*scale_y), int(width*scale_x)),
                    dtype=np.uint8)
    for y in range(int(height*scale_y)):
        for x in range(int(width*scale_x)):
            try:
                img_[y,x]=img[round(y/scale_y), round(x/scale_x)]
            except:
                pass

    return img_
```

최근방법의 경우 이미지를 확대하는 비율이 클수록 계단현상이 더 많이 보이는 단점이 있습니다.

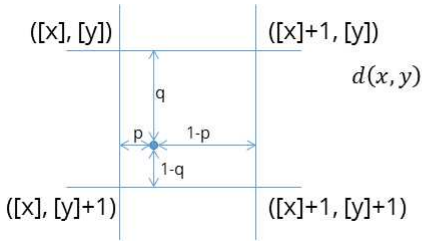
다음 코드는 레나 영상의 일부분을 가로/세로 각 3배씩 확대하기 전(왼쪽)과 후(오른쪽)의 이미지를 보여줍니다.

```
import cv2

lena_roi = cv2.imread("lena_roi.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow('Lena ROI', lena_roi)
cv2.imshow('Scale Nearest', scale_nearest(lena_roi, 3, 3))
cv2.waitKey()
cv2.destroyAllWindows()
```



2) 선형방법

최근 방법의 단점을 극복하는 방법으로 선형방법(Linear)이 있습니다. 선형방법은 원본 영상에서 인접한 네 개의 화소와 소수점 이하를 가중치로 사용하여 실수 좌표상의 화소값을 계산하는 방법입니다.



$$d(x,y) = (1-p)*(1-q)*d([x],[y]) + p*(1-q)*d([x]+1,[y]) + (1-p)*q*d([x],[y]+1) + p*q*d([x]+1,[y]+1)$$

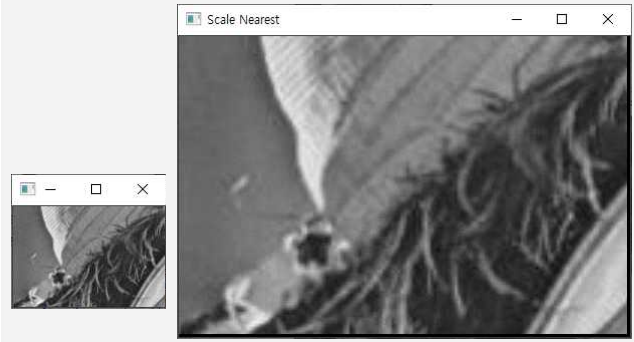
그림 6. 선형방법

다음 코드는 선형방법을 이용한 이미지 크기를 바꿉니다.

```
import numpy as np

def scale_bilinear(img, scale_x=1, scale_y=1):
    height, width = img.shape
    img_ = np.zeros((int(height*scale_y), int(width*scale_x)),
                    dtype=np.uint8)
    for y in range(int(height*scale_y)):
        for x in range(int(width*scale_x)):
            q = x/scale_x-int(x/scale_x)
            p = y/scale_y-int(y/scale_y)
            try:
                X = int(x/scale_x)
                Y = int(y/scale_y)
                value = (1-p)*(1-q)*img[Y,X] + p*(1-q)*img[Y+1,X] +
(1-p)*q*img[Y,X+1] + p*q*img[Y+1,X+1]
                if value > 255:
                    img_[y, x] = 255
                else:
```

```
                    img_[y, x] = int(value)
            except:
                pass
    return img_
```

```
import cv2

lena_roi = cv2.imread("lena_roi.jpg", cv2.IMREAD_GRAYSCALE)

cv2.imshow('Lena ROI', lena_roi)
cv2.imshow('Scale Nearest', scale_bilinear(lena_roi, 3, 3))
cv2.waitKey()
cv2.destroyAllWindows()
```



위 이미지에서 오른쪽 이미지는 선형방법으로 3배 확대한 이미지입니다. 최근방법에 비해 경계선 부분의 계단현상이 줄어든 것을 확인할 수 있습니다.

- 책의 인쇄 품질에 따라 최근방법과 선형방법의 차이를 느끼지 못할 수 있습니다. 코드를 직접 작성하고 실행하여 그 결과의 차이점을 확인해 보시길 바랍니다.

2.4. Max Pooling

이미지의 크기를 줄이는 과정에서 줄여야 할 대상 영역의 화소 중에서 어떤 화소의 값을 줄여지는 이미지의 화소로 선택할지 결정해야 합니다.

풀링(Pooling) 방법으로는 다음 3가지를 주로 사용합니다.
- Max Pooling : 대상 화소 중에서 가장 큰 화소 값을 선택
- Mean Pooling(Average Pooling) : 대상 화소의 평균값을 선택
- Min Pooling : 가장 작은 값을 선택

Max Pooling의 경우 아래와 같이 영상의 크기를 1/2로 줄여야 하면 2x2화소들 중에서 가장 큰 화소의 값을 대상 화소의 값으로 선택하는 것을 의미합니다.

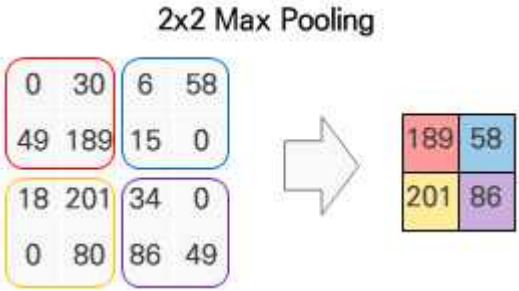


그림 7. Max Pooling

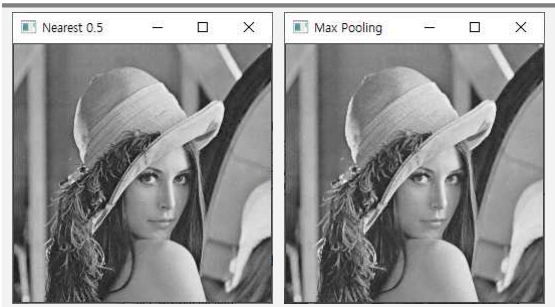
맥스 풀링은 인공신경망 CNN을 배울 때 나오는 내용입니다. 영상을 합성곱(Convolution)한 후 크기를 줄이기 위해 사용하는 것이 풀링인데, 특징이 더 강조된 영상을 얻기 위해 해당 화소 중에서 가장 큰 화소의 값만 빼내는 맥스풀링을 가장 많이 사용합니다. 합성곱은 4장에서 설명합니다.

다음 코드는 2x2화소에서 가장 큰 값을 추출하는 Max Pooling의 예입니다.

```
import numpy as np

def maxpool2d(img):
    height, width = img.shape
    img_ = np.zeros((int(height/2),int(width/2)), dtype=np.uint8)
    for y in range(int(height/2)):
        for x in range(int(width/2)):
            try :
                img_[y,x] = np.max(img[2*y:2*y+2, 2*x:2*x+2])
            except:
                pass
    return img_
```

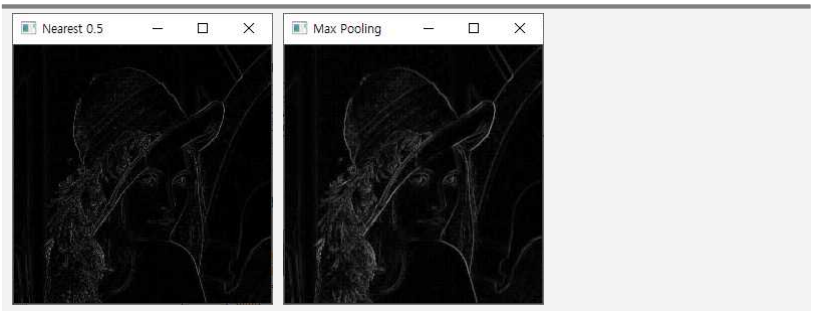
Max Pooling은 다음 코드의 결과처럼 Gray Scale 영상의 경우 전체적으로 이미지가 밝아지는 현상이 있습니다.

```
import cv2

img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow('Nearest 0.5', scale_nearest(img, 0.5, 0.5))
cv2.imshow('Max Pooling', maxpool2d(img))
cv2.waitKey()
cv2.destroyAllWindows()
```



엣지(Edge, 영상의 경계선 또는 윤곽선) 영상 등 특징을 추출해 저장한 이미지의 경우에는 엣지가 더 뚜렷해지는 효과가 있습니다.

```
1  import cv2
2
3  img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
4  cv2.imshow('Nearest 0.5', scale_nearest(img, 0.5, 0.5))
5  cv2.imshow('Max Pooling', maxpool2d(img))
6  cv2.waitKey()
7  cv2.destroyAllWindows()
```



2.5. cv2.resize()

OpenCV의 resize() 함수를 이용하면 쉽게 이미지의 크기를 변경할 수 있습니다.

cv2.resize(img, dsize, fx, fy, interpolation)

- img – 크기를 변경할 원본 이미지 객체입니다.
- dsize – 변경할 크기를 가로, 세로 형태의 튜플(tuple)로 지정합니다. (예: dsize=(100,200))
- fx – 가로 크기의 배수입니다. 2배로 크게 하려면 fx=2. 반으로 줄이려면 fx=0.5입니다.
- fy – 세로 크기의 배수입니다.
- interpolation – 보간법을 지정합니다.

함수의 interpolation 매개변수는 보간법을 지정합니다.

- INTER_NEAREST : 가장 가까운 이웃 보간법
- INTER_LINEAR : 쌍 선형 보간법(기본적으로 사용됨)
- INTER_LINEAR_EXACT : 비트 쌍 선형 보간법
- INTER_AREA : 영역 보간법(이미지를 줄일 때 주로 사용, 이미지를 확대하면 INTER_NEAREST 방법과 유사)
- INTER_CUBIC : 4x4픽셀 이웃에 대한 쌍 입방 보간법
- INTER_LANCZOS4 : 8x8픽셀 이웃에 대한 란죠스(Lanczos)[5] 보간법

이미지의 크기를 줄일 때는 INTER_AREA(영역 보간법)를 주로 사용하며, 이미지의 크기를 크게 할 때는 INTER_LINEAR(쌍 선형 보간법) 또는 INTER_CUBIC(바이큐빅 보간법)을 가장 많이 사용합니다.

이 책은 바이큐빅 보간법이나 란죠스 보간법에 대한 내용은 설명하지 않습니다. 자세한 내용은 아래의 문서를 참고하세요.

- https://en.wikipedia.org/wiki/Bicubic_interpolation
- https://en.wikipedia.org/wiki/Lanczos_resampling

다음 그림들은 5x5 크기의 이미지를 100x100 크기로 이미지를 크게 했을 경우의 각 보간법의 결과를 보여줍니다.

(실제 이미지의 크기는 5x5화소이므로 매우 작습니다. 그래서 이해를 돕기 위해 이미지를 확대했습니다.)

5) Cornelius (Cornel) Lanczos(코넬리우스 란죠스['laːnt͡soʃ]) : 헝가리 수학자이며 물리학자, 1893.2.2.~1974.6.25

그림 8. INTER_NEAREST

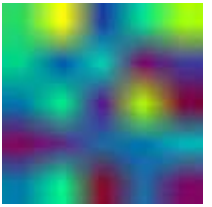


그림 9. INTER_LINEAR

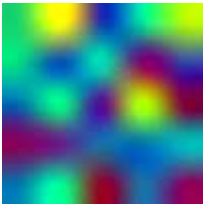


그림 10. INTER_CUBIC

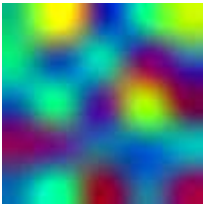


그림 11. INTER_LANCZOS4

다음 코드는 원본 이미지의 화소 값을 출력합니다.

```
1  img = cv2.imread("resize_origin_5.png")
2  img
```

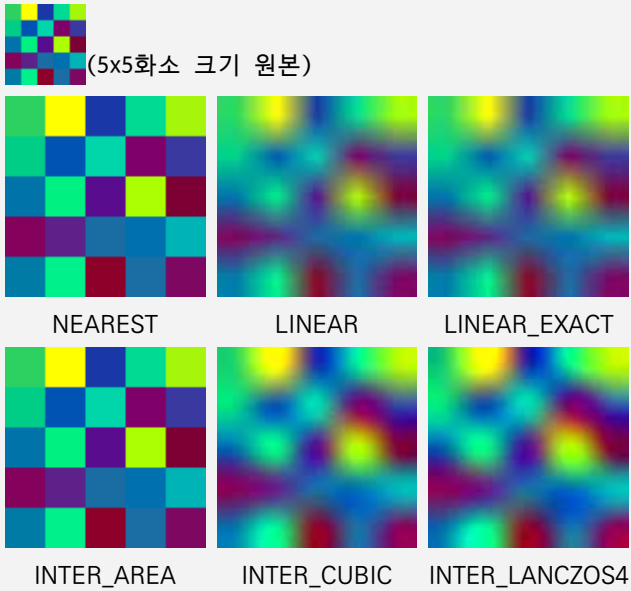
```
array([[[ 98, 209,  44],
        [  0, 255, 255],
        [166,  55,  25],
        [148, 219,   0],
        [ 10, 246, 165]],

       [[134, 206,   0],
        [179,  82,   0],
        [169, 214,   0],
        [105,   0, 128],
        [160,  57,  58]],

       [[169, 115,   0],
```

```
        [129, 240,   0],
        [143,  12,  90],
        [  0, 255, 171],
        [ 53,   0, 125]],

       [[ 92,   2, 132],
        [137,  33,  94],
        [162, 109,  25],
        [174, 112,   0],
        [181, 180,   0]],

       [[166, 123,   0],
        [140, 240,   0],
        [ 41,   0, 141],
        [164, 110,  26],
        [ 97,   7, 125]]], dtype=uint8)
```

다음 코드는 5x5 이미지를 100x100화소 크기로 확대한 결과를 저장합니다.

```
1  import cv2
2
3  img = cv2.imread("resize_origin_5.png")
4  cv2.imwrite("resize-5to100_nearest.png",
               cv2.resize(img, dsize=(100, 100),
                          interpolation=cv2.INTER_NEAREST))
5  cv2.imwrite("resize-5to100_linear.png",
               cv2.resize(img, dsize=(100, 100),
                          interpolation=cv2.INTER_LINEAR))
6  cv2.imwrite("resize-5to100_linear_exact.png",
               cv2.resize(img, dsize=(100, 100),
                          interpolation=cv2.INTER_LINEAR_EXACT))
7  cv2.imwrite("resize-5to100_area.png",
               cv2.resize(img, dsize=(100, 100),
                          interpolation=cv2.INTER_AREA))
```

```
8  cv2.imwrite("resize-5to100_cubic.png",
                cv2.resize(img, dsize=(100, 100),
                           interpolation=cv2.INTER_CUBIC))
9  cv2.imwrite("resize-5to100_lanczos4.png",
                cv2.resize(img, dsize=(100, 100),
                           interpolation=cv2.INTER_LANCZOS4))
```

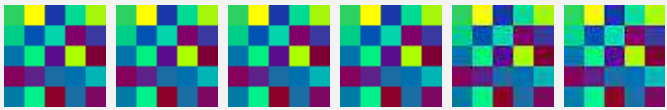


다음 코드는 100x100 이미지를 50x50화소 크기로 줄인 결과를 저장합니다.

```
1  import cv2
2
3  img = cv2.imread("resize_origin_100.png")
4
5  cv2.imwrite("resize-100to50_nearest.png",
                cv2.resize(img, dsize=(50, 50),
                           interpolation=cv2.INTER_NEAREST))
6  cv2.imwrite("resize-100to50_linear.png",
```

```
                cv2.resize(img, dsize=(50, 50),
                           interpolation=cv2.INTER_LINEAR))
cv2.imwrite("resize-100to50_linear_exact.png",
                cv2.resize(img, dsize=(50, 50),
                           interpolation=cv2.INTER_LINEAR_EXACT))
cv2.imwrite("resize-100to50_area.png",
                cv2.resize(img, dsize=(50, 50),
                           interpolation=cv2.INTER_AREA))
cv2.imwrite("resize-100to50_cubic.png",
                cv2.resize(img, dsize=(50, 50),
                           interpolation=cv2.INTER_CUBIC))
cv2.imwrite("resize-100to50_lanczos4.png",
                cv2.resize(img, dsize=(50, 50),
                           interpolation=cv2.INTER_LANCZOS4))
```

(100x100화소 크기 원본)


(실행 결과)

왼쪽부터 NEAREST, LINEAR, LINEAR_EXACT, AREA, CUBIC, LANCZOS4 순서

3절. 이동

이동(Translation)은 이미지의 위치를 변경하는 변환입니다. 이미지의 위치를 변환하기 위해서 변환될 좌표를 원본 좌표에 이동량을 더한 좌표의 화소로 매핑시키면 됩니다.

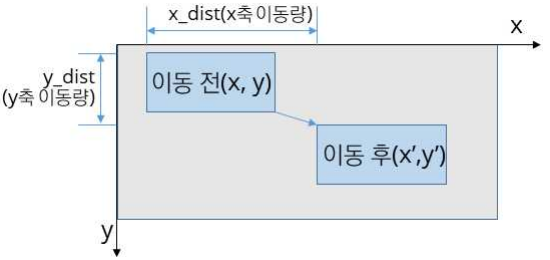


그림 13. 이동

이동 전 위치 (x, y)를 이용해 x축으로 x_dist 만큼 오른쪽으로 이동시키고, y_dist 만큼 아래쪽으로 이동시킨 위치 (x', y')를 계산하려면 다음 수식을 이용합니다.

$$x' = x + x_{dist}$$

$$y' = y + y_{dist}$$

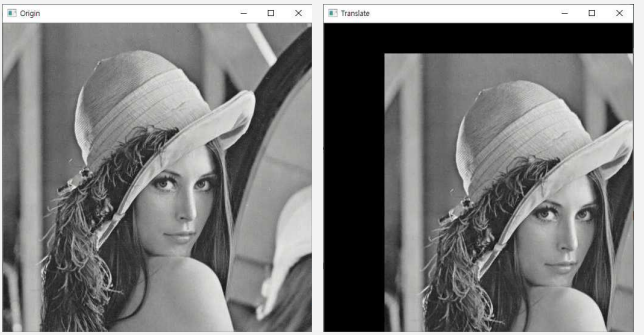
다음 코드는 이미지의 위치를 변경하는 예입니다.

```
import numpy as np

def translate(img, dist=(0,0)):
    height, width = img.shape[:2]
    img_ = np.zeros(img.shape, dtype=np.uint8)
    for y in range(height):
        for x in range(width):
            X = x + dist[0]
            Y = y + dist[1]
            if (X<0) ¦ (X>=width) ¦ (Y<0) ¦ (Y>=height):
```

```
                continue
            img_[Y, X] = img[y, x]

    return img_
```

다음 코드는 원본 이미지를 x축 방향으로 100픽셀, y축 방향으로 50픽셀 이동시킵니다.

```
import cv2

img = cv2.imread("lena.jpg")

cv2.imshow("Origin", img)
cv2.imshow("Translate", translate(img, (100, 50)))
cv2.waitKey()
cv2.destroyAllWindows()
```



4절. 회전

이미지의 특정 위치를 기준으로 회전시키는 것을 말합니다. 보통 회전의 기준이 되는 위치는 이미지의 정 중앙이며, 회전 각도 θ(θ는 라디안 각도)가 양수이면 시계 반대 방향으로 회전합니다.

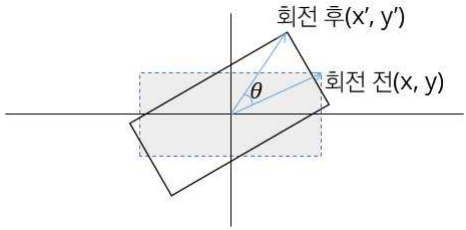


그림 14. 이미지 회전

위 그림에서 x', y'를 구하려면 아래의 행렬식을 이용합니다.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

위 행렬을 풀이하면 아래의 식이 만들어집니다.

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + ycos\theta$$

이 수식에서 우리가 알고 있는 것은 (x, y) 좌표 화소의 값이므로 새로운 좌표 (x', y')의 화소 값을 알려면 역변환 식을 구해야 합니다.

$$x = x'\cos\theta + y'\sin\theta$$
$$y = -x'\sin\theta + y'\cos\theta$$

이 식을 통해서 얻어지는 (x, y) 좌표의 화소값이 회전된 좌표(x', y')의 화소 값이 됩니다.

다음 코드는 이미지의 중심을 기준으로 θ만큼 회전시키는 예입니다.

```
import numpy as np

def rotate(img, deg=30):
    height, width = img.shape
    img_ = np.zeros(img.shape, dtype=np.uint8)
    theta = np.pi / (180.0 / deg)
    center_x = int(width / 2)
    center_y = int(height / 2)

    for y in range(height):
        for x in range(width):
            X = int((x-center_x)*np.cos(theta) +
                    (y-center_y)*np.sin(theta)) + center_x
            Y = int(-(x-center_x)*np.sin(theta) +
                    (y-center_y)*np.cos(theta)) + center_y
            if (X<0) ¦ (X>=width) ¦ (Y<0) ¦ (Y>=height):
                continue
            img_[Y, X] = img[y, x]

    return img_
```

다음 코드는 원본 이미지와 회전 후 이미지를 출력합니다.

```
import cv2

img = cv2.imread("lena-256.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow("Origin", img)
cv2.imshow("Rotate", rotate(img))
cv2.waitKey()
cv2.destroyAllWindows()
```

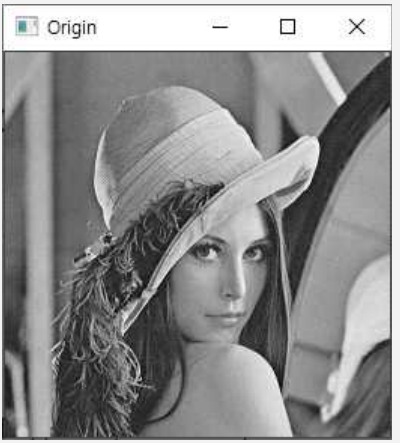


그림 15. 회전하기 전 원본

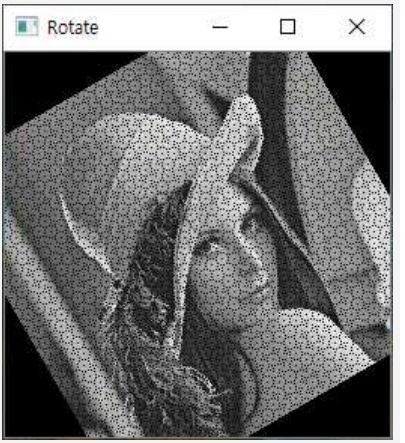


그림 16. 회전 후 Hole이 있는 이미지

앞의 회전은 언뜻 보면 회전이 잘 된 것처럼 보이지만 자세히 보면 일부 점들이 까맣게 보입니다. 이미지를 회전시킬 경우 회전 후 매핑될 이미지의 화소가 없으면 이미지에 까만 점이 표시되는데 이것을 홀(Hole)이라 부릅니다.

회전의 경우에 홀을 없애는 쉬운 방법이 있는 데, 그것은 해당 화소가 까만 점(0)일 경우 주위 화소의 평균값으로 대체하는 방법입니다. 회전 시 홀을 없애기 위해 주위화소의 평균값을 이용할 수 있지만 최근방법이나 선형보간법 등 보간법을 사용해서 회전 시 생기는 홀을 없앨 수 있습니다.

다음 코드는 회전을 하고 평균값을 이용해서 홀을 없애는 코드입니다. 앞의 코드에 홀을 없애기 위해 두 번째 반복문을 추가했습니다. 18라인의 반복문이 홀을 없애는 코드입니다. 이전 예제 소스코드에 18라인 이하의 내용을 추가하면 실습시간을 조금 줄일 수 있습니다.

```
import numpy as np

def rotate2(img, deg=30):
    height, width = img.shape
    img_ = np.zeros(img.shape, dtype=np.uint8)
    theta = np.pi / (180.0 / deg)
    center_x = int(width / 2)
    center_y = int(height / 2)

    for y in range(height):
        for x in range(width):
            X = int((x-center_x)*np.cos(theta) +
                    (y-center_y)*np.sin(theta)) + center_x
            Y = int(-(x-center_x)*np.sin(theta) +
                    (y-center_y)*np.cos(theta)) + center_y
            if (X<0) ¦ (X>=width) ¦ (Y<0) ¦ (Y>=height):
                continue
            img_[Y, X] = img[y, x]

    for y in range(1,height-1):
        for x in range(1,width-1):
            if img_[y, x] == 0:
                outer = \
                 [img_[y-1,x-1], img_[y-1,x], img_[y-1,x+1],
                 img_[y  ,x-1],               img_[y  ,x+1],
                 img_[y+1,x-1], img_[y+1,x], img_[y+1,x+1] ]
                img_[y, x] = np.uint8(np.mean(outer))

    return img_
```

다음은 홀을 없앤 코드를 실행시켜 회전 후 이미지를 확인합니다.

```
import cv2

img = cv2.imread("lena256.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow("Rotate", rotate2(img))
```

```
5  cv2.waitKey()
6  cv2.destroyAllWindows()
```

그림 17. 회전 후 Hole을 없앤 이미지

이 예제는 평균값을 이용해서 Hole을 없앴습니다. 크기변환에서 설명한 최근방법이나 선형 보간법을 이용해서 Hole을 없앨 수 있습니다.

5절. 아핀 변환

5.1. 아핀 변환

기하학에서, 아핀 변환(-變換, 영어: Affine Transformation)은 아핀 기하학적 성질들을 보존하는 두 아핀 공간 사이의 함수입니다.

아핀 변환은 이동, 확대/축소, 반전까지 포함된 변환입니다. 아핀 변환 작업은 선의 평행성은 유지가 되면서 이미지를 변환하는 작업입니다.

이동과 회전을 위해 좌표를 직접 계산해서 변환할 수 있습니다. 그러나 변환 행렬을 이용해서 이동 또는 회전등의 변환을 수행하면 더 빠르게 변환된 이미지를 얻을 수 있습니다.

5.2. 변환 행렬

아핀 변환을 위해서는 3개의 일치(Match)가 되는 점이 있으면 변환 행렬을 구할 수 있습니다.

$$X = (x - x_0) * cos\theta + (y - y_0) * sin\theta - x_0$$
$$Y = -(x - x_0) * sin\theta + (y - y_0) * cos\theta + y_0$$

앞에서 회전의 예에서처럼 아핀 변환을 위한 코드를 직접 구현할 수 있습니다. 그런데 다행히도 OpenCV에는 변환 행렬을 구할 수 있는 함수가 있습니다.

cv2.getRotationMatrix2D() 함수는 중심점, 회전각도, 크기 배율을 이용하여 변환 행렬을 생성합니다.

cv2.getRotationMatrix2D(center, angle, scale) -> M

- center - 이미지의 중심 좌표입니다.
- angle - 회전 각도를 지정합니다.
- scale - 크기 배율입니다. 0.5이면 1/2크기, 1이면 원래 이미지 크기입니다.
- 사용 예

```
rows, cols = img.shape[:2]
M= cv2.getRotationMatrix2D((cols/2, rows/2), 90, 0.5)
```

- 반환값 M - 이 함수의 반환값은 변환 행렬입니다.

cv2.getAffineTransform() 함수는 변환 전/후 대응하는 3점의 위치를 이용해서 3점 쌍에 대한 변환 행렬을 생성합니다.

cv2.getAffineTransform(src, dst) -> M

- src - 입력 영상의 3점입니다. [[x1, y1], [x2, y2], [x3, x4]] 형식으로 3점의 위치를 지정합니다.
- dst - 변환 후 3점입니다. [[x1, y1], [x2, y2], [x3, x4]] 형식으로 3점의 위치를 지정합니다.

- 반환값 M - 이 함수의 반환값은 변환 행렬입니다.

4개의 점을 이용해서 원근법 변환을 위한 변환 행렬을 구하는 함수도 있습니다. 그것은 원근법 변환을 설명할 때 같이 설명합니다.

이 함수들의 결과는 이미지 객체가 아니고 변환 행렬입니다. 그러므로 실제 회전된 이미지를 얻으려면 cv2.warpAffine() 함수를 사용해야 합니다. 다음은 cv2.warpAffine() 함수를 이용한 아핀 변환에 대해 설명합니다.

5.3. cv2.warpAffine()

cv2.warpAffine() 함수는 변환 행렬을 적용한 배열을 반환합니다.

cv2.warpAffine(src, M, dsize) -> dst

- src – 아핀 변환을 위한 원본 이미지 객체입니다.
- M – 변환 행렬입니다.
- dsize – 출력 이미지의 크기를 튜플 형식(cols, rows)으로 지정합니다.

- 반환값 dst – 변환 행렬을 적용한 결과 이미지 개체입니다.

다음 코드는 cv2.getRotationMatrix2D() 함수를 이용해서 이미지의 가운데를 회전의 중심으로 하고, 시계방향 30도, 1.2배 크기로 크기를 변경하는 변환 행렬을 생성합니다. 그리고 cv2.warpAffine() 함수를 이용해서 변환 행렬이 적용된 배열을 얻습니다.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('lena.jpg')
6  rows, cols, ch = img.shape
7
8  M = cv2.getRotationMatrix2D((rows/2,cols/2), -30, 1.2)
9
10 dst = cv2.warpAffine(img, M, (cols,rows))
11
12 plt.subplot(121)
13 plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
14 plt.title('image')
15 plt.xticks([])
16 plt.yticks([])
17
18 plt.subplot(122)
19 plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
```

```
20  plt.title('Affine')
21  plt.xticks([])
22  plt.yticks([])
23  plt.show()
```

다음 코드는 cv2.getAffineTransform() 함수와 변환 전/후 3점의 위치를 이용해서 변환 행렬을 구하고 warpAffine() 함수를 이용해서 변환 행렬이 적용된 배열을 얻습니다.

```
1   import cv2
2   import numpy as np
3   from matplotlib import pyplot as plt
4
5   img = cv2.imread('lena.jpg')
6   rows, cols, ch = img.shape
7
8   pts1 = np.float32([[200,100],[400,100],[200,200]])
9   pts2 = np.float32([[200,300],[400,200],[200,400]])
10
11  cv2.circle(img, (200,100), 10, (255,0,0),-1)
12  cv2.circle(img, (400,100), 10, (0,255,0),-1)
13  cv2.circle(img, (200,200), 10, (0,0,255),-1)
```

```
14
15  M = cv2.getAffineTransform(pts1, pts2)
16
17  dst = cv2.warpAffine(img, M, (cols,rows))
18
19  plt.subplot(121)
20  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
21  plt.title('image')
22  plt.xticks([])
23  plt.yticks([])
24
25  plt.subplot(122)
26  plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
27  plt.title('Affine')
28  plt.xticks([])
29  plt.yticks([])
30  plt.show()
```

6절. 원근법 변환

6.1. 원근법 변환

원근법(Perspective) 변환은 직선의 성질만 유지가 되고, 선의 평행성은 유지가 되지 않는 변환입니다.

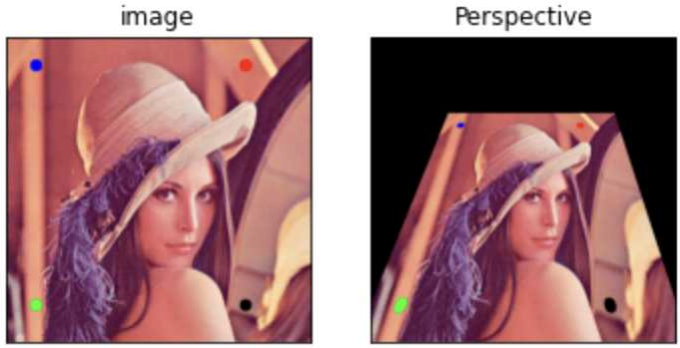


그림 19. 원근법 변환

6.2. 변환 행렬

cv2.getPerspectiveTransform(src, dst) 함수는 4점 쌍을 이용해서 변환 행렬을 구하기 위해서 사용합니다.

```
cv2.getPerspectiveTransform(src, dst) -> M
```

- src – 소스 이미지에서 사각형 꼭지점의 좌표입니다.
- dst – 대상 이미지에서 사각형 꼭지점의 좌표입니다.

- 반환값 M – 이 함수의 반환값은 변환 행렬입니다.

이 함수에 의해 만들어진 변환 행렬이 적용된 이미지를 얻기 위해서는 cv2.warpPerspective() 함수를 이용합니다.

6.3. cv2.warpPerspective()

cv2.warpPerspective() 함수는 변환 행렬을 이용하여 원근법 변환을 하는 함수입니다.

cv2.warpPerspective(src, M, dsize, flags) -> dst

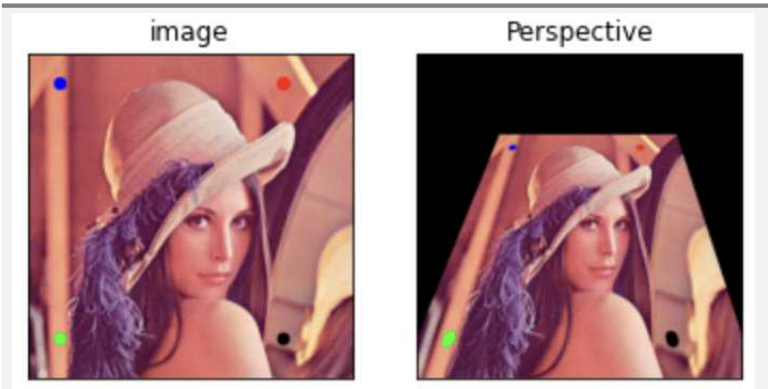
- src – 입력 이미지입니다.
- M – 변환 행렬입니다.
- dsize – 출력 이미지의 크기입니다.
- flags – 보간법(INTER_LINEAR 또는 INTER_NEAREST)과 선택적 플래그 WARP_INVERSE_MAP의 조합으로 M을 역변환으로 설정합니다. 기본값은 CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS

- 반환값 dst – 이 함수의 반환값은 변환 행렬이 적용된 이미지 객체입니다.

다음 코드는 4점을 이용해서 변환 행렬을 구한 후 원근법 변환을 합니다. 결과 화면에서 4점은 코드에서 pts1과 pts2의 위치입니다.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('lena.jpg')
6
7  pts1 = np.float32([[50,50],[50,450],[400,50],[400,450]])
8  pts2 = np.float32([[150,150],[50,450],[350,150],[400,450]])
9
10 cv2.circle(img, (50,50), 10, (255,0,0), -1)
11 cv2.circle(img, (50,450), 10, (0,255,0), -1)
12 cv2.circle(img, (400,50), 10, (0,0,255), -1)
13 cv2.circle(img, (400,450), 10, (0,0,0), -1)
14
15 M = cv2.getPerspectiveTransform(pts1, pts2)
16 dst = cv2.warpPerspective(img, M, (512,512))
17
18 pplt.subplot(121)
```

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('image')
plt.xticks([])
plt.yticks([])

plt.subplot(122)
plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
plt.title('Perspective')
plt.xticks([])
plt.yticks([])

plt.show()
```



7절. 푸리에 변환

주변 픽셀과의 밝기 변화가 많은 곳은 고주파로, 변화가 적은 곳은 저주파로 표현이 가능합니다. 이미지에서 고주파의 의미는 경계선을 의미하고, 저주파는 배경을 의미합니다. 고주파를 제거하면 경계선이 사라지고, 저주파를 제거하면 경계선만 남게 됩니다.

7.1. 푸리에 변환

푸리에 변환(Fourier Transform)은 시간 도메인(X축)에서 표현된 신호(일반적인 파형 도표)를 주파수 도메인으로 변환하는 것입니다. 시간축이 제거되어 대상의 전체적인 특징을 파악할 수 있습니다. 푸리에 변환이 이미지에 적용되면 중심이 저주파 영역, 주변이 고주파 영역을 나타냅니다. 푸리에 변환을 하여 저주파 또는 고주파를 제거하고 다시 역으로 이미지로 변환함으로써 이미지를 가공할 수 있습니다.

원본 이미지 | 고주파 영역 제거 | 저주파 영역 제거

그림 20. 푸리에 변환

푸리에 변환 후 중심의 저주파를 제거하고 다시 이미지로 전환하면 이미지의 경계선만 남게 됩니다. 반대로 푸리에 변환 후 주변의 고주파를 제거하면 경계선이 사라지게 할 수 있습니다.

7.2. 넘파이를 이용한 푸리에 변환

푸리에 변환(Fourier Transform)을 적용하면 0,0, 즉 화면 좌측상단이 중심이고, 그곳에 저주파가 모여 있습니다. 분석을 쉽게 하기 위해 왼쪽상단 좌표 (0,0)을 이미지의 중심으로 이동시키고 로그 스케일링(Log Scaling)을 하여 분석이 용이한 결과값으로 변환합니다.

넘파이의 fft2()함수는 2차원 이산 푸리에 변환(Discrete Fourier Transform[6])을 계산합니다. 이 함수는 FFT(Fast Fourier Transform; 고속 푸리에 변환)를 사용하여 M차원 배열의 모든 축에 대해 n차원 이산 푸리에 변환을 계산합니다. 기본적으로 변환은 입력 배열의 마지막 두 축, 즉 2차원 FFT에서 계산됩니다.

```
numpy.fft.fft2(a, s=None, axes=(-2, -1), norm=None) -> out
```

- a – 입력배열입니다.
- s – 출력의 모양(변환된 각 축의 길이)입니다.
- axes – 축을 지정합니다.
- norm – 사용 가능한 노름은 {“backward”, “ortho”, “forward”}중 하나입니다.

- 반환값 out – 출력 배열입니다.

fftshift() 함수는 제로 주파수 성분을 스펙트럼의 중심으로 이동합니다.

```
numpy.fft.fftshift(x, axis=None) -> y
```

- x – 입력배열입니다.
- axis – 축을 지정합니다.

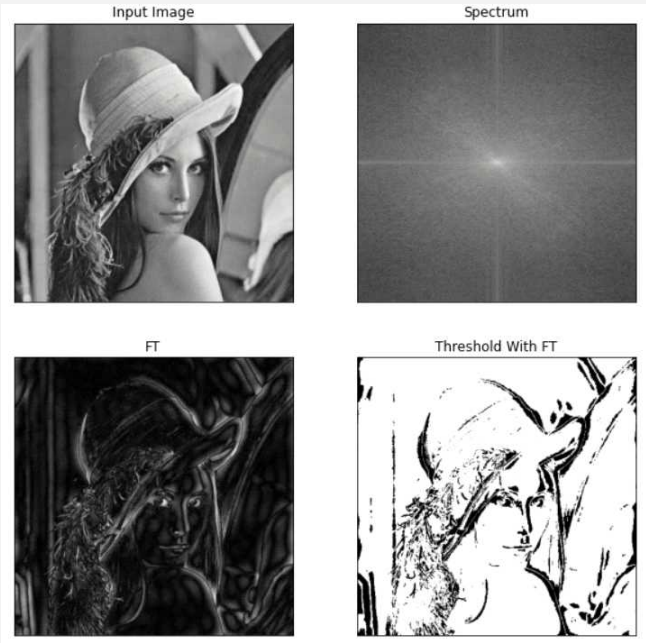
- 반환값 y – 이동된 배열입니다.

6) https://ko.wikipedia.org/wiki/이산_푸리에_변환

다음 코드는 넘파일 배열을 조작하여 푸리에 변환하는 예입니다.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
6
7  # 이미지에 푸리에 변환 적용
8  f = np.fft.fft2(img)
9  # 분석을 쉽게 하기 위해 주파수가 0인 부분을 중앙에 위치시킴.
10 # 중앙에 저주파가 모이게 됨.
11 fshift = np.fft.fftshift(f)
12 # spectrum을 구하는 수학식.
13 magnitude_spectrum = 20*np.log(np.abs(fshift))
14
15 rows, cols = img.shape
16 crow, ccol = int(rows/2), int(cols/2) # 이미지의 중심 좌표
17
18 # 중앙에서 10X10 사이즈의 사각형의 값을 1로 설정함.
19 # 중앙의 저주파를 모두 제거
20 # 저주파를 제거하였기 때문에 배경이 사라지고 경계선만 남게 됨.
21 d = 10
22
23 fshift[crow-d:crow+d, ccol-d:ccol+d] = 1
24
25 # 푸리에 변환결과를 다시 이미지로 변환
26 f_ishift = np.fft.ifftshift(fshift)
27 img_back = np.fft.ifft2(f_ishift)
28 img_back = np.abs(img_back)
29
30 # threshold를 적용하기 위해 float 타입을 int 타입으로 변환
31 img_new = np.uint8(img_back)
32 _, thresh = cv2.threshold(img_new,30,255,cv2.THRESH_BINARY_INV)
33
34 plt.subplot(221),plt.imshow(img, cmap='gray')
35 plt.title('Input Image')
36 plt.xticks([]), plt.yticks([])
```

```python

plt.subplot(222)
plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Spectrum')
plt.xticks([]), plt.yticks([])

plt.subplot(223)
plt.imshow(img_back, cmap='gray')
plt.title('FT')
plt.xticks([]), plt.yticks([])

plt.subplot(224),plt.imshow(thresh, cmap='gray')
plt.title('Threshold With FT')
plt.xticks([]), plt.yticks([])
plt.show()
```



7.3. OpenCV를 이용한 푸리에 변환

OpenCV는 푸리에 변환을 위해 cv2.dft() 함수와 cv2.idft() 함수를 제공합니다.

cv2.dft() 함수는 1D 또는 2D 배열의 이산 푸리에 변환(Discrete Fourier Transform)을 계산하고, cv2.idft() 함수는 1D 또는 2D 배열의 역(Inverse) 이산 푸리에 변환을 계산합니다.

cv2.dft(src, dst[, flags) -> dst

- src – 입력배열입니다. float32
- dst – 크기와 타입이 플래그에 따라 달라지는 출력 배열입니다.
- flags – DftFlags의 조합을 나타내는 변환 플래그입니다. 예제에서 사용된 cv2.DFT_COMPLEX_OUTPUT 플래그는 1D 또는 2D 실수 배열의 순방향 변환을 수행합니다.

- nonzeroRows – 매개변수가 0이 아닌 경우 함수는 입력 배열의 첫 번째 nonzeroRows행(DFT_INVERSE가 설정되지 않음) 또는 출력 배열의 첫 번째 nonzeroRows만(DFT_INVERSE가 설정됨) 0이 아닌 행을 포함한다고 가정합니다. 따라서 함수는 나머지 행을 보다 효율적으로 처리하고 시간을 절약합니다. 이 기술은 DFT를 사용하여 배열 상호 상관 또는 컨볼루션을 계산하는 데 매우 유용합니다.

- 반환값 dst – 크기와 타입이 플래그에 따라 달라지는 출력 배열입니다.

fftshift() 함수는 제로 주파수 성분을 스펙트럼의 중심으로 이동합니다.

numpy.fft.fftshift(x, axis=None) -> y

- x – 입력배열입니다.
- axis – 축을 지정합니다.

- 반환값 y – 이동된 배열입니다.

cv2.magnitude() 함수는 2D 벡터의 크기를 구하는 함수입니다.

cv2.magnitude(x, y) -> magnitude

- x - 벡터의 x 좌표로 구성된 부동 소수점 배열입니다.
- y - 벡터의 y 좌표로 구성된 부동 소수점 배열입니다. x와 크기가 같아야 합니다.

- 반환값 magnitude - x와 같은 크기와 타입의 출력 배열입니다.

코드에서는 d=20 크기의 사각형을 생성한 후, 사각형 바깥쪽을 제거하는 형태입니다. 그러면 고주파영역을 제거하게 됩니다. d 값이 작을수록 사각형이 작고, 바깥영역 즉, 고주파영역이 많이 제거되기 때문에 이미지가 뭉게지고 d 값이 클수록 사각형이 크고, 바깥영역 즉, 고주파 영역이 적게 제거되기 때문에 원래 이미지와 가까워집니다.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('lenagray.png', cv2.IMREAD_GRAYSCALE)
6  dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
7
8  dft_shift = np.fft.fftshift(dft)
9
10 rows, cols = img.shape
11 crow, ccol = int(rows/2), int(cols/2)
12
13 d = 20
14 mask = np.zeros((rows,cols,2), np.uint8)
15 mask[crow-d:crow+d, ccol-d:ccol+d] = 1
16
17 fshift = dft_shift*mask
18 f_ishift = np.fft.ifftshift(fshift)
19 img_back = cv2.idft(f_ishift)
20 img_back = cv2.magnitude(img_back[:,:,0], img_back[:,:,1])
```

```
21
22  plt.subplot(121)
23  plt.imshow(img, cmap='gray')
24  plt.title('Input Image')
25  plt.xticks([]), plt.yticks([])
26  plt.subplot(122)
27  plt.imshow(img_back, cmap='gray')
28  plt.title('FT')
29  plt.xticks([]), plt.yticks([])
30  plt.show()
```

위 코드에서 14, 15라인을 다음처럼 바꾸면 경계선을 찾아줍니다.

```
14  mask = np.ones((rows,cols,2),np.uint8)
15  mask[crow-d:crow+d, ccol-d:ccol+d] = 0
```

8절. 문서 스캔하기

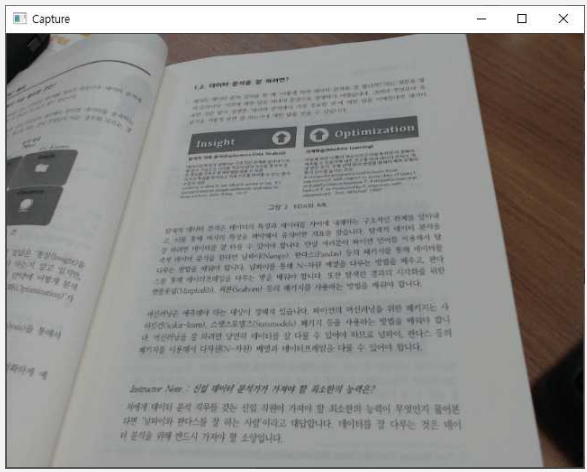
이 절에서 웹카메라로 문서를 스캔하는 프로그램을 만들어 보겠습니다. 사용한 함수는 원근법 변환에 v2.getPerspectiveTransform() 함수와 cv2.sarpPerspective() 함수입니다.

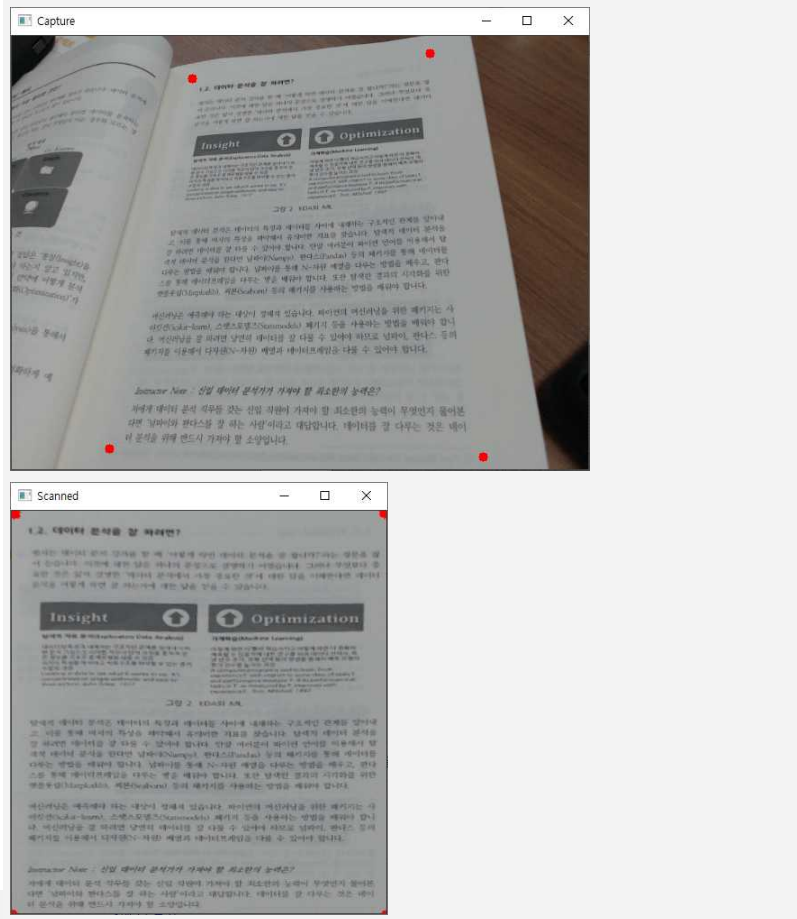
키보드 이벤트에서 c키를 누르면 그 순간의 화면이 캡쳐되고 그 화면에서 마우스로 원근법 변환을 수행할 4개의 점을 표시합니다. 그러면 원근법 변환에 의해 직사각형 모형의 스캔된 이미지 문서가 만들어집니다.

```
1  import cv2
2  import numpy as np
3  from datetime import datetime
4  from time import sleep
5
6  points = np.zeros((4,2), dtype=np.float32)
7  count = 0
8
9  def mouseHandler(event, x, y, flags, param):
10     global count
11     if event==cv2.EVENT_LBUTTONDOWN:
12         cv2.circle(img, (x,y), 5, (0, 0, 255), -1)
13         cv2.imshow("Capture", img)
14         try:
15             points[count] = [x, y]
16             count += 1
17             if count == 4:
18                 sum_ = points.sum(axis=1)
19                 diff = np.diff(points, axis=1)
20
21                 top_left = points[np.argmin(sum_)]
22                 bottom_right = points[np.argmax(sum_)]
23                 top_right= points[np.argmin(diff)]
24                 bottom_left = points[np.argmax(diff)]
```

```
25
26                  pts1 = np.float32([top_left, top_right,
                                       bottom_right, bottom_left])
27
28                  width_bottom = abs(bottom_right[0]-bottom_left[0])
29                  width_top = abs(top_right[0]-top_left[0])
30                  height_right = abs(top_right[1]-bottom_right[1])
31                  height_left = abs(top_left[1]-bottom_left[1])
32
33                  width = int(max([width_bottom, width_top]))
34                  height = int(max([height_right, height_left]))
35
36                  pts2 = np.float32([[0,0], [width-1,0],
                                       [width-1,height-1],
                                       [0,height-1]])
37
38                  M = cv2.getPerspectiveTransform(pts1, pts2)
39
40                  dst = cv2.warpPerspective(img,M,(width,height))
41                  cv2.imshow("Capture", img)
42                  cv2.imshow("Scanned", dst)
43          except Exception as e:
44              print(e)
45
46  cap = cv2.VideoCapture(0)
47  captured = False
48
49  if cap.isOpened():
50      delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
51      while True:
52          ret, img = cap.read()
53          if ret:
54              cv2.imshow("Capture", img)
55              key = cv2.waitKey(delay)
56              if key & 0xFF == 27 :
57                  print("아무 작업도 하지 않고 종료함")
58                  break
```

```
59                  elif key == ord('c'): # 캡쳐시작은 c키를 누르세요.
60                      captured = True
61                      break
62              else:
63                  print(ret, img)
64                  break
65  else:
66      print("File not opened")
67
68  if captured:
69      cap.release()
70      while True:
71          cv2.imshow("Capture", img)
72          cv2.setMouseCallback("Capture", mouseHandler)
73          key = cv2.waitKey(delay)
74          if key & 0xFF == 27 :
75              print("ESC Key pressed")
76              break
77
78  cap.release()
79  cv2.destroyAllWindows()
```




Capture
Insight
Optimization
Scanned
Insight
Optimization

4장. 엣지 검출

1절. 이미지 필터와 합성곱

1.1. 이미지 필터

이미지 필터 커널(Filter Kernel; 필터) 행렬의 특성에 따라 원본 이미지로부터 특성이 강조된 이미지를 얻을 수 있습니다. 어떤 필터를 사용하느냐에 따라 그 결과는 전혀 다릅니다.

다음 그림은 원영상(왼쪽위)에 여러 필터를 적용했을 때의 결과입니다. 어떤 필터는 경계선 부분을 더 뚜렷하게 하기도 하지만 어떤 필터는 더 흐리게 하기도 합니다. 어떤 필터를 사용하면 이미지의 엣지(Edge)를 찾아주기도 합니다. 엣지는 이미지의 경계선 또는 윤곽선이라고 부르기도 합니다.

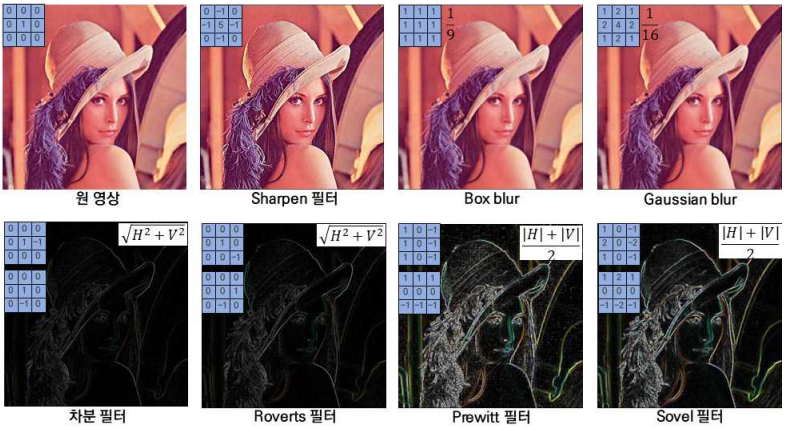


그림 1. 이미지 필터 처리의 예

일반적으로 필터는 영상의 주목 화소를 포함한 주위 화소들과 1:1로 대응되어 곱해진 후 합해집니다.

1.2. 합성곱

합성곱(컨볼루션; Convolution)은 필터를 적용할 수 있는 영역에 대해서 현재 주목 화소와 주위 화소의 배열을 각 요소끼리 모두 곱한 후 모든 항목을 더합니다. 합성곱에 사용하는 필터의 크기는 정해진 것은 없습니다. 일반적으로 현재 화소와 주위의 화소에 필터(3x3 또는 5x5)를 적용해서 영상을 처리합니다.

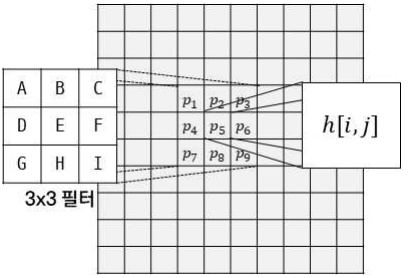


그림 2. 합성곱

3x3 필터를 적용하는 합성곱에서 현재 주목화소가 p_5이고 출력 결과가 *h[i, j]*라면 *h[i, j]*는 다음 식으로 계산됩니다.

$$h[i,j] = A \times p_1 + B \times p_2 + C \times p_3 + D \times p_4 + E \times p_5 + F \times p_6 + G \times p_7 + H \times p_8 + I \times p_9$$

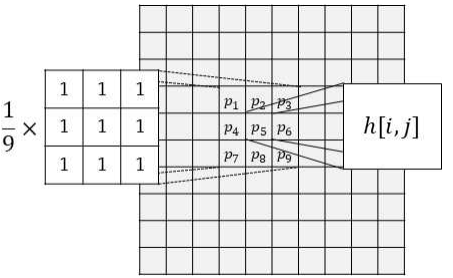


그림 3. Mean Filter

만일 그림3 같은 필터를 적용한다고 하면 현재 화소와 주위 화소는

1/9 가중치를 갖게 됩니다. 다시 말하면 현재 화소와 주위 화소의 값을 모두 더한 후 9로 나누게 되는데 이렇게 하면 영상이 흐릿하게 되는 효과가 있습니다. 이를 흐림(Blur) 효과라고 합니다. 그림 3의 필터는 흐림 효과를 주는 필터 중에서 평균값 필터에 해당합니다.

다음 5x5 이미지 배열에 3x3 크기 필터를 이용해서 합성곱을 수행하면 img[1,1]의 출력 화소의 값은 다음 식을 통해 계산됩니다.

img[1,1] = (1x0) + (3x1) + (3x0) + (0x1) + (3x0) + (1x1) + (0x1) + (0x1) + (2x0) = 4

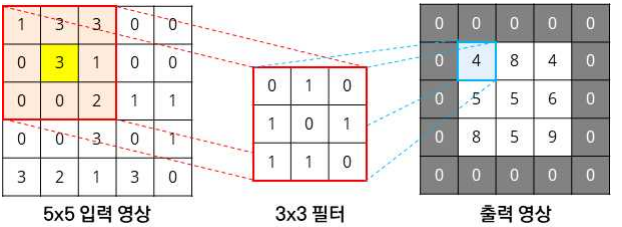


그림 4. 1행 1열 화소

- 바깥 화소의 경우는 필터를 적용할 수 없으므로 0을 출력합니다.

img[1,2] 화소는 다음 식을 통해 구해진 값이 출력됩니다.

img[1,2] = (3x0) + (3x1) + (0x0) + (3x1) + (1x0) + (0x1) + (0x1) + (2x1) + (1x0) = 8

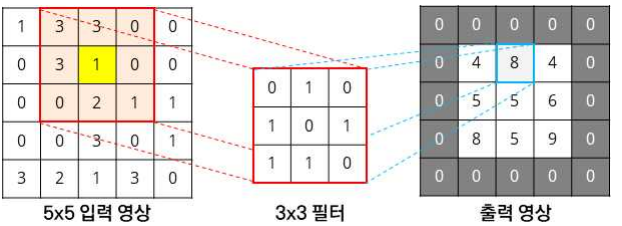


그림 5. 1행 2열 화소

img[1,3] 화소는 다음 식을 통해 구해진 값이 출력됩니다.

img[1,3] = (3x0) + (0x1) + (0x0) + (1x1) + (0x0) + (0x1) + (2x1) + (1x1) + (1x0) = 4

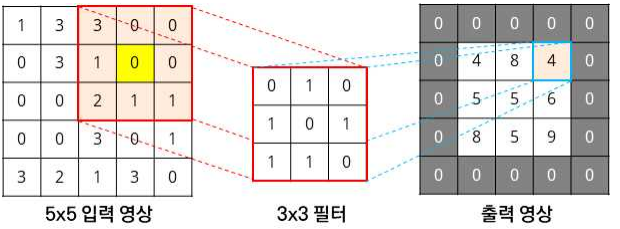


그림 6. 1행 3열 화소

이런 방법으로 모든 화소에 대해 합성곱을 수행하면 필터를 적용한 결과를 얻게 되는 것입니다.

1.3. 합성곱 함수 구현하기

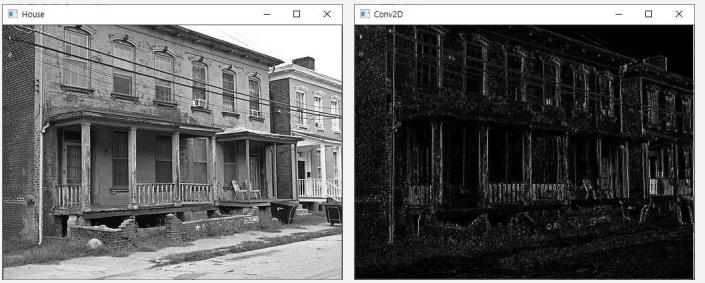
합성곱 함수가 있다면 다양한 필터를 여러분이 테스트해 볼 수 있을 것입니다. 다음은 이미지에 필터를 적용하는 합성곱 함수를 구현합니다. 함수의 padding 매개변수가 same이면 필터의 크기에 의해 계산되지 않는 바깥 테두리의 0화소가 생기지 않게 해 줍니다.

```
import numpy as np

def Conv2D(img, kernel=None, padding="valid"):
    if kernel is not None :
        h, w = img.shape[0:2]
        sy, sx = int(len(kernel)/2), int(len(kernel[0])/2)
        if padding=="same":
            new_shape = (h+sy, w+sx)
            img_out = np.zeros(new_shape, dtype=np.uint8)
```

```
10          elif padding=="valid":
11              img_out = np.zeros(img.shape, dtype=np.uint8)
12
13          height, width = img_out.shape
14          for y in range(sy, height-sy):
15              for x in range(sx, width-sx):
16                  roi = img[y-sy:y+sy+1, x-sx:x+sx+1]
17                  y_, x_ = roi.shape
18                  filtered = roi * kernel[:y_, :x_]
19                  pixel_value = np.abs(np.sum(filtered))
20                  img_out[y, x] = np.uint8(pixel_value)
21
22          return img_out[sy:-sy, sx:-sx]
23      else:
24          print("Kernel array not found!")
```

다음 코드는 합성곱 함수를 이용해서 필터를 적용하는 예입니다.

```
1  import cv2
2
3  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
4  kernel = np.array([[0,0,0],[0,1,-1],[0,0,0]])
5  output = Conv2D(img, kernel=kernel, padding="same")
6  cv2.imshow("House", img)
7  cv2.imshow("Conv2D", output)
8  cv2.waitKey()
9  cv2.destroyAllWindows()
```



1.4. cv2.filter2D()

OpenCV의 합성곱 함수는 cv2.filter2D()입니다. 이 함수를 이용하면 쉽게 필터를 적용할 수 있습니다.

cv2.filter2D(src, ddepth, kernel, anchor=None, delta=None, borderType=None) -> dst

- src - 입력 이미지입니다.
- ddepth - 출력 영상의 데이터 타입(예: cv2.CV_8U, cv2.CV_32F, cv2.CV_64F)을 지정합니다. -1을 지정하면 src와 같은 타입의 dst 영상을 생성합니다.
- kernel - 필터 커널 행렬입니다. 실수형입니다.
- anchor - 고정점의 위치입니다. (-1, -1)이면 필터 중앙을 고정점으로 사용합니다.
- delta - 추가적으로 더할 값입니다.
- borderType - 가장자리 픽셀 확장 방식을 지정합니다. Zero padding 또는 Reflect(mirror) padding 등을 적용할 수 있습니다. 사용할 수 있는 타입은 아래와 같습니다.

borderType	설명
BORDER_CONSTANT	iiiiii|abcdefgh|iiiiiii 일부 지정된 i
BORDER_REPLICATE	aaaaaa|abcdefgh|hhhhhhh
BORDER_REFLECT	fedcba|abcdefgh|hgfedcb
BORDER_WRAP	cdefgh|abcdefgh|abcdefg
BORDER_REFLECT_101	gfedcb|abcdefgh|gfedcba
BORDER_TRANSPARENT	uvwxyz|abcdefgh|ijklmno
BORDER_REFLECT101	BORDER_REFLECT_101과(와) 동일
BORDER_DEFAULT	BORDER_REFLECT_101과(와) 동일
BORDER_ISOLATED	ROI 밖을 사용하지 않습니다.

- 반환값 dst - 반환하는 값은 출력 영상입니다.

OpenCV에서 이미지의 데이터 타입은 아래와 같이 표현됩니다.

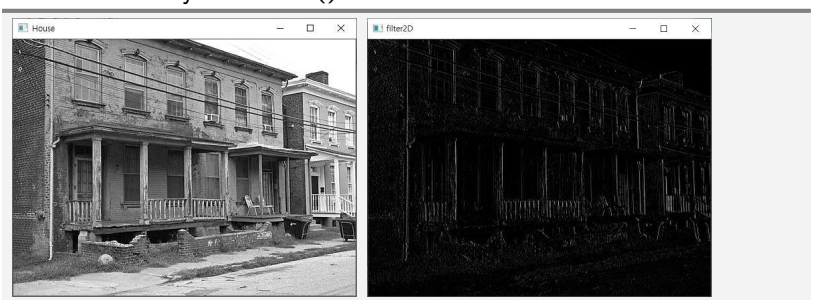
- CV_8U : 8-bit unsigned integer: uchar(0~255)
- CV_8S : 8-bit signed integer: schar(-128~127)
- CV_16U : 16-bit unsigned integer: ushort(0~65535)

- CV_16S : 16-bit signed integer: short(-32768~32767)
- CV_32S : 32-bit signed integer: int(-2147483648~2147483647)
- CV_32F : 32-bit floating-point number: float(-FLT_MAX~FLT_MAX, INF, NAN)
- CV_64F : 64-bit floating-point number: double(-DBL_MAX~DBL_MAX, INF, NAN)

일반적으로 데이터 타입과 채널 수가 같이 표현되는데, 만일 데이터 타입이 CV_8UC1이라면 이것은 8bit Unsigned integer이면서 채널(Channel)이 1개임을 의미합니다.

다음 코드는 cv2.filter2D() 함수를 사용하는 예입니다.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
6
7  kernel = np.array([[0,0,0],[0,1,-1],[0,0,0]])
8  output = cv2.filter2D(img, -1, kernel)
9
10 cv2.imshow("House", img)
11 cv2.imshow("filter2D", output)
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```



2절. 엣지 검출 필터

이미지의 특징을 찾기 위해 전처리하는 것 중 하나가 경계선 영역을 찾는 것입니다. 이미지의 경계선 또는 윤곽선을 엣지(Edge)라고 부르는데, 엣지를 추출하기 위해 차분 필터, 로버츠 필터, 프리위트 필터, 소벨 필터, 라플라시안 필터 등이 있습니다. 이 절에서는 엣지를 찾는 필터들을 소개하고 직접 함수로 구현합니다.

2.1. 차분 필터

차분(Differential) 필터는 현재 화소와 오른쪽 및 아래 이웃 화소의 차이를 이용해서 엣지를 찾습니다.

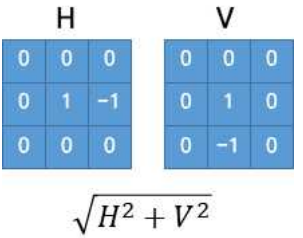


그림 7. 차분 필터

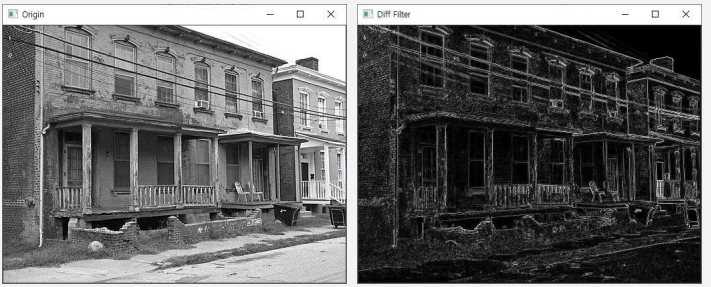
다음 코드는 차분필터를 적용하는 함수입니다.

```
import numpy as np

def diff_filter(img):
    height, width = img.shape
    img_ = np.zeros(img.shape, dtype=np.uint8)

```

```
    for y in range(height-1):
        for x in range(width-1):
            H = 1*img[y, x] + -1*img[y, x+1]
            V = 1*img[y, x] + -1*img[y+1, x]
            img_[y,x] = np.sqrt(H**2 + V**2)

    return img_
```

다음 코드는 차분 필터 함수를 적용한 결과를 확인합니다.

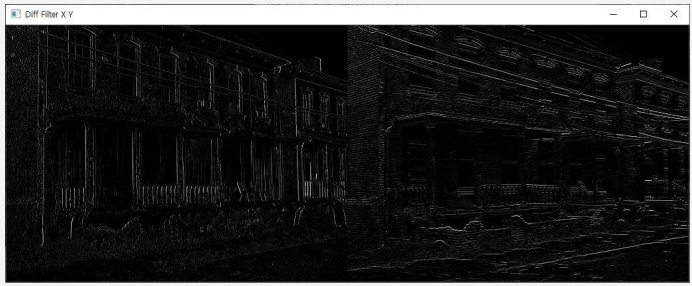
```
import cv2

img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow("Origin", img)
cv2.imshow("Diff Filter", diff_filter(img))
cv2.waitKey()
cv2.destroyAllWindows()
```

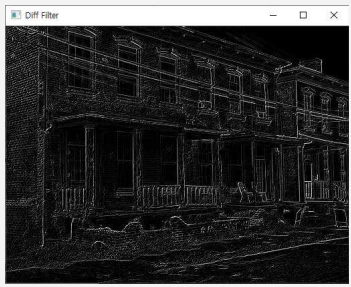


filter2D() 함수를 이용하면 차분필터를 적용할 수 있습니다. 다음 코드는 차분필터를 적용하는 예입니다.

```
import cv2
import numpy as np

img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
x_kernel = np.array([[-1, 1]])
y_kernel = np.array([[-1],[1]])
```

```
 7
 8  x_edge = cv2.filter2D(img, -1, x_kernel)
 9  y_edge = cv2.filter2D(img, -1, y_kernel)
10
11  cv2.imshow("Origin", img)
12  cv2.imshow("Diff Filter X Y", np.c_[x_edge, y_edge])
13  cv2.imshow("Diff Filter", x_edge + y_edge)
14  cv2.waitKey()
15  cv2.destroyAllWindows()
```





2.2. 로버츠 필터

로버츠(Rovert) 필터는 대각선 방향의 이웃 화소의 차이를 이용해서 엣지를 찾습니다. 그래서 대각선 경계를 찾는데 유리합니다.

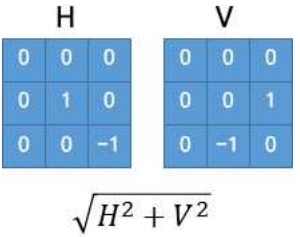


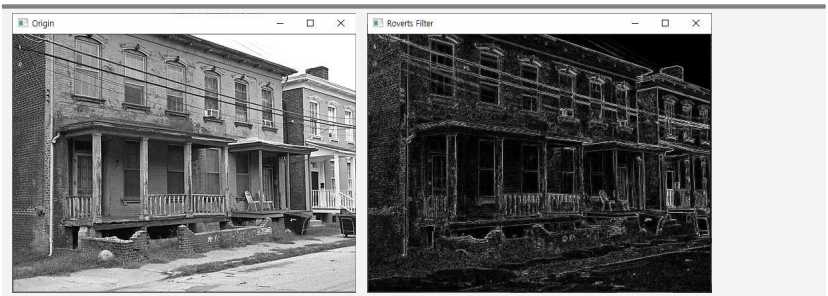
그림 8. 로버츠 필터

다음 코드는 로버츠 필터를 구현한 함수입니다. 현재 화소의 주위 8 근방 화소를 이용하여 처리하는 경우 가장 바깥 테두리의 화소는 처리할 수 없습니다. 그래서 프로그램의 인덱싱 오류를 발생시키지 않으려면 반복문의 range(height)는 range(1, height-1)로 바꾸고 range(width)는 range(1, width-1)로 바꾸야 합니다.

```
import numpy as np

def roverts_filter(img):
    height, width = img.shape
    img_ = np.zeros(img.shape, dtype=np.uint8)

    for y in range(1, height-1):
        for x in range(1, width-1):
            H =  1*img[y, x] +  -1*img[y+1, x+1]
            V =  1*img[y, x+1] + -1*img[y+1, x]
            img_[y,x] = np.sqrt(H**2 + V**2)

    return img_
```

다음 코드는 로버츠 필터를 구현한 함수를 적용합니다.

```
1  import cv2
2
3  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
4  cv2.imshow("Origin", img)
5  cv2.imshow("Roverts Filter", roverts_filter(img))
6  cv2.waitKey()
7  cv2.destroyAllWindows()
```

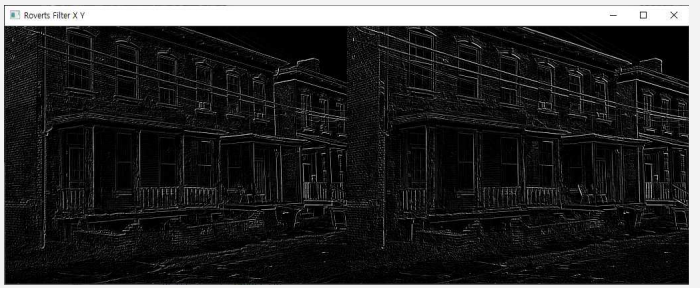


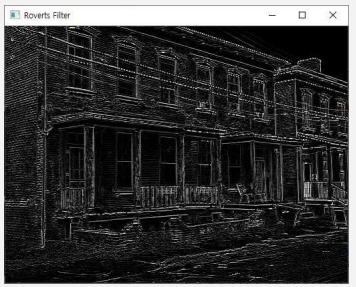
차분 필터와 로버츠 필터는 이웃하는 화소와의 차이를 이용해서 엣지를 찾습니다. 차분 필터는 이웃하는 오른쪽과 아래 화소와의 차이를 이용하므로 수직 성분과 수평 성분의 엣지에 더 민감합니다. 로버츠 필터는 현재 주목 화소를 기준으로 이웃 4개 화소의 대각선 화소의 차이를 이용하므로 대각선 성분의 엣지에 더 민감합니다.

다음 코드는 filter2D() 함수를 이용해 로버츠 필터를 적용합니다.

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
5  x_kernel = np.array([[1,0], [0,-1]])
6  y_kernel = np.array([[0,1], [-1,0]])
7
8  x_edge = cv2.filter2D(img, -1, x_kernel)
9  y_edge = cv2.filter2D(img, -1, y_kernel)
```

```
10
11  cv2.imshow("Origin", img)
12  cv2.imshow("Rovert Filter X Y", np.c_[x_edge, y_edge])
13  cv2.imshow("Rovert Filter", x_edge + y_edge)
14  cv2.waitKey()
15  cv2.destroyAllWindows()
```





2.3. 프리위트 필터

1970년 주디스 프리위트(Judith M. S. Prewitt)가 개발한 프리위트(Prewitt) 필터는 수평과 수직성분의 경계를 더 강조합니다. 이 필터의 결과는 그래디언트(Gradient) 벡터입니다. 각 지점에서 이미지 화소의 기울기를 계산하여 가능한 최대 증가 방향과 해당 방향의 변화율을 제공합니다

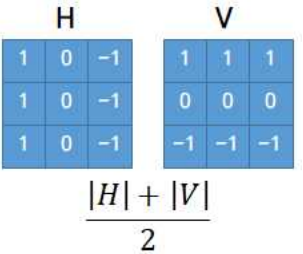


그림 9. 프리위트 필터

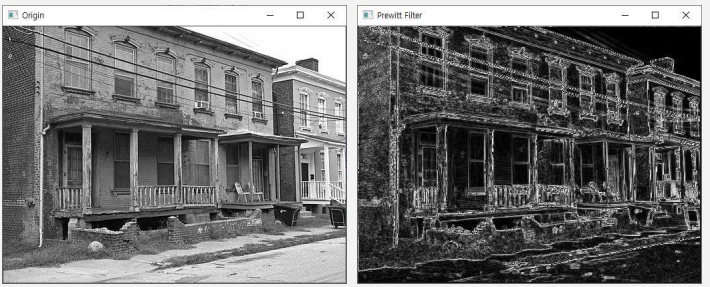
다음 코드는 프리위트 필터를 구현한 함수입니다. 프리위트 필터는 현재 화소 주위의 8근방 화소에 적용됩니다. 코드에서 9라인부터 14라인에 있는 반복문 안에 0이 곱해지는 곳은 코드를 작성할 필요가 없습니다. 그러나 이렇게 작성해 놓은 이유는 현재 화소와 주위 8근방 화소를 더 잘 이해할 수 있게 하기 위해서입니다.

```
1  import numpy as np
2
3  def prewitt_filter(img):
4      img_ = np.zeros(img.shape, dtype=np.uint8)
5      height, width = img.shape[:2]
6
7      for y in range(1, height-1):
8          for x in range(1, width-1):
9              H =  1*img[y-1,x-1] +  0*img[y-1,x] + -1*img[y-1,x+1] \
```

```
10                + 1*img[y  ,x-1] + 0*img[y  ,x] + -1*img[y  ,x+1] \
11                + 1*img[y+1,x-1] + 0*img[y+1,x] + -1*img[y+1,x+1]
12              V = 1*img[y-1,x-1] + 1*img[y-1,x] + 1*img[y-1,x+1] \
13                + 0*img[y  ,x-1] + 0*img[y  ,x] + 0*img[y  ,x+1] \
14                + -1*img[y+1,x-1] + -1*img[y+1,x] + -1*img[y+1,x+1]
15              img_[y,x] = (np.abs(H) + np.abs(V))/2
16
17      return img_
```

다음 코드는 프리위트 필터를 구현한 함수를 사용합니다.

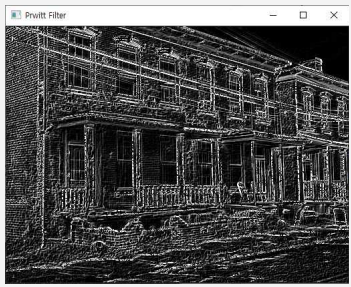
```
1  import cv2
2
3  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
4  cv2.imshow("Origin", img)
5  cv2.imshow("Prewitt Filter", presitt_filter(img))
6  cv2.waitKey()
7  cv2.destroyAllWindows()
```



다음 코드는 filter2D() 함수를 이용해 프리위트 필터를 적용합니다.

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
5  x_kernel = np.array([[-1,0,1], [-1,0,1], [-1,0,1]])
6  y_kernel = np.array([[-1,-1,-1], [0,0,0], [1,1,1]])
```

```
 7
 8  x_edge = cv2.filter2D(img, -1, x_kernel)
 9  y_edge = cv2.filter2D(img, -1, y_kernel)
10
11  cv2.imshow("Origin", img)
12  cv2.imshow("Diff Filter X Y", np.c_[x_edge, y_edge])
13  cv2.imshow("Diff Filter", x_edge + y_edge)
14  cv2.waitKey()
15  cv2.destroyAllWindows()
```



2.4. 소벨 필터

소벨(Sobel) 필터는 1968년 스탠포드 인공지능 연구소에서 어윈 소벨(Irwin Sobel)이 고안해낸 엣지 검출 알고리즘으로써 3x3크기의 행렬을 사용하여 연산했을 경우 중심을 기준으로 각 방향의 앞뒤의 값을 비교하여서 변화량을 검출하는 알고리즘입니다.

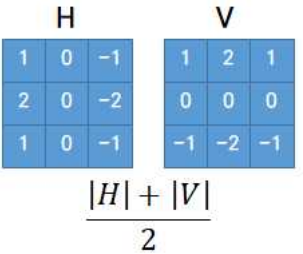


그림 10. 소벨 필터

소벨 필터는 프리위트 필터와 비교하면 소벨 필터의 경계선 영역의 값이 더 큰 것을 알 수 있는데, 이는 Sobel 연산자는 중앙값을 강조하기 위하여 2의 가중치를 사용하는 반면 Prewitt 연산자는 모두 동일하게 1의 가중치를 강조하여 필터링을 수행하기 때문입니다.

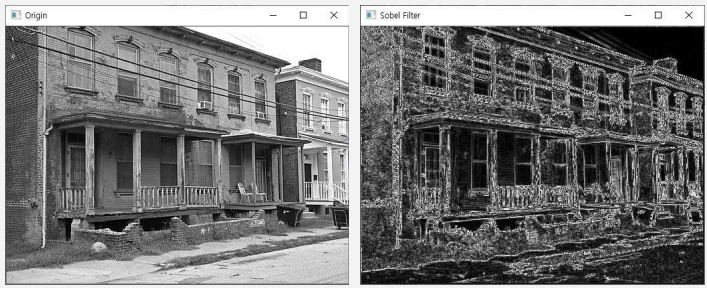
다음 코드는 소벨 필터를 구현한 함수입니다.

```
import numpy as np

def sobel_filter(img):
    img_ = np.zeros(img.shape, dtype=np.uint8)
    height, width = img.shape[:2]

    for y in range(1, height-1):
        for x in range(1, width-1):
            H =  1*img[y-1, x-1] +  0*img[y-1, x] + -1*img[y-1, x+1] \
              +  2*img[y  , x-1] +  0*img[y  , x] + -2*img[y  , x+1] \
```

```
                + 1*img[y+1, x-1] + 0*img[y+1, x] + -1*img[y+1, x+1]
            V = 1*img[y-1, x-1] + 2*img[y-1, x] + 1*img[y-1, x+1] \
                + 0*img[y  , x-1] + 0*img[y  , x] + 0*img[y  , x+1] \
                + -1*img[y+1, x-1] + -2*img[y+1, x] + -1*img[y+1, x+1]
            img_[y,x] = (np.abs(H) + np.abs(V))/2

    return img_
```

다음 코드는 소벨 필터를 구현한 함수를 적용합니다.

```
import cv2

img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow("Origin", img)
cv2.imshow("Sobel Filter", sobel_filter(img))
cv2.waitKey()
cv2.destroyAllWindows()
```



소벨 필터와 프리위트 필터의 결과를 비교해 보세요. 소벨 필터는 대각선 방향 엣지에 민감하게 반응하지만, 프리위트 필터는 대각선 방향 경계보다 수직/수평 방향 경계에 더 민감하게 반응한다는 특징이 있습니다.

2.5. cv2.Sobel()

cv2.Sobel() 함수는 소벨 필터를 구현한 함수입니다.

cv.Sobel(src, ddepth, dx, dy, dst, ksize, scale, delta) -> dst

- ▶ src - 입력 이미지입니다.
- ▶ ddepth - 출력 이미지의 깊이입니다. ddepth=-1이면 출력 이미지는 소스와 동일한 깊이를 갖습니다.
- ▶ dx, dy - x, y의 미분 차수입니다. x방향으로 1차 미분은 1, 0이며, y 방향으로 1차 미분은 0, 1입니다. 0, 1, 2중 하나여야 하며, dx, dy 둘 다 0일 수는 없습니다.
- ▶ dst - src와 동일한 크기 및 채널 수의 출력 이미지입니다.
- ▶ ksize - 필터 커널의 크기(1, 3, 5, 7중 하나)입니다.
- ▶ scale - 미분 값에 대한 계수입니다. 기본적으로 스케일링이 적용되지 않습니다.
- ▶ delta - dst에 저장하기 전에 결과에 더해져야 하는 값입니다.

▶ 반환값 dst - 소벨필터를 적용한 결과 이미지입니다.

다음 코드는 cv2.Sobel() 함수를 테스트합니다. x방향으로 1차미분과, y방향으로 1차미분하여 엣지를 검출하고 두 결과를 합한 이미지를 출력합니다.

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
5  x_edge = cv2.Sobel(img, -1, 1, 0, ksize=3)
6  y_edge = cv2.Sobel(img, -1, 0, 1, ksize=3)
7
8  cv2.imshow("Origin", img)
9  cv2.imshow("Sobel Filter X Y", np.c_[x_edge, y_edge])
10 cv2.imshow("Sobel Filter", x_edge+y_edge)
11 cv2.waitKey()
12 cv2.destroyAllWindows()
```

cv2.filter2D() 함수를 이용하려면 앞의 5, 6라인을 아래처럼 수정하면 됩니다.

```
5  x_kernel = np.array([[-1,0,1], [-2,0,2], [-1,0,1]])
6  y_kernel = np.array([[-1,-2,-1], [0,0,0], [1,2,1]])
7
8  x_edge = cv2.filter2D(img, -1, x_kernel)
9  y_edge = cv2.filter2D(img, -1, y_kernel)
```

● cv2.Sobel() 함수와 유사한 cv2.Scharr() 함수도 있습니다.

2.6. 라플라시안 필터

라플라시안(Laplace) 필터는 현재 주목화소에 가중치를 더 많이 부여합니다. 중앙값에 값에 가중치를 더 주면 테두리 성분이 더 강하게 나타나는 결과를 얻을 수 있습니다.

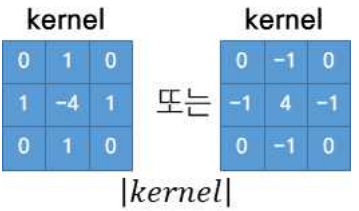


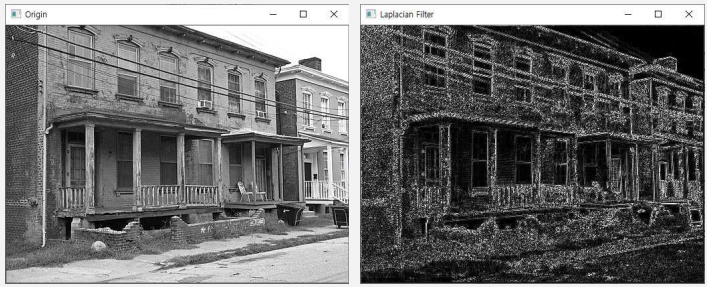
그림 11. 라플라시안 필터

다음 코드는 라플라시안 필터를 구현한 함수입니다.

```
import numpy as np

def laplacian_filter(img):
    img_ = np.zeros(img.shape, dtype=np.uint8)
    height, width = img.shape[:2]

    for y in range(1, height-1):
        for x in range(1, width-1):
            T =  0*img[y-1, x-1] + -1*img[y-1, x] +  0*img[y-1, x+1] \
              + -1*img[y  , x-1] +  4*img[y  , x] + -1*img[y  , x+1] \
              +  0*img[y+1, x-1] + -1*img[y+1, x] +  0*img[y+1, x+1]
            img_[y,x] = np.abs(T)

    return img_
```

다음 코드는 라플라시안 필터를 적용합니다.

```
1  import cv2
2
3  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
4  cv2.imshow("Origin", img)
5  cv2.imshow("Laplacian Filter", laplacian_filter(img))
6  cv2.waitKey()
7  cv2.destroyAllWindows()
```



앞 절에서 만들었던 합성곱 함수인 Conv2D()를 이용하여 라플라시안 필터를 적용하거나 cv2.Laplacian() 함수를 사용할 할 수 있습니다.

```
1  import cv2
2
3  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
4
5  filter = [[0,1,0],[1,-4,1],[0,1,0]]
6  mask = np.array(filter)
7  output = Conv2D(img, mask=mask)
8  # cv2.Laplacian(img, cv2.CV_64F) # 5,6,7라인 대신 사용 가능
9
10 cv2.imshow("House", img)
11 cv2.imshow("Laplacian Filter", output)
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```

* 실행 결과는 위 예제의 실행 결과와 같습니다.

2.7. cv2.Laplacian()

cv2.Laplacian() 함수는 라플라시안 필터터를 구현한 함수입니다.

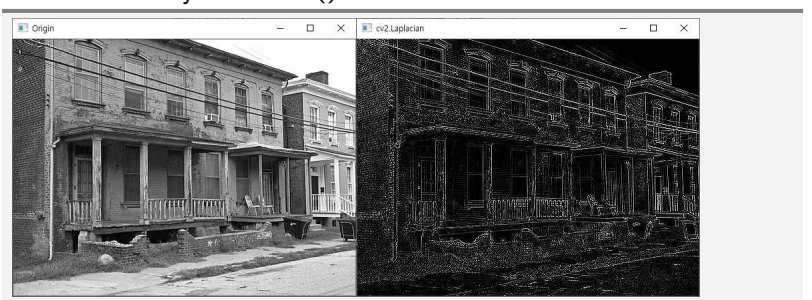
cv.Laplacian(src, ddepth, dst, ksize, scale, delta) -> dst

- src - 입력 이미지입니다.
- ddepth - 출력 이미지의 깊이입니다. ddepth=-1이면 출력 이미지는 소스와 동일한 깊이를 갖습니다.
- dst - src와 동일한 크기 및 채널 수의 출력 이미지입니다.
- ksize - 필터 커널의 크기(1, 3, 5, 7중 하나)입니다.
- scale - 미분 값에 대한 계수입니다. 기본적으로 스케일링이 적용되지 않습니다.
- delta - dst에 저장하기 전에 결과에 더해져야 하는 값입니다.

- 반환값 dst - 라플라시안 필터를 적용한 결과 이미지입니다.

다음 코드는 cv2.Laplacian() 함수를 테스트합니다.

```
1  import cv2
2
3  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
4
5  cv2.imshow("Origin", img)
6  cv2.imshow("cv2.Laplacian", cv2.Laplacian(img, -1))
7  cv2.waitKey()
8  cv2.destroyAllWindows()
```

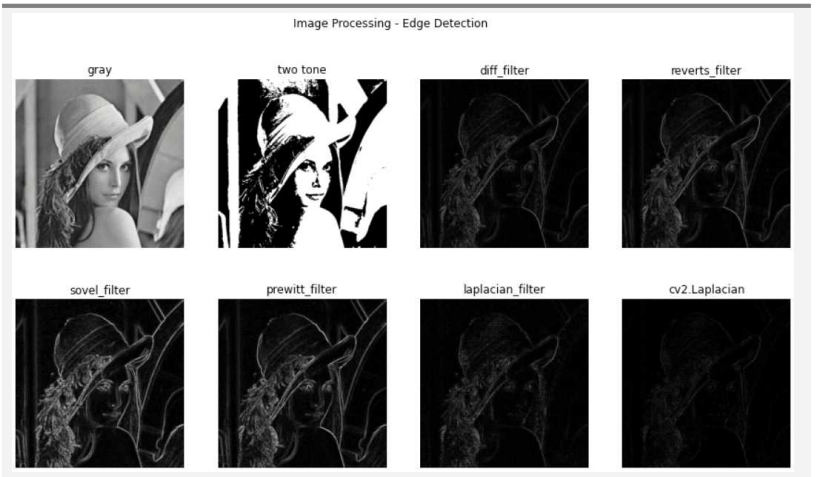


2.8. 엣지 추출 필터 비교

다음 코드는 앞에서 만든 엣지를 추출하는 함수를 이용해 엣지를 검출하고 그 결과를 출력합니다.

```
import cv2

lena_gray = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)

A = diff_filter(lena_gray)
B = reverts_filter(lena_gray)
C = sobel_filter(lena_gray)
D = prewitt_filter(lena_gray)
E = laplacian_filter(lena_gray)
F = cv2.Laplacian(lena_gray, cv2.CV_8U)

import matplotlib.pyplot as plt

fig, axes = plt.subplots(2,4, figsize=(15,8))
fig.suptitle("Image Processing - Edge Detection")
axes[0,0].imshow(lena_gray, cmap="gray")
axes[0,0].axis("off"); axes[0,0].set_title("gray")
axes[0,1].imshow(two_tone(lena_gray), cmap="gray")
axes[0,1].axis("off"); axes[0,1].set_title("two tone")
axes[0,2].imshow(A, cmap="gray")
axes[0,2].axis("off"); axes[0,2].set_title("diff_filter")
axes[0,3].imshow(B, cmap="gray")
axes[0,3].axis("off"); axes[0,3].set_title("reverts_filter")

axes[1,0].imshow(C, cmap="gray")
axes[1,0].axis("off"); axes[1,0].set_title("sobel_filter")
axes[1,1].imshow(D, cmap="gray")
axes[1,1].axis("off"); axes[1,1].set_title("prewitt_filter")
axes[1,2].imshow(E, cmap="gray")
axes[1,2].axis("off")
axes[1,2].set_title("laplacian_filter")
axes[1,3].imshow(F, cmap="gray")
```

```
axes[1,3].axis("off")
axes[1,3].set_title("cv2.Laplacian_filter")

plt.show()
```



3절. 캐니 엣지

3.1. 캐니 엣지

캐니 엣지는 1986년 존 캐니(John F. Canny)가 제안한 알고리즘입니다. 캐니 엣지 알고리즘은 다음 단계에 걸쳐 엣지를 검출합니다.

1. 노이즈를 제거 : 엣지 추출은 이미지의 노이즈에 취약하므로 첫 번째 단계는 5x5 가우시안 필터를 사용하여 이미지의 노이즈를 제거하는 것입니다.
2. 소벨 필터로 엣지를 찾기 : 수평 및 수직 방향 모두에서 소벨(Sobel) 필터 커널로 필터링되어 수평 방향(G_x) 및 수직 방향(G_y)에서 1차 도함수를 얻습니다. 그리고 이 두 이미지에서 다음과 같이 각 픽셀의 엣지 그래디언트와 방향을 찾을 수 있습니다.

$$Edge_Gradient\,(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\,(\theta) = \tan^{-1}\left(\frac{G_y}{G_y}\right)$$

   그래디언트 방향은 항상 엣지에 수직입니다. 수직, 수평 및 두 개의 대각선 방향을 나타내는 네 가지 각도 중 하나로 반올림됩니다.
3. 기울기 크기 임계값 또는 하한 컷오프 억제를 적용 : 그래디언트 크기와 방향을 얻은 후 엣지를 구성하지 않을 수 있는 원하지 않는 픽셀을 제거하기 위해 이미지 전체 스캔이 수행됩니다. 이를 위해 모든 픽셀에서 픽셀이 그래디언트 방향으로 이웃의 로컬 최댓값인지 확인합니다.
4. 히스테리시스 임계값 : 이 단계는 모든 엣지가 실제로 엣지인지 아닌지를 결정합니다. 이를 위해서는 minVal 및 maxVal 두 개

의 임계값이 필요합니다. 강도 기울기가 maxVal보다 큰 모든 엣지는 엣지가 되고 minVal 미만의 엣지는 엣지가 아니므로 버려집니다. 이 두 임계값 사이에 있는 것은 연결성을 기반으로 엣지 또는 비 엣지로 분류됩니다. "확실한 엣지" 픽셀에 연결되어 있으면 엣지의 일부로 간주됩니다. 그렇지 않으면 그것들도 버려집니다. 아래 이미지를 참조하세요.

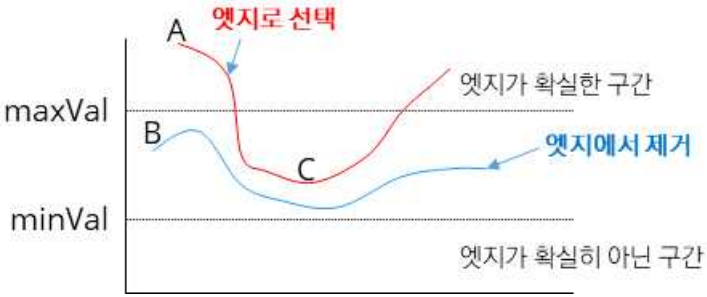


그림 12. 히스테리시스 임계값

엣지 A는 maxVal 위에 있으므로 "확실한 엣지"로 간주됩니다. 엣지 C가 maxVal 아래에 있지만 엣지 A에 연결되어 있으므로 유효한 엣지로 간주되어 전체 곡선을 얻습니다. 그러나 엣지 B는 minVal 보다 높고 엣지 C와 동일한 영역에 있지만 "확실한 엣지"에 연결되어 있지 않으므로 폐기됩니다. 따라서 올바른 결과를 얻으려면 minVal과 maxVal을 적절하게 선택해야 합니다.

이 단계에서는 엣지가 긴 선이라는 가정하에 작은 화소 잡음도 제거합니다.

3.2. cv2.Canny()

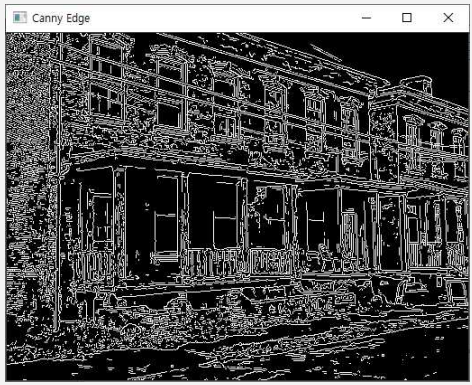
cv2.Canny()는 캐니 엣지 알고리즘을 구현한 함수입니다.

cv2.Canny(img, minVal, maxVal, apertureSize, L2gradient)

- img – 입력 이미지입니다.
- minVal – 캐니 엣지 알고리즘이 마지막 단계인 Hysteresis를 수행하기 위한 입계값1입니다.
- maxVal – 캐니 엣지 알고리즘이 마지막 단계인 Hysteresis를 수행하기 위한 임계값2입니다.
- apertureSize – 소벨 필터의 크기입니다. 3, 5, 7만 지정 가능합니다.
- L2gradient – 기울기 크기를 찾기 위한 방정식을 지정합니다. True이면 앞에서 설명한 Edge_Gradient 방정식을 사용하고 False이면 $Edge_Gradient\,(G) = |G_x| + |G_y|$를 이용합니다.

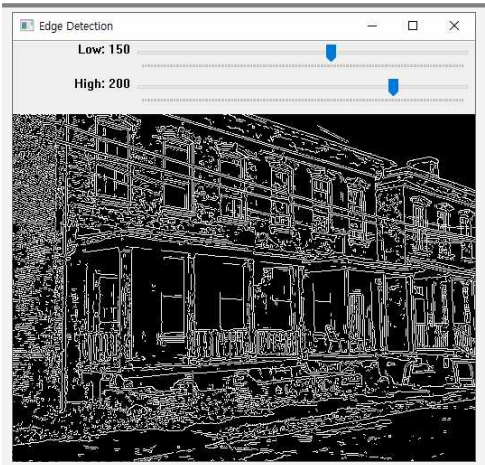
다음 코드는 케니 엣지 알고리즘으로 엣지를 검출합니다.

```
1  import cv2
2
3  img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
4  output = cv2.Canny(img, 150, 200)
5  cv2.imshow("Canny Edge", output)
6  cv2.waitKey()
7  cv2.destroyAllWindows()
```



다음 코드는 트랙바를 이용해서 minVal과 maxVal을 지정합니다. 트랙바를 수정하면 엣지가 적용됩니다. 아무키나 누르면 프로그램이 종료됩니다.

```
import cv2

def onChange(x):
    low = cv2.getTrackbarPos("Low", "Edge Detection")
    high = cv2.getTrackbarPos("High", "Edge Detection")
    if low > high:
        pass # Low threshold must be low than high threshold
    else:
        canny = cv2.Canny(img, low, high)
        cv2.imshow("Edge Detection", canny)

img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
cv2.namedWindow("Edge Detection")
cv2.createTrackbar("Low", "Edge Detection", 0, 255, onChange)
cv2.createTrackbar("High", "Edge Detection", 0, 255, onChange)

cv2.imshow("Edge Detection", img)
cv2.waitKey()
cv2.destroyAllWindows()
```



5장. 화질 개선

1절. 잡음 제거

잡음 제거는 영상처리에서 전처리에 필수 요소입니다. 이 절에서는 잡음을 어떻게 하면 효과적으로 제거할 수 있는지에 대해 설명합니다.

1.1. 평균값 필터

주목 화소와 주위 8근방 화소의 평균(Mean)을 이용해서 잡음을 제거합니다.

p_o	p_1	p_2
p_3	p_4	p_5
p_6	p_7	p_8

$$\frac{p_o + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

그림 1. 평균값 필터

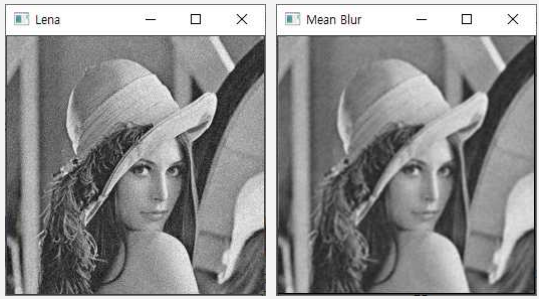
다음은 평균값 필터를 적용하는 함수입니다.

```
import numpy as np

def mean_blur(img):
    A = np.zeros(img.shape, dtype=np.uint8)
    height, width = img.shape
    for y in range(1, height-1):
        for x in range(1, width-1):
            S = 1*img[y-1, x-1] + img[y-1, x] + img[y-1, x+1] \
                + img[y  , x-1] + img[y  , x] + img[y  , x+1] \
```

```
                + img[y+1, x-1] + img[y+1, x] + img[y+1, x+1]

            S = S/9
            if S > 255:
                A[y, x] = 255
            elif S < 0:
                A[y, x] = 0
            else:
                A[y, x] = int(S)

    return A
```

다음 코드는 평균값 필터를 구현한 함수를 적용합니다.

```
import cv2

lena_gray = cv2.imread("lena_std_5.jpg", cv2.IMREAD_GRAYSCALE)

cv2.imshow("Lena", lena_gray)
cv2.imshow("Mean Blur", mean_blur(lena_gray))
cv2.waitKey()
cv2.destroyAllWindows()
```



5% 잡음 추가 영상　　평균값 필터 적용

1.2. cv2.blur()

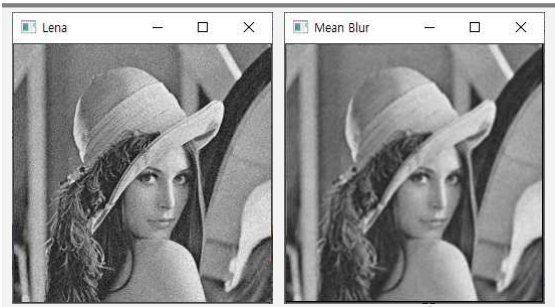
cv2.blur() 함수는 커널 영역의 모든 픽셀의 평균을 계산하고 현재 주목하는 중앙 화소를 이 평균으로 대체합니다. 이 함수는 커널의 너비와 높이를 지정해야 합니다.

cv2.blur(src, ksize[, borderType]) -> dst

- src - 입력 이미지입니다.
- ksize - 필터 커널의 크기입니다. ksize=(3,3) 형식으로 지정합니다.
- borderType - 가장자리 픽셀 확장 방식을 지정합니다. Zero padding 또는 Reflect(mirror) padding 등을 적용할 수 있습니다. cv2.BORDER_XXXX 형식으로 정의되어 있습니다. 더 자세한 내용은 cv2.filter2D() 함수의 설명을 참고하세요.

다음 코드는 cv2.blur() 함수를 적용합니다.

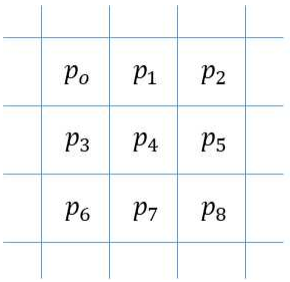
```
1  import cv2
2
3  lena_gray = cv2.imread("lena_std_5.jpg", cv2.IMREAD_GRAYSCALE)
4  cv2.imshow("Lena", lena_gray)
5  cv2.imshow("Mean Blur", cv2.blur(lena_gray, ksize=(3,3)))
6  cv2.waitKey()
7  cv2.destroyAllWindows()
```



5% 잡음 추가 영상 cv2.blur() 적용

1.3. 메디안 필터

주목 화소와 8근방 화소의 중앙값(Median)을 이용해서 잡음을 제거합니다.



p_o~p_8 화소를 정렬한 후 중앙 값을 선택

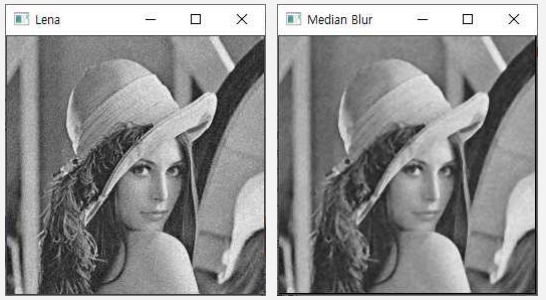
그림 2. 메디안 필터

다음 코드는 메디안 필터를 구현한 함수입니다.

```
import numpy as np

def median_blur(img):
    A = np.zeros(img.shape, dtype=np.uint8)
    height, width = img.shape[:2]
    for y in range(height):
        for x in range(width):
            S = [img[y-1, x-1], img[y-1, x], img[y-1, x+1], \
                 img[y  , x-1], img[y  , x], img[y  , x+1], \
                 img[y+1, x-1], img[y+1, x], img[y+1, x+1] ]

            A[y, x] = np.median(S)

    return A
```

다음 코드는 메디안 필터를 적용합니다.

```
import cv2

lena_gray = cv2.imread("lena_std_5.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow("Lena", lena_gray)
cv2.imshow("Median", median_blur(lena_gray))
cv2.waitKey()
cv2.destroyAllWindows()
```



5% 잡음 추가 영상　　메디안 필터 적용

1.4. cv2.medianBlur()

메디안 필터를 구현한 함수는 cv2.medianBlur() 입니다.

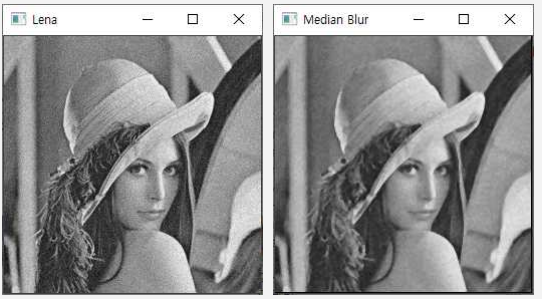
cv2.medianBlur(src, ksize[, borderType]) -> dst

- src - 입력 이미지입니다.
- ksize - 필터 커널의 크기입니다. ksize=3형식으로 지정합니다. 1보다 큰 홀수여야 합니다.

다음 코드는 cv2.medianBlur() 함수를 사용한 예입니다.

```
import cv2

lena_gray = cv2.imread("lena_std_5.jpg", cv2.IMREAD_GRAYSCALE)
```

```
4  cv2.imshow("Lena", lena_gray)
5  cv2.imshow("Median", cv2.medianBlur(lena_gray, 3))
6  cv2.waitKey()
7  cv2.destroyAllWindows()
```



5% 잡음 추가 영상　　cv2.medianBlur() 적용

1.5. scipy.ndimage.median_filter()

싸이파이(Scipy) 패키지에 메디안 필터를 구현한 함수가 있습니다. 이를 이용해서 메디안 필터를 적용할 수 있습니다. 이 함수는 참고만 하세요. 실제로 사용하는 일이 많지 않습니다.

scipy.ndimage.median_filter(input, size=None, footprint=None, output=None, mode='reflect', cval=0.0, origin=0)

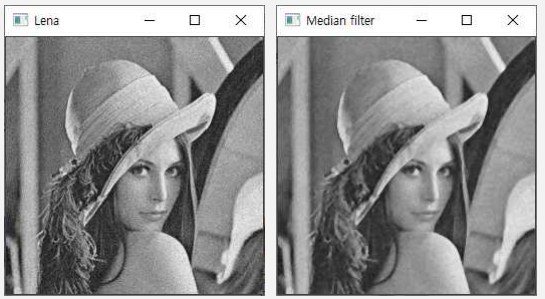
- input – 입력 이미지입니다.
- size – 필터 커널의 크기입니다. size=(n,m) 형식으로 지정합니다.
- footpring – size 또는 footprint 둘 중 하나는 지정해야 합니다. 풋프린트는 모양을 지정하는 논리 배열이지만 모양 내의 요소 중 어떤 요소가 필터함수로 전달될 것인지도 지정합니다. size=(n,m)은 footprint=np.ones((n,m))과 같습니다.
- output – 반환될 배열의 dtype입니다. 기본적으로 입력과 동일한 dtype의 배열이 생성됩니다.
- mode – 입력 배열의 경계를 넘어 확장되는지를 판정합니다. {'reflect',

'constant', 'nearest', 'mirror', 'wrap'} 중 하나를 선택합니다.

- cval – mode='constant'일 경우 입력의 가장자리를 채울 값입니다.
- origin – 입력 배열의 픽셀에서 필터 배치를 제어합니다. 값 0(기본값)은 필터를 픽셀 중앙에 배치하고 양수 값은 필터를 왼쪽으로, 음수 값은 오른쪽으로 이동합니다.

다음 코드는 싸이파이의 median_filter() 함수를 사용하는 예입니다.

```
1  from scipy import ndimage
2  import cv2
3
4  lena_gray = cv2.imread("lena_std_5.jpg", cv2.IMREAD_GRAYSCALE)
5  result = ndimage.median_filter(lena_gray, size=3)
6  cv2.imshow("Lena", lena_gray)
7  cv2.imshow("Lena", result)
8  cv2.waitKey()
9  cv2.destroyAllWindows()
```



5% 잡음 추가 영상　　median_filter() 적용

이 함수에 대해 더 자세한 내용을 보려면 아래 주소를 참고하세요.
https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html

1.6. 가우시안 필터

가우시안 분포를 갖는 커널을 이용해서 필터 처리를 할 수 있습니다. 가우시안 필터는 중앙값이 가장 크고 멀어질수록 그 값이 작아지는 커널을 사용합니다. 현재 주목 화소에 가중치를 더 부여하고 현재 화소와 상/하/좌/우 화소의 길이가 1이라면 대각선 성분의 길이는 $\sqrt{2}$이므로 상/하/좌/우 화소는 대각선 화소에 비해 더 높은 가중치를 줍니다.

1	2	1
2	4	2
1	2	1

$$\frac{1}{16}$$

그림 3. 가우시안 필터

다음 코드는 가우시안 필터를 구현한 함수입니다.

```
import numpy as np

def gaussian_blur(img):
    img_ = np.zeros(img.shape, dtype=np.uint8)
    height, width = img.shape
    for y in range(1, height-1):
        for x in range(1, width-1):
            T = 1*img[y-1,x-1] + 2*img[y-1,x] + 1*img[y-1,x+1] \
              + 2*img[y  ,x-1] + 4*img[y  ,x] + 2*img[y  ,x+1] \
              + 1*img[y+1,x-1] + 2*img[y+1,x] + 1*img[y+1,x+1]

            img_[y,x] = int(T/16)

    return img_
```

다음 코드는 가우시안 필터를 구현한 함수를 적용합니다.

```
1  import cv2
2
3  lena_gray = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
4  cv2.imshow("Lena", gaussian_blur(lena_gray))
5  cv2.waitKey()
6  cv2.destroyAllWindows()
```

5% 잡음 추가 영상 Gaussian Blur

1.7. cv2.GaussianBlur()

가우시안 필터링은 이미지에서 가우시안 노이즈를 제거하는 데 매우 효과적입니다. cv2.GaussianBlur() 함수를 이용하면 가우시안 필터링을 쉽게 적용할 수 있습니다.

cv2.GaussianBlur(src, ksize, sigmaX[, sigmaY, borderType]) -> dst

- src – 8비트 1채널 또는 3채널 입력 이미지입니다.
- ksize – 필터 커널의 크기입니다. ksize=(width, height) 형태로 지정합니다. width와 height는 서로 다를 수 있지만, 양수의 홀수로 지정해야 합니다.
- sigmaX, sigmaY – X방향과 Y방향 가우시안 커널 표준편차입니다.
- borderType – 이미지 외부의 픽셀을 외삽하는데 사용되는 테두리 모드입니다. cv2.BORDER_XXXX 형식으로 정의되어 있습니다.

1.8. cv2.bilateralFilter()

cv.bilateralFilter()는 가장자리를 선명하게 유지하면서 노이즈 제거에 매우 효과적입니다. 그러나 작업은 다른 필터에 비해 느립니다. 앞에서 보여준 가우시안 필터는 공간 만의 함수입니다. 즉, 필터링하는 동안 주변 픽셀이 고려됩니다. 픽셀의 강도가 거의 동일한지 여부는 고려하지 않습니다. 픽셀이 엣지 픽셀인지 여부는 고려하지 않습니다. 그래서 가우시안 필터는 우리가 원하지 않는 가장자리도 흐리게 만듭니다. 물론 가장자리가 흐려지는 것은 평균값 필터와 메디안 필터에서도 일어나는 현상입니다.

양방향 필터링은 공간에서 가우시안 필터를 사용하지만 화소 차이의 함수인 가우시안 필터를 하나 더 사용합니다. 공간의 가우시안 함수는 주변 화소만 블러링을 위해 고려되는 반면, 강도 차이의 가우시안 함수는 중앙 화소와 유사한 강도를 가진 화소만 블러링을 고려하도록 합니다. 따라서 경계의 화소는 강도 변화가 크기 때문에 경계를 유지합니다.

cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace[, borderType]) -> dst

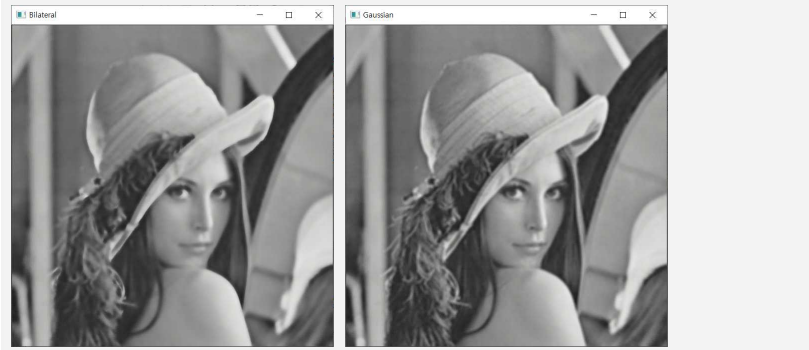
- src - 8비트 1채널 또는 3채널 입력 이미지입니다.
- d - 필터링 시 고려할 주변 화소의 지름입니다.
- sigmaColor - 색 공간에서 시그마를 필터링합니다. 매개 변수의 값이 클수록 픽셀 주변 내에서 더 멀리 떨어진 색상이 함께 혼합되어 반균등 색상의 더 큰 영역이 형성된다는 것을 의미합니다.
- sigmaSpace - 좌표 공간에서 시그마를 필터링합니다. 좌표 공간에서 시그마를 필터링합니다. 매개 변수의 값이 클수록 픽셀의 색상이 충분히 가까울 경우 더 먼 픽셀이 서로 영향을 미친다는 것을 의미합니다. d>0일 경우 sigmaSpace에 관계없이 인접 크기를 지정합니다. 그렇지 않으면 d는 sigmaSpace에 비례합니다.
- * 단순화를 위해 2개의 시그마 값을 동일하게 설정할 수 있습니다. 크

기가 작으면(<10) 필터 효과가 많지 않은 반면, 크면(>150) 매우 강한 필터 효과를 줍니다.

- borderType - 이미지 외부의 픽셀을 외삽하는데 사용되는 테두리 모드입니다. cv2.BORDER_XXXX 형식으로 정의되어 있습니다.

다음 코드는 cv2.GaussianBlur()와 cv2.bilateralFilter() 함수의 블러링 효과를 비교합니다.

```
import cv2

lena_gray = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow("Bilateral", cv2.bilateralFilter(lena_gray, 9, 75, 75))
cv2.imshow("Gaussian", cv2.GaussianBlur(lena_gray, (5, 5), 75))
cv2.waitKey()
cv2.destroyAllWindows()
```



- 두 함수의 시그마 값을 바꿔보면서 그 결과가 어떻게 달라지는지 느껴보세요.

양방향 필터에 대한 이론적인 지식은 아래의 주소를 참고하세요.
https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html

2절. 모폴로지 연산

모폴로지(Morphology) 연산은 형태학적 변환을 의미합니다. 그래서 이 절에서 설명할 연산은 수축(또는 수축(Erosion))과 팽창(Dilation)입니다. 수축과 팽창이 이미지에 어떤 영향을 주는지 그리고 수축 후 팽창 또는 팽창 후 수축이 어떤 영향을 주는지에 대해 설명합니다.

2.1. 수축

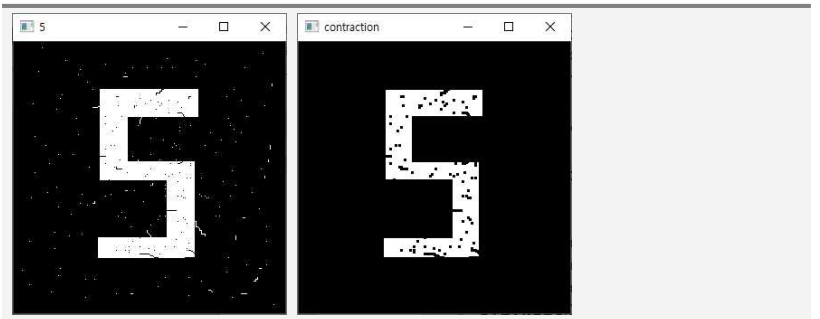
수축은 화소의 부근에 하나라도 0이면 그 화소를 0으로, 그 외는 255로 처리합니다. 수축 후 물체의 두께 또는 크기가 감소하거나 단순히 흰색 영역이 이미지에서 감소합니다. 수축은 작은 백색 잡음을 제거하고 두 개의 연결된 개체를 분리하는 데 유용합니다.

다음 코드는 수축을 구현한 함수입니다.

```
1  import numpy as np
2
3  def contraction(img_in):
4      height, width = img_in.shape
5      img_out = np.zeros(img_in.shape, np.uint8)
6      for y in range(1, height-1):
7          for x in range(1, width-1):
8              img_out[y,x] = img_in[y,x]
9              if img_in[y-1,x-1]==0 ¦ img_in[y-1,x]==0 ¦ \
                  img_in[y-1,x+1]==0 ¦ img_in[y,x-1]==0 ¦ \
                  img_in[y,x+1]==0 ¦ img_in[y+1,x-1]==0 ¦ \
                  img_in[y+1,x]==0 ¦ img_in[y+1,x+1]==0:
10                 img_out[y,x] = 0
11     return img_out
```

다음 코드는 수축을 구현한 함수를 적용합니다.

```
import cv2

img5 = cv2.imread("five.png", cv2.IMREAD_GRAYSCALE)
img5_cont = contraction(img5)

cv2.imshow("5", img5)
cv2.imshow("수축", img5_cont)
cv2.waitKey()
cv2.destroyAllWindows()
```



2.2. 팽창

팽창은 수축의 반대입니다. 화소의 부근에 하나라도 255이면 그 화소를 255로, 그 외는 0으로 처리합니다. 따라서 이미지의 흰색 영역이 증가하거나 전경 물체의 크기가 증가합니다.

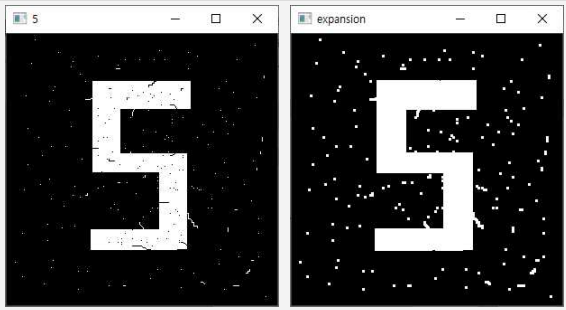
수축은 백색 잡음을 제거하지만 물체를 축소시키기 때문에 확장을 적용합니다. 잡음은 없어졌기 때문에 다시 돌아오지는 않지만 물체 영역은 증가합니다. 일반적으로 잡음 제거와 같은 경우 수축 후에 팽창을 수행합니다. 팽창은 개체의 끊어진 부분을 결합하는 데 유용합니다.

다음은 팽창을 구현한 함수입니다.

```
import numpy as np

def expansion(img_in):
    height, width = img_in.shape
    img_out = np.zeros(img_in.shape, np.uint8)
    for y in range(1, height-1):
        for x in range(1, width-1):
            img_out[y,x] = img_in[y,x]
            if img_in[y-1,x-1]==255 ¦ img_in[y-1,x  ]==255 ¦ \
               img_in[y-1,x+1]==255 ¦ img_in[y  ,x-1]==255 ¦ \
               img_in[y  ,x+1]==255 ¦ img_in[y+1,x-1]==255 ¦ \
               img_in[y+1,x  ]==255 ¦ img_in[y+1,x+1]==255:
                img_out[y,x] = 255
    return img_out
```

다음은 팽창을 구현한 함수를 적용합니다.

```
import cv2

img5 = cv2.imread("five.png", cv2.IMREAD_GRAYSCALE)
img5_exp = expansion(img5)

cv2.imshow("5", img5)
cv2.imshow("팽창", img5_exp)
cv2.waitKey()
cv2.destroyAllWindows()
```

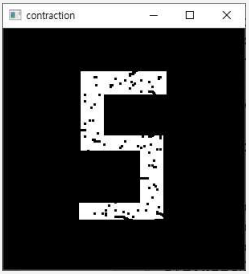


2.3. 열기(Opening)

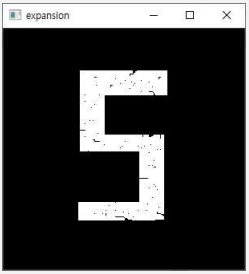
Opening(열기 또는 개방)은 수축 후 팽창의 다른 이름입니다. 수축 후 팽창을 하면 검은 바탕의 흰 잡음을 없애줍니다.

```
import cv2

img5 = cv2.imread("five.png", cv2.IMREAD_GRAYSCALE)

img5_cont = contraction(img5)
img5_exp = expansion(img5_cont)

cv2.imshow("5", img5)
cv2.imshow("contraction", img5_cont)
cv2.imshow("expansion", img5_exp)
cv2.waitKey()
cv2.destroyAllWindows()
```

잡음을 포함한 원본



수축

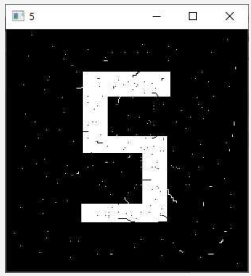


수축 후 팽창

2.4. 닫기(Closing)

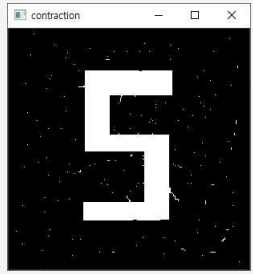
닫기(Closing)는 팽창 후 수축의 다른 이름입니다. 이것은 전경 개체 내부의 작은 구멍이나 개체의 작은 검은 점을 닫을 때 유용합니다.

```
import cv2

img5 = cv2.imread("five.png", cv2.IMREAD_GRAYSCALE)

img5_exp = expansion(img5)
img5_cont = contraction(img5_exp)

cv2.imshow("5", img5)
cv2.imshow("expansion", img5_exp)
cv2.imshow("contraction", img5_exp)
cv2.waitKey()
cv2.destroyAllWindows()
```



잡음을 포함한 원본

팽창



팽창 후 수축

2.5. cv2.erode()

cv2.erode()는 수축을 위한 OpenCV의 함수입니다.

cv2.erode(src, dst, kernel, anchor=(-1,-1), iteration=1)

- src - 입력 이미지입니다. 채널 수는 임의로 지정할 수 있지만 깊이는 CV_8U, CV_16U, CV_16S, CV_32F 또는 CV_64F 중 하나여야 합니다.
- dst - src와 동일한 크기 및 타입의 출력 이미지입니다.
- kernel - 수축에 사용되는 필터커널입니다.
- anchor - 요소 내 현재 화소의 위치입니다. 기본값 (-1,-1)은 현재 화소가 필터 커널의 중심에 있음을 의미합니다.
- iteration - 수축이 적용되는 횟수입니다.

다음 코드는 cv2.erode() 함수를 테스트합니다.

```
import cv2
import numpy as np

img = cv2.imread('j.png',0)
kernel = np.ones((5,5), np.uint8)
cv2.imshow("j", img)
cv2.imshow("erode", cv2.erode(img, kernel , iterations=1))
cv2.waitKey()
cv2.destroyAllWindows()
```

원본 수축

2.6. cv2.dilate()

cv2.dilate() 함수는 팽창을 위한 OpenCV의 함수입니다.

cv2.dilate(src, dst, kernel, anchor=(-1,-1), iteration=1)

- src - 입력 이미지입니다. 채널 수는 임의로 지정할 수 있지만 깊이는 CV_8U, CV_16U, CV_16S, CV_32F 또는 CV_64F 중 하나여야 합니다.
- dst - src와 동일한 크기 및 타입의 출력 이미지입니다.
- kernel - 팽창에 사용되는 필터커널입니다.
- anchor - 요소 내 현재 화소의 위치입니다. 기본값 (-1,-1)은 현재 화소가 필터 커널의 중심에 있음을 의미합니다.
- iteration - 팽창이 적용되는 횟수입니다.

다음 코드는 cv2.dilate() 함수를 테스트합니다.

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread('j.png',0)
5  kernel = np.ones((5,5), np.uint8)
6  cv2.imshow("j", img)
7  cv2.imshow("dilate", cv2.dilate(img, kernel , iterations=1))
8  cv2.waitKey()
9  cv2.destroyAllWindows()
```

원본 팽창

2.7. cv2.morphologyEx()

cv2.morphologyEx()는 수축 후 팽창과 팽창 후 수축을 위한 함수입니다.

cv2.morphologyEx(src, dst, op, kernel, anchor=(-1,-1), iteration=1)

- src - 입력 이미지입니다. 채널 수는 임의로 지정할 수 있지만 깊이는 CV_8U, CV_16U, CV_16S, CV_32F 또는 CV_64F 중 하나여야 합니다.
- dst - src와 동일한 크기 및 타입의 출력 이미지입니다.
- op - 모폴로지 타입입니다. 아래 표를 참고하세요.
- kernel - 수축에 사용되는 필터커널입니다.
- anchor - 요소 내 현재 화소의 위치입니다. 기본값 (-1,-1)은 현재 화소가 필터 커널의 중심에 있음을 의미합니다.
- iteration - 수축이 적용되는 횟수입니다.

다음 표는 cv2.morphologyEx() 함수에서 사용할 수 있는 모폴로지 타입입니다.

표 19. 모폴로지 타입

모폴로지 타입	설명
MORPH_ERODE	수축
MORPH_DILATE	팽창
MORPH_OPEN	수축 후 팽창(열기)
MORPH_CLOSE	팽창 후 수축(닫기)
MORPH_GRADIENT	팽창과 수축의 차이(a morphological gradient)
MORPH_TOPHAT	입력 이미지와 열기의 차이(top hat)
MORPH_BLACKHAT	수축과 입력 이미지의 차이(black hat)
MORPH_HITMISS	CV_8UC1 바이너리 이미지만 지원합니다. Hit-or-Miss 변환은 이진 이미지에서 패턴을 찾는 데 유용합니다.

수축 후 팽창은 노이즈를 제거하는 데 유용합니다.

```
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

팽창 후 수축은 전경 개체 내부의 작은 구멍이나 개체의 작은 검은 점을 닫을 때 유용합니다.

```
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

그래디언트는 이미지의 팽창과 수축의 차이입니다.

```
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
```

탑햇은 입력 이미지와 이미지 열기의 차이입니다. 아래 예제는 9x9 커널에 대해 수행됩니다.

```
tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
```

블랙햇은 수축과 입력 이미지의 차이입니다.

```
blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)
```

히트미스 변환은 이진 이미지에서 패턴을 찾는 데 유용합니다.

위의 이미지는 위/오른쪽 끝을 찾는 필터 커널을 적용한 결과입니다.

```
kernel = np.array([[0,-1,-1],[1,1,-1],[0,1,0]], np.uint8)
```

0	0	0
1	1	0
0	1	0

0	-1	-1
0	0	-1
0	0	0

0	-1	-1
1	1	-1
0	1	0

그림 4. (커널). 왼쪽: '적중'할 커널. 중간: '미스'할 커널. 오른쪽: 최종 결합 커널

히트미스 변환에 사용한 위의 필터는 현재 화소의 왼쪽과 아래가 전경에 속하고 위, 오른쪽, 위/오른쪽이 배경에 속하는 패턴을 찾고 있습니다. 이웃에 있는 나머지 픽셀은 어떤 종류이든 상관없습니다. 이 패턴을 위/오른쪽 모서리를 찾습니다.

3절. 화질 개선

이미지를 더 잘 인지할 수 있도록 하는 것은 영상처리에서 매우 중요한 일입니다. 다음 이미지는 어두운 이미지입니다. 이 이미지를 더 보기 쉬운 이미지로 바꿀 수 있는 몇 가지 방법이 있습니다.

그림 5. 어두운 이미지

3.1. 히스토그램과 밝기 조절

이미지의 화질을 개선하기 위해서 히스토그램 분포를 확인하고 수정하는 작업이 필요합니다. 이 절에서는 히스토그램 분포를 변경하여 이미지의 화질을 개선하는 방법을 설명합니다.

1) 히스토그램

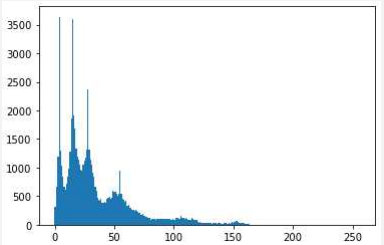
이미지 히스토그램은 이미지의 각 화소가 이미지 내에 얼마나 분포하고 있는지를 알려줍니다. 만일 8비트 그레이스케일 이미지라면 화소의 값은 0부터 255까지 값을 갖게 됩니다. 히스토그램은 0부터 255까지 값에 해당하는 화소가 몇 개인지를 알려줍니다.

다음 코드는 이미지의 히스토그램을 계산하는 함수입니다.

```
import numpy as np

def histogram(img):
    height, width = img.shape
    hist = np.zeros(256)
    for y in range(height):
        for x in range(width):
            hist[img[y,x]] = hist[img[y,x]] + 1
    return hist
```

다음 코드는 히스토그램을 막대그래프로 표시합니다.

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
plt.bar(x=range(256), height=histogram(img), width=1)
plt.show()
```



막대그래프의 왼쪽에 많은 화소가 분포되어 있다는 것은 전체적으로 낮은 화소의 개수가 많다는 것을 의미합니다. 화소 값이 0에 가까울수록 어두운 화소이므로 이 히스토그램을 통해서 알 수 있는 것은 이미지가 대체로 어두운 이미지라는 것입니다.

2) 히스토그램 밝기 조절

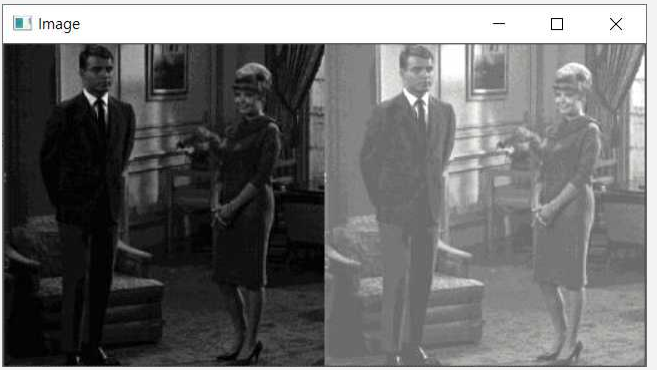
화소의 값을 전체적으로 크게 하면 더 밝은 이미지를 얻을 수 있습니다. 반대로 화소의 값을 전체적으로 작게 하면 더 어두운 이미지를 얻을 수 있습니다.

다음 함수는 이미지의 밝기를 조절하는 함수입니다. 기존의 화소에 threshold만큼 더해서 밝기를 조절할 수 있습니다. threshold가 음수이면 더 어두워집니다.

```
1  import numpy as np
2
3  def brightness(img, threshold=0):
4      height, width = img.shape
5      img_ = np.zeros(img.shape, dtype=np.uint8)
6
7      for y in range(height):
8          for x in range(width):
9              temp = img[y][x] + threshold;
10             if (temp > 255):
11                 img_[y][x] = 255
12             elif(temp < 0):
13                 img_[y][x] = 0
14             else:
15                 img_[y][x] = temp
16     return img_
```

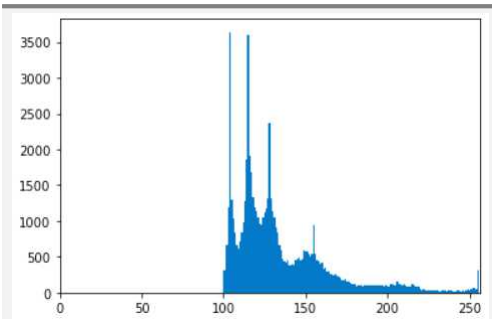
다음 함수는 밝기를 조절하는 함수를 적용합니다.

```
1  img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
2  img2 = brightness(img, threshold=100)
3  cv2.imshow("Image", np.c_[img, img2])
4  cv2.waitKey()
5  cv2.destroyAllWindows()
```



다음 코드는 밝기를 변경한 후 히스토그램을 출력합니다.

```
import matplotlib.pyplot as plt

img2 = brightness(img, threshold=100)
plt.bar(x=list(range(256)), height=histogram(img2), width=1)
plt.xlim([0,256])
```

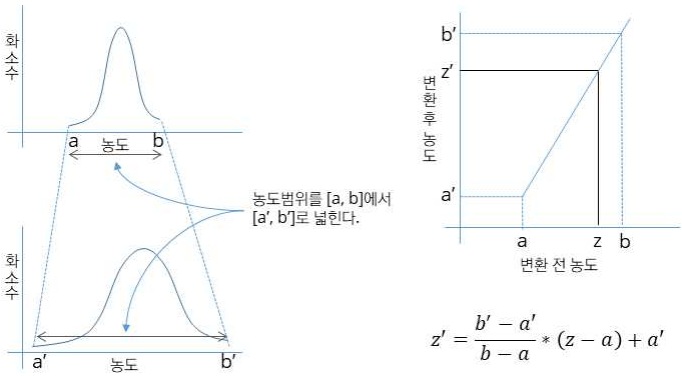


결과에서 보듯이 히스토그램이 전체적으로 오른쪽으로 100만큼 옮겨진 것을 확인할 수 있습니다. 그런데 단순히 밝기만 바꾸는 것으로 선명한 이미지를 얻을 수 없습니다. 다음은 콘트라스트 강조에 대해 설명합니다.

3.2. 콘트라스트 강조

콘트라스트 강조는 명암의 대비를 극대화시켜 선명한 영상을 만드는 것입니다. 이것은 입력 영상의 임의 구간을 출력영상의 정해진 다른 구간에 매핑시킵니다. 그래서 정규화(Normalize)라고 합니다.

다음 그림에서 보듯이 입력 이미지의 농도 구간 a ~ b를 출력 이미지의 새로운 구간 a' ~ b'가 되도록 하는 방법입니다. 만일 화소의 농도가 0부터 255 사이로 표현된다면 출력 이미지의 구간은 0 ~ 255입니다.



$$z' = \frac{b' - a'}{b - a} * (z - a) + a'$$

그림 6. 콘트라스트 강조

화소의 농도 구간을 0부터 255 사이로 사용한다면 콘트라스트 강조를 위한 출력 이미지의 화소 농도값은 다음 식을 통해 구할 수 있습니다.

$$output = \frac{255 - 0}{high - low} * (x - low) + 0$$

다음 코드는 콘트라스트 강조를 구현한 함수입니다.

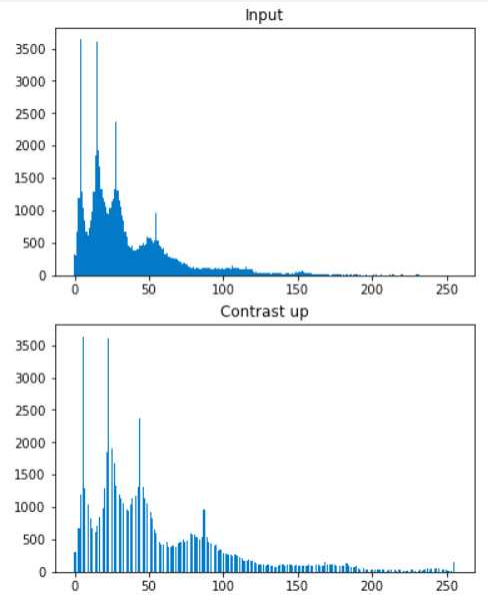
```
import numpy as np

def contrast(img, low=0, high=255):
    height, width = img.shape
    img_ = np.zeros(img.shape, dtype=np.uint8)

    for y in range(height):
        for x in range(width):
            temp = (int)((255/(high-low)) * (img[y][x]-low))
            if (temp > 255):
                img_[y][x] = 255
            elif(temp < 0):
                img_[y][x] = 0
            else:
                img_[y][x] = temp
    return img_
```

다음 코드는 콘트라스트 강조를 적용합니다.

```
img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
img2 = contrast(img, 0, 160)
cv2.imshow("Image", np.c_[img, img2])
cv2.waitKey()
cv2.destroyAllWindows()
```

다음 코드는 콘트라스트 강조 전/후 히스토그램을 비교합니다.

```
import matplotlib.pyplot as plt
import cv2

img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
img2 = contrast(img, 0, 160)

plt.figure(figsize=(6,8))
plt.subplot(211)
plt.bar(x=range(256), height=histogram(img), width=1)
plt.title("Input")

plt.subplot(212)
plt.bar(x=range(256), height=histogram(img2), width=1)
plt.title("Contrast up")
plt.show()
```



3.3. cv2.normalize()

cv2.normalize()는 콘트라스트 강조를 구현한 함수입니다.

cv2.normalize(src, dst, alpha, beta, flag) -> dst

- src - 입력 이미지입니다.
- dst - src와 동일한 크기 및 타입의 출력 이미지입니다.
- alpha - 정규화하기 위한 구간 1입니다. 이미지 콘트라스트 강조 시 0을 사용합니다.
- beta - 정규화하기 위한 구간 2입니다. 이미지 콘트라스트 강조 시 255를 사용합니다.
- flag - 알고리즘을 선택할 플래그입니다. 사용 가능한 플래그는 다음과 같습니다.
  cv2.NORM_MINMAX : alpha와 beta 구간으로 정규화합니다.
  cv2.NORM_L1 : 전체 합으로 나눕니다.
  cv2.NORM_L2 : 단위벡터로 정규화합니다.
  cv2.NORM_INF : 최댓값으로 나눕니다.

다음 코드는 cv2.normalize()를 테스트합니다. 이 함수가 올바르게 실행되려면 입력 이미지에서 정규화 구간외 농도를 갖는 화소는 있으며 안됩니다. 5~8라인은 160보다 큰 화소는 모두 160으로 바꿉니다.

```
1  import cv2
2
3  img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
4  height, width = img.shape
5  for y in range(height):
6      for x in range(width):
7          if img[y,x] > 160:
8              img[y,x] = 160
9
10 img2 = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX)
11 cv2.imshow("Image", np.c_[img, img2])
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```

* 이 코드의 실행결과로 만들어진 이미지의 히스토그램도 그려보세요.

3.4. 히스토그램 이퀄라이제이션

이미지의 히스토그램이 특정 영역에 너무 집중되어 있으면 콘트라스트가 낮아 좋은 이미지라고 할 수 없습니다. 화소의 값들이 히스토그램 전체 영역에 골고루 분포되어 있을 때 좋은 이미지라고 할 수 있습니다.

히스토그램 이퀄라이제이션의 이론적인 방법은 이미지의 각 픽셀의 누적분포함수(CDF; Cumulative Distribution Function)값을 구하고 히스토그램 균일화 공식에 대입하여 0~255 사이의 값으로 변환을 하게 됩니다. 이렇게 새롭게 구해진 값으로 이미지를 표현하면 균일화된 이미지를 얻을 수 있습니다.

히스토그램 이퀄라이제이션은 히스토그램 균등화, 히스토그램 평탄화, 또는 히스토그램 평활화 등으로 불립니다. 언뜻 보면 이미지의 모든 농도가 같은 개수의 히스토그램을 가질 것처럼 보이지만 사실을 그렇지 않습니다. 이 책에서 히스토그램 이퀄라이제이션은 히스토그램 균일화라는 이름으로 사용하고 히스토그램 평탄화는 모든 농도가 같은 개수의 히스토그램을 갖는 완전하게 평평한 히스토그램 이퀄라이제이션을 의미합니다.

1) 히스토그램 이퀄라이제이션 구현

히스토그램 이퀄라이제이션의 구현은 아래의 방법으로 합니다.

- 입력 이미지의 히스토그램을 계산하고 정규화된 합을 계산하기 위해 스케일 팩터(Scale Factor)를 구합니다.
  scale_factor = 255 / (width*height)
- 각 농도의 히스토그램 누적합을 계산합니다.
- (x,y) 위치의 농도가 히스토그램 누적합의 농도가 되도록 합니다.

다음 코드는 히스토그램 이퀄라이제이션을 구현한 함수입니다.

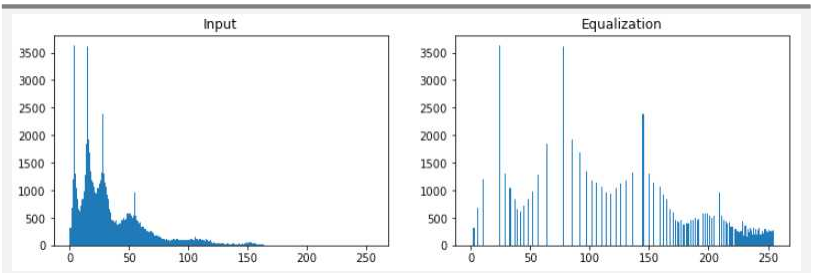
```
import numpy as np

def equalization(img):
    height, width = img.shape
    img_in = img.copy()
    img_out = np.zeros(img_in.shape, dtype=np.uint8)
    histogram = np.zeros(256, dtype=int)
    sum_histo = np.zeros(256, dtype=int)

    for y in range(height):
        for x in range(width):
            histogram[img_in[y,x]] = histogram[img_in[y,x]] + 1

    sum = 0
    scale_factor =  255 / (width * height)
    for i in range(256):
        sum = sum + histogram[i]
        sum_histo[i] = np.ceil(sum*scale_factor)

    for y in range(height):
        for x in range(width):
            img_out[y,x] = sum_histo[img_in[y][x]]

    return img_out
```

다음 코드는 구현된 eqalization() 함수를 테스트합니다.

```
import cv2

img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
img2 = equalization(img)
cv2.imshow("Image", np.c_[img, img2])
cv2.waitKey()
cv2.destroyAllWindows()
```

다음 코드는 이퀄라이제이션 전/후 히스토그램을 비교합니다.

```
import matplotlib.pyplot as plt
import cv2

img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
img2 = equalization(img)

plt.figure(figsize=(12,3.5))
plt.subplot(121)
plt.bar(x=range(256), height=histogram(img), width=1)
plt.title("Input")
plt.subplot(122)
plt.bar(x=range(256), height=histogram(img2), width=1)
plt.title("Equalization")
plt.show()
```



2) cv2.equalizeHist()

OpenCV의 cv2.equalizeHist() 함수는 히스토그램 이퀄라이제이션을 구현한 함수입니다.

cv2.equalizeHist(sec[, dst]) -> dst

- src - 8비트 단일채널(그래이스케일) 입력 이미지입니다.
- dst - src와 동일한 크기 및 타입의 출력 이미지입니다.

이 함수는 다음 알고리즘을 사용하여 그레이스케일 이미지의 히스토그램을 평활화합니다. 알고리즘은 밝기를 정규화하고 이미지의 대비를 높입니다.

① src 에 대한 히스토그램 H를 계산합니다.

② 히스토그램 빈의 합계가 255가 되도록 히스토그램을 정규화합니다.

③ 히스토그램의 적분을 계산합니다.

$$H_i' = \sum_{0 \le j < i} H(j)$$

④ H_i'를 조회 테이블로 사용하여 영상을 변환합니다.

$$\mathrm{dst}(x,y) = H'(\mathrm{src}(x,y))$$

다음 코드는 equalizeHist() 함수를 테스트합니다.

```
1  import cv2
2
3  img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
4
5  img2 = cv2.equalizeHist(img)
6
7  cv2.imshow("Image", np.c_[img, img2])
8  cv2.waitKey()
9  cv2.destroyAllWindows()
```

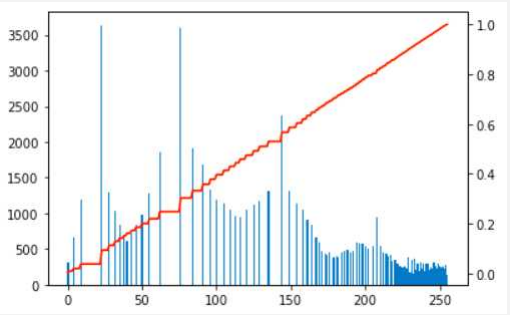
* 이 함수를 이용한 실행결과는 equalize() 함수의 결과와 같습니다.

3) 누적 히스토그램 그래프

히스토그램 이퀄라이제이션이라고 부르는 이유는 이미지의 농도에 따른 화소 개수의 누적 비율을 그래프로 나타내면 아래의 코드를 실행시킨 것처럼 우상향으로 화소의 개수가 균등하게 증가하는 그래프를 보이기 때문입니다.

다음 코드는 이퀄라이제이션 이미지의 히스토그램과 히스토그램 누적 비율을 그래프로 나타냅니다.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
img2 = cv2.equalizeHist(img)

hist = histogram(img2)

cumsum_hist = np.cumsum(hist)
sum_hist = np.sum(hist)
plt.bar(range(256), hist, width=1)
ax = plt.twinx()
ax.plot(range(256), cumsum_hist/sum_hist, 'r')
plt.show()
```



3.5. 적응적 히스토그램 이퀄라이제이션

앞에서의 히스토그램 이퀄라이제이션은 이미지의 전체적인 부분에 대한 적용이었습니다. 기존의 방법은 밝은 부분과 어두운 부분이 섞여 있는 이미지에 적용하는 것은 유용하지 않습니다.

1) CLAHE

적응적 히스토그램 이퀄라이제이션(AHE; Adaptive histogram equalization)은 각각 이미지의 개별 영역에 해당하는 여러 히스토그램을 계산하고 이를 사용하여 이미지의 밝기를 재분배한다는 점에서 일반적인 히스토그램 이퀄라이제이션과 다릅니다. 따라서 로컬 대비를 개선하고 이미지의 각 영역에서 엣지의 정의를 향상시키는데 적합합니다.

AHE는 이미지의 비교적 균일한 영역에서 잡음(Noise)을 과도하게 증폭(콘트라스트 강조)하는 경향이 있습니다. 그래서 CLAHE(Contrast Limited Adaptive Histogram Equalization)라고 하는 적응형 히스토그램 이퀄라이제이션의 변형은 콘트라스트를 제한하여 이를 방지합니다.

CLAHE에서 주어진 화소 값 근처의 대비 증폭은 변환 함수의 기울기에 의해 제공됩니다. 이는 인접 누적 분포 함수(CDF[7])의 기울기에 비례하므로 해당 화소 값에서 히스토그램값에 비례합니다. CLAHE는 CDF를 계산하기 전에 미리 정의된 값에서 히스토그램을 클리핑하여 증폭을 제한합니다. 이것은 CDF의 기울기와 따라서 변환 함수의 기울기를 제한합니다. 히스토그램이 클리핑되는 값인 소위 클립 제한은

7) 누적 분포 함수에 대한 자세한 설명을 원한다면 아래 주소를 참고하세요.
https://en.wikipedia.org/wiki/Cumulative_distribution_function

히스토그램의 정규화와 이에 따른 인접 영역의 크기에 따라 달라집니다. 일반적인 값은 결과 증폭을 3과 4 사이로 제한합니다.

클립 제한을 초과하는 히스토그램 부분(위 그림의 왼쪽에서 위쪽 부분)은 버리지 않고 모든 히스토그램 빈에 균등하게 재배포하는 것이 좋습니다.

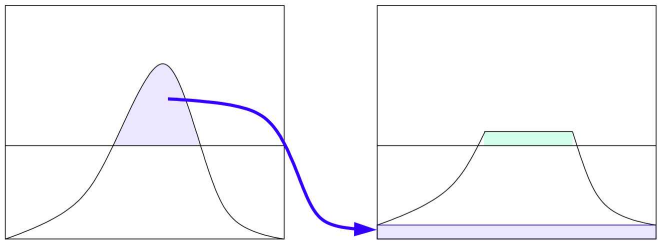


그림 7. CLAHE
https://commons.wikimedia.org/wiki/File:Clahe-redist.svg

재배포는 일부 빈을 다시 클립 제한 (그림에서 오른쪽 클립 제한을 넘어선 영역; 녹색으로 음영 처리된 영역)을 넘어서서 규정된 제한보다 크고 정확한 값이 이미지에 따라 달라지는 효과적인 제한을 생성합니다.

2) cv2.createCLAHE()

cv2.createCLAHE()는 CLAHE를 구현한 OpenCV의 함수입니다.

```
cv2.createCLAHE(clipLimit=40.0, tileGridSize=(8,8))
```

- clipLimit : 대비 제한을 위한 임계값입니다.
- tileGridSize : 히스토그램 이퀄라이제이션을 위한 그리드 크기입니다. 입력 이미지는 동일한 크기의 직사각형 타일로 분할됩니다. tileGridSize는 행과 열의 타일 수를 정의합니다.

다음 그림을 예제로 사용하겠습니다.

그림 8. CLAHE에 사용할 이미지(tsukuba.png)
Kolmogorov and Zabih's Graph Cuts Stereo Matching Algorithm
http://www.ipol.im/pub/art/2014/97/?utm_source=doi

다음 그림은 예제에 사용할 이미지를 그레이스케일 이미지로 불러온 후 히스토그램입니다.

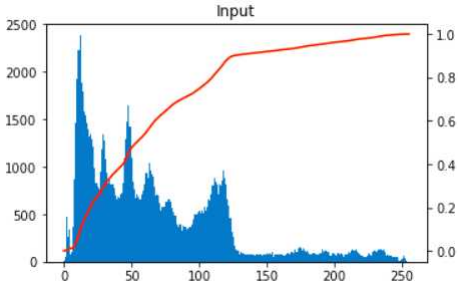


그림 9. CLAHE에 사용할 이미지의 히스토그램

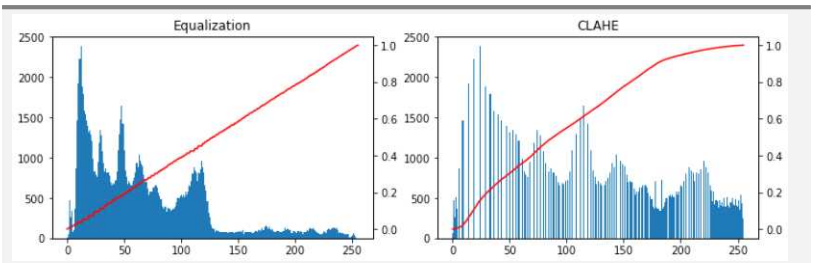
일반적인 히스토그램 이퀄라이제이션과 CLAHE를 비교하기 위해서 cv2.equalizeHist() 함수와 createCLAHE() 함수에 의한 결과를 출력합니다.

```
1  import cv2
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  img = cv2.imread("tsukuba.png", cv2.IMREAD_GRAYSCALE)
6
7  img2 = cv2.equalizeHist(img)
8  clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
9  img3 = clahe.apply(img)
10
11 cv2.imshow('Equalization vs. CLAHE', np.c_[img2, img3])
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```

다음 코드는 히스토그램 이퀄라이제이션과 CLAHE를 적용한 이미지의 히스토그램을 출력합니다. 위의 코드에서 9라인까지는 같은 내용이므로 복사해 사용해도 됩니다.

```
1  import cv2
2  import matplotlib.pyplot as plt
3  import numpy as np
4
```

```
 5  img = cv2.imread("tsukuba.png", cv2.IMREAD_GRAYSCALE)
 6
 7  img2 = cv2.equalizeHist(img)
 8  clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
 9  img3 = clahe.apply(img)
10
11  hist1 = histogram(img2)
12  cumsum_hist1 = np.cumsum(hist1)
13  sum_hist1 = np.sum(hist1)
14
15  hist2 = histogram(img3)
16  cumsum_hist2 = np.cumsum(hist2)
17  sum_hist2 = np.sum(hist2)
18
19  plt.figure(figsize=(12,3.5))
20  plt.subplot(121)
21  plt.bar(x=range(256), height=histogram(img), width=1)
22  ax = plt.twinx()
23  ax.plot(range(256), cumsum_hist1/sum_hist1, 'r')
24  plt.title("Equalization")
25
26  plt.subplot(122)
27  plt.bar(x=range(256), height=histogram(img2), width=1)
28  ax = plt.twinx()
29  ax.plot(range(256), cumsum_hist2/sum_hist2, 'r')
30  plt.title("CLAHE")
31  plt.show()
```



3.6. 완벽하게 평평한 히스토그램 이퀄라이제이션

완벽하게 평평한 히스토그램 이퀄라이제이션(여기서는 히스토그램 평탄화라고 부름)은 아래 그림의 오른쪽처럼 모든 화소의 값들이 히스토그램 전체 영역에 완전히 균등하게 분포하도록 하는 방법입니다.

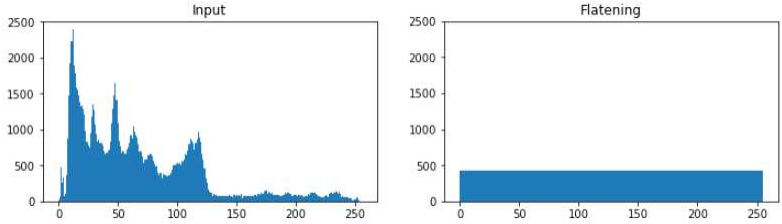


그림 10. 히스토그램 균일화 전(왼쪽), 후(오른쪽) 히스토그램

1) 히스토그램 평탄화 알고리즘

다음 그림은 화소의 농노레벨을 8단계(0~7)로 정하고 화소의 수는 40개라고 가정하고 각 농도에 따른 화소의 개수를 구한 히스토그램이 어떻게 균일화 과정으로 되는지 설명하기 위한 것입니다.

농도 레벨	7	6	5	4	3	2	1	0
원화상의 각 레벨의 화소수	0	4	9	11	5	7	4	0
	0 4	1 5	3 2	5 4	1 4	1 5	1 4	
평탄화 후의 각 레벨의 화소수	5	5	5	5	5	5	5	5

그림 11. 히스토그램 평탄화

먼저 처음 히스토그램은 농도 4와 5에 가장 많이 분포되어 있고 7과 0에는 분포되어 있지 않습니다. 이것이 평탄화 과정을 거치려면 다음 단계를 따릅니다.

전체 화소의 수가 40개이고 농도 레벨이 8이므로 평탄화한 후 농도별 화소의 수는 5개가 되어야 합니다. 그래서 농도 7부터 5개 화소가 될 때까지 입력 이미지의 가장 큰 농도를 갖는 화소들부터 차례대로 출력 이미지의 농도 레벨로 바꿉니다.

출력 이미지의 농도 7이 되어야 하는 화소는 입력 이미지에서 농도 6의 화소의 수는 4개와 입력 이미지 농도 5에 해당하는 화소 중 1개입니다. 입력 이미지 농도 7화소는 0개이므로 계산에서 제외합니다.

여기에서 입력 이미지의 농도 5화소 9개 중에서 1개만 출력 이미지의 농도 7로 바꿔야 합니다. 다시 말하면 입력 이미지 농도 5화소 9개 중에서 1개만 출력 이미지 농도 7이 되도록 해야 한다는 의미입니다. 이때 선택의 기준이 주위(현재 화소를 둘러싼 8개) 화소의 농도입니다. 입력 이미지의 농도 5화소 9개 중에서 주위 화소들의 총합이 가장 큰 값을 갖는 화소 중 1개를 출력 이미지 농도 7로 바꿉니다.

출력 이미지의 농도 7에 해당하는 화소들이 선택되면 다음은 출력 이미지의 농도 6에 해당하는 화소들을 선택해야 합니다. 앞서 입력 이미지의 농도 5에 해당하는 화소 9개 중에서 1개 화소는 출력 이미지의 농도 7이 되었으므로 나머지 8개 중에서 주위 화소 총합이 가장 큰 5개를 출력 이미지의 농도 6이 되게 합니다.

입력 이미지의 농도 5화소들 중에서 남은 3개는 출력 이미지의 농도 5가 되도록 합니다. 그리고 출력 이미지의 농도 5가 되어야 하는 나머지 화소 2개는 입력 이미지의 농도 4에 해당하는 화소 11개 중에서 주위 화소의 총합이 가장 큰 2개를 선택합니다.

출력 이미지의 농도 4가 되는 화소는 입력 이미지 농도 4 화소 11개 중에서 2개는 이미 출력 이미지 농도 5에 할당하였으므로 남은 9개 중에서 주위 화소 총합이 큰 상위 5개여야 합니다.

출력 이미지의 농도 3이 되는 화소는 입력 이미지 농도 4 화소 중 남은 4개를 모두 할당합니다. 그리고 출력 이미지 농도 3이 되어야 하는 화소 1개는 입력 이미지 농도 3화소 5개 중에서 주위 화소 총합이 가장 큰 1개를 선택합니다.

출력 이미지의 농도 2가 되어야 하는 화소는 입력 이미지 농도 3화소 5개 중에서 1개는 출력 이미지 농도 3이 되도록 할당하고 남은 4개와 입력 이미지의 농담 2에 해당하는 화소 7개 중에서 주위 화소 총합이 가장 큰 1개입니다.

출력 이미지의 농도 1이 되어야 하는 화소는 입력 이미지의 농도 2화소 7개 중 출력 이미지의 농도 2에 할당한 1개를 제외한 6개 주에서 주위 화소 총합이 큰 5개여야 합니다.

마지막으로 출력 이미지의 농도 0이 되어야 하는 화소는 입력 이미지의 농도 2인 화소 중 남은 1개와 입력 이미지의 농도 1에 해당하는 화소 4개입니다.

히스토그램 평탄화의 결과는 밝은 이미지나 어두운 이미지 어떤 것을 사용해도 같은 결과가 나옵니다. 이것은 이미지의 인식을 할 때 유용합니다.

완벽하게 평평한 히스토그램 이퀄라이제이션의 구현은 일반 히스토그램 이퀄라이제이션에 비해 알고리즘이 복잡하고 실행시간도 많이 소요됩니다. 다음에 설명할 코드를 모두 이해할 필요는 없습니다. 이러한 알고리즘이 있다는 것만 알아두셔도 됩니다.

2) 히스토그램 평탄화 구현

flattening() 함수를 작성하기 전에 주위 8화소의 총합(코드에서는 가중치(weight)로 표현)을 구하는 함수와 이를 이용해 정렬하는 함수, 그리고 정렬된 화소의 위치와 총합을 저장하는 클래스를 먼저 구현해야 합니다.

다음 코드는 현재 화소의 위치와 그 화소의 주위 8화소 총합을 저장하는 클래스입니다.

```
class SortedPixel():
    def __init__(self, x, y, weight):
        self.x = x
        self.y = y
        self.weight = weight
```

다음 코드는 현재 화소 주위 8화소의 총합을 구합니다.

```
import numpy as np

def get_weight(pixels, x, y):
    weight = 0
    xm = x-1
    ym = y-1
    xp = x+1
    yp = y+1
    d = np.zeros(8, dtype=int)
    height, width = pixels.shape

    if (xm < 0): xm = x
    if (ym < 0): ym = y
    if (xp >= width): xp = x
    if (yp >= height): yp = y

```

```
    d[0]=pixels[ym, xm];  d[1]=pixels[ym, x];   d[2]=pixels[ym, xp]
    d[3]=pixels[y,  xm];                        d[4]=pixels[y,  xp]
    d[5]=pixels[yp, xm];  d[6]=pixels[yp, x];   d[7]=pixels[yp, xp]

    weight = np.sum(d)

    return weight;
```

다음 코드는 주위 8개 화소의 총합을 이용해 정렬하는 함수입니다. 이 코드에서 임포트한 operator.attrgetter() 함수는 SortPixel 객체의 weight 속성을 정렬 기준이 되는 키로 지정하기 위해 사용했습니다.

```
from operator import attrgetter

def sort_by_weight(pixels, level, count):
    height, width = pixels.shape
    data = []
    inum = 0
    weight = 0

    for y in range(height):
        for x in range(width):
            try:
                if(pixels[y,x]==level):
                    w = get_weight(pixels, x, y)
                    data.append(SortedPixel(x, y, w))
                    inum = inum + 1
            except:
                pass

    data.sort(key=attrgetter('weight'))

    return data
```

다음 코드는 히스토그램 평탄화를 구현한 함수입니다.

```
def flattening(img):
    height, width = img.shape
    img_in = img.copy()
    img_out = np.zeros(img.shape, dtype=np.uint8)
    histogram = np.zeros(256, dtype=int)

    for y in range(height):
        for x in range(width):
            histogram[img_in[y,x]] = histogram[img_in[y,x]] + 1
    low, high, delta = 255, 255, 0
    avg_pixel_count = (int)((height*width)/256)

    for i in range(255,-1,-1) :
        sum = 0
        while (sum < avg_pixel_count):
            sum = sum + histogram[low]
            low = low - 1

        low = low + 1
        delta = histogram[low]-(sum-avg_pixel_count)
        buffer = sort_by_weight(img_in, low, histogram[low])

        if (low<high):
            for y in range(height):
                for x in range(width):
                    if(img_in[y,x]>=low+1) & (img_in[y,x]<=high) :
                        img_out[y,x] = i

        for j in range(delta):
            img_out[int(buffer[j].y), int(buffer[j].x)] = i
            img_in[int(buffer[j].y), int(buffer[j].x)] = low+1

        histogram[low] = histogram[low] - delta
        high = low

    return img_out
```

다음 코드는 히스토그램 평탄화를 테스트합니다. 실행결과의 왼쪽은 원본 이미지이고 오른쪽은 히스토그램 평탄화 결과입니다. 이 코드는 컴퓨터 사양에 따라 수십초 이상 소요될 수 있습니다.

```
import cv2

img = cv2.imread("tsukuba.png", cv2.IMREAD_GRAYSCALE)
img2 = flattening(img)
cv2.imshow("Image flattening", np.c_[img, img2])
cv2.waitKey()
cv2.destroyAllWindows()
```

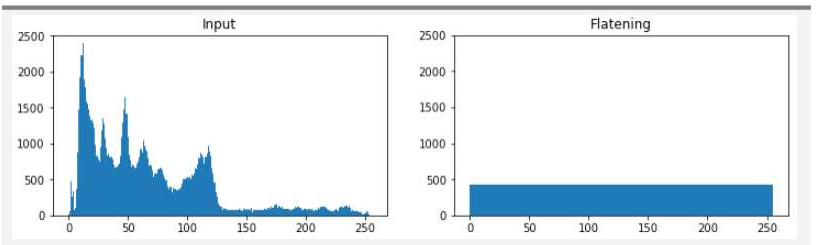
이 예제는 히스토그램 평탄화를 이용해서 이미지의 해상도를 개선시킬 수 있는 것을 보여주는 예입니다. 이 알고리즘이 가장 빠르거나 결과 이미지의 해상도가 가장 좋은 것은 아닙니다. 여러분이 이 예제를 통해 히스토그램과 이미지 화질 개선의 개념을 이해하도록 포함했습니다.

OpenCV의 히스토그램 관련 함수는 이 알고리즘과는 다르게 구현되어 있으며 변환 후 히스토그램이 모두 같은 값을 갖지 않습니다.

3) 실행 전/후 히스토그램

다음 코드는 히스토그램 평탄화 전/후 히스토그램을 막대 그래프로 표현합니다. 오른쪽 이미지의 히스토그램은 모든 농도가 같은 개수임을 알 수 있습니다.

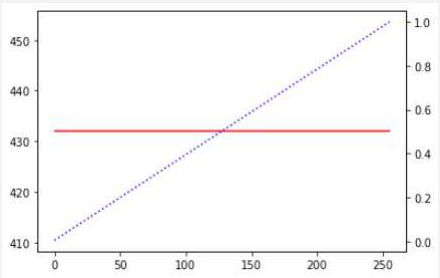
```
1  import matplotlib.pyplot as plt
2  import cv2
3
4  img = cv2.imread("tsukuba.png", cv2.IMREAD_GRAYSCALE)
5  img2 = flattening(img)
6
7  hist1 = cv2.calcHist(images=[img], channels=[0], mask=None,
   histSize=[256], ranges=[0,256]) # 원본 이미지의 히스토그램
8  hist2 = cv2.calcHist(images=[img2], channels=[0], mask=None,
   histSize=[256], ranges=[0,256]) # 평탄화 이미지의 히스토그램
9
10 plt.figure(figsize=(12,3))
11 plt.subplot(121)
12 plt.bar(x=range(256), height=hist1.flatten(), width=1)
13 plt.title("Input")
14 y_lim = plt.ylim() # 첫 번째 그래프의 y축을 저장
15
16 plt.subplot(122)
17 plt.bar(x=range(256), height=hist2.flatten(), width=1)
18 plt.xlim(y_lim)    # 첫 번째 그래프의 y축 범위와 같게 지정
19 plt.title("flattening")
20 plt.show()
```



4) 누적 히스토그램

다음 코드는 히스토그램 평탄화 후 히스토그램과 농도별 누적합 비율을 출력합니다.

```
import cv2

img = cv2.imread("tsukuba.png", cv2.IMREAD_GRAYSCALE)
img2 = flattening(img)

hist2 = cv2.calcHist(images=[img2], channels=[0], mask=None,
                     histSize=[256], ranges=[0,256])

cumsum_hist = np.cumsum(hist2)
sum_hist = np.sum(hist2)
plt.plot(range(256), hist2, 'r')
ax = plt.twinx()
ax.plot(range(256), cumsum_hist/sum_hist, 'b:')
plt.show()
```



실행결과에서 빨간 실선(수평선)은 히스토그램 분포이고 파란 점선(대각선)은 누적합의 비율입니다.

다음 그림은 원본 이미지와 완벽하게 평평한 히스토그램 평활화에 의한 결과 그리고 CLAHE 방법에 의한 결과입니다. 이 이미지는 석고상의 얼굴 영역에 있어서 CLAHE 알고리즘이 더 좋은 결과를 얻는다는 것을 알 수 있습니다.

그림 12. 원본

그림 13. 완벽하게 평평한 히스토그램 평활화

그림 14. CLAHE

완벽하게 평평한 히스토그램 평활화는 복잡한 알고리즘과 느린 실행 속도, 그리고 히스토그램 평활화가 가지고 있는 국소적인 부분에서의 극단적인 밝기 변화로 인해 실제 많이 사용되고 있지는 않습니다. 여러분은 이 책에서 언급한 완벽하게 평평한 히스토그램 평활화는 영상 처리의 한 방법이라는 것만 기억하세요.

6장. 객체 특징 추출

1절. 라벨링과 특징 파라미터

영상처리에서 라벨링(Labeling)은 이미지에서 각각의 물체를 찾아 특정한 값을 부여하는 것입니다.

1.1. 라벨링

라벨링(Labeling)은 연결된 모든 화소에 같은 번호를 붙이고 다른 연결 성분에는 다른 번호를 붙입니다. 라벨링을 할때 고려해야 할 부분은 크기 두 가지가 입니다.

- 인접하여 연결된 화소들을 찾는 것이므로 화소의 인접성(Neighborhood) 4근방으로 할 것인가? 8근방으로 할 것인가?
- 연결된 화소들을 찾아 같은 숫자로 표시할 때의 알고리즘은 어떤 것을 사용할 것인가?

첫 번째 고려해야 할 사항은 현재 주목 화소의 상/하/좌/우 4근방 화소를 관심 화소로 할지 아니면 현재 주목 화소의 주위 8근방 화소를 주목 화소로 할지에 따라 추출되는 물체가 달라질 수 있습니다. 4근방은 상/하/좌/우만 찾지만 8근방은 대각선 영역의 화소까지 찾아줍니다.

두 번째 고려해야 할 사항은 알고리즘입니다. 라벨링 알고리즘은 Grass-Fire와 2-Pass[8] 알고리즘이 있습니다. 그리고 이 알고리즘 내에서 인접요소가 모두 라벨링 될 때까지 재귀 호출(Recursive Call)로 검사해서 라벨링을 구현하는 방법과, 라벨링 정보를 저장하고 반환할 수 있는 스택(Stack)을 이용하는 방법이 있습니다.

8) 호센 코플맨(Hoshen-Kopelman) 알고리즘이라고도 합니다.
Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm. J. Hoshen and R. Kopelman. Phys. Rev. B 14, 3438 – Published 15 October 1976

다음은 라벨링 알고리즘을 설명합니다.

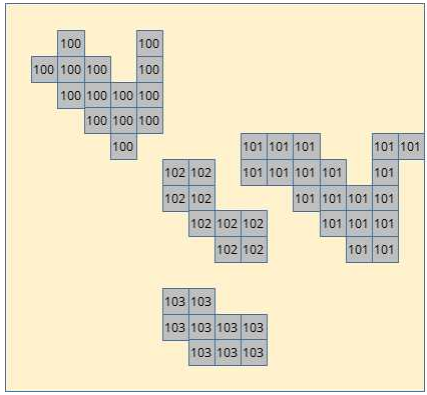


그림 1. 라벨링할 예제

앞의 그림에서 100번으로 라벨링 된 첫 번째 객체가 어떻게 라벨링 되는지 다음 그림에서 보여주고 있습니다.

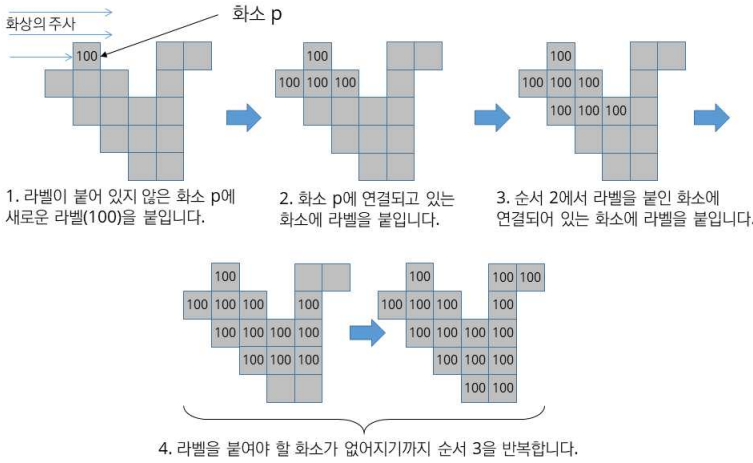


그림 2. 라벨링 단계

라벨링은 다음 단계에 따라 이뤄집니다.

1) 영상의 처리는 왼쪽에서 오른쪽으로 그리고 위에서 아래로 화소를 읽으면서 처리합니다. 가장 먼저 마주친 화소에 번호를 부여합니다. 이 예에서는 100부터 부여했습니다.
2) 그리고 현재 화소(p)에 연결되고 있는 주위 화소에 같은 라벨을 붙입니다.
3) 2)에서 라벨을 붙인 화소에 연결되어 있는 화소에 라벨을 붙입니다.
4) 라벨을 붙여야 할 화소가 없어질 때까지 3)을 반복합니다.

다음 코드들은 라벨링을 구현한 함수입니다. labelset() 함수는 현재 주목 화소가 라벨이 있으면 연결되어 있는 주위 8근방 화소에 같은 라벨을 붙여줍니다.

```
def labelset(img, xs, ys, label):
    height, width = img.shape
    img[ys, xs] = label
    while True:
        cnt = 0
        for y in range(1, height-1):
            for x in range(1, width-1):
                if img[y,x] == label :
                    if  img[y  , x+1] == 255 :
                        img[y  , x+1] = label; cnt = cnt+1
                    if  img[y-1, x+1] == 255 :
                        img[y-1, x+1] = label; cnt = cnt+1
                    if  img[y-1, x  ] == 255 :
                        img[y-1, x  ] = label; cnt = cnt+1
                    if  img[y-1, x-1] == 255 :
                        img[y-1, x-1] = label; cnt = cnt+1
                    if  img[y  , x-1] == 255 :
                        img[y  , x-1] = label; cnt = cnt+1
                    if  img[y+1, x-1] == 255 :
                        img[y+1, x-1] = label; cnt = cnt+1
                    if  img[y+1, x  ] == 255 :
```

```
                        img[y+1, x  ] = label; cnt = cnt+1
                    if  img[y+1, x+1] == 255 :
                        img[y+1, x+1] = label; cnt = cnt+1
        if cnt==0:
            return (0, img)
    return (1, img)
```

다음 함수는 모든 화소에 대해 라벨링을 수행합니다.

```
def labeling(img_in, base=100):
    height, width = img_in.shape
    img_label = img_in.copy()

    label = base # 베이스값부터 라벨링 함

    for y in range(1, height-1):
        for x in range(1, width-1):
            if img_label[y,x] == 255 :
                if label >= 255:
                    print("Error! too many labels")
                    return -1

                _, img_label = labelset(img_label, x, y, label)
                label = label + 1
    cnt = label - base

    return img_label, cnt
```

다음 코드는 준비된 이미지를 이용해서 라벨링을 수행합니다. 원본 이미지는 2진화한 후 라벨링을 수행합니다. 라벨링 후 검출된 객체 수와 원, A, 세모, 네모 안의 임의 좌표를 지정해서 해당 객체의 레이블을 확인해 봅니다. 라벨링 후 각 객체의 레이블은 100, 101, 102, 103으로 라벨링된 것을 확인할 수 있습니다. 라벨의 베이스를 100으

로 주었기 때문에 100부터 라벨링 된 것입니다.

```
1  import cv2
2
3  img = cv2.imread('shape.png', 0)
4  _, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
5
6  labeled_img, cnt = labeling(bin_img)
7
8  print("객체 수", cnt)
9  print(labeled_img[80, 80], end=" ")
10 print(labeled_img[69,173], end=" ")
11 print(labeled_img[160,190], end=" ")
12 print(labeled_img[160,60])
13
14 cv2.imshow('image', img)
15 cv2.imshow('Labeling', labeled_img)
16 cv2.waitKey()
17 cv2.destroyAllWindows()
```

```
객체 수 4
100 101 102 103
```

1.2. cv2.connectedComponents()

OpenCV의 cv2.connectedComponents() 함수를 이용하면 쉽게 라벨링할 수 있습니다.

cv2.connectedComponents(img, connectivity=8) -> cnt, labels

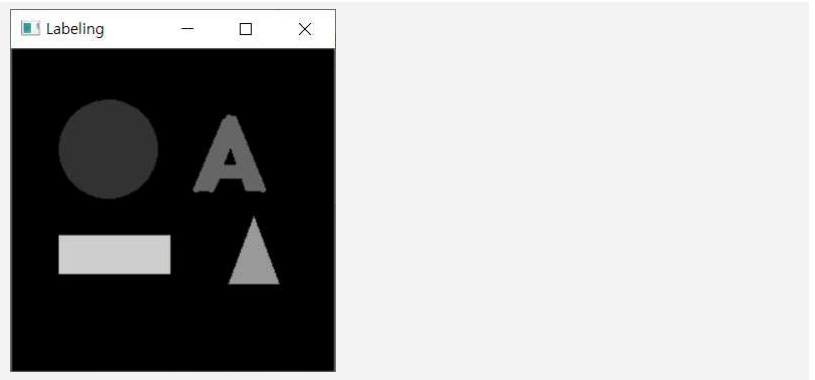
- img - 8비트 단일채널(그래이스케일) 이미지입니다.
- connectivity - 연결성 4 또는 8을 지정합니다.

- 반환값 cnt, labels - cnt는 라벨의 개수이며, labels는 라벨링된 출력 이미지입니다. 출력 라벨 이미지의 타입은 CV_32S(부호 있는 32비트)이며, ltype 매개변수를 이용해 CV_16U 타입으로 지정 가능합니다.

다음 코드는 cv2.connectedComponents() 함수를 이용하여 라벨링하는 예입니다. 라벨은 1부터 지정됩니다. 라벨링된 결과를 더 잘 볼 수 있도록 라벨에 50씩 곱한 결과로 이미지를 출력했습니다.

```
1  import cv2
2
3  img = cv2.imread('shape.png', 0)
4  _, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
5  cnt, labeled_img = cv2.connectedComponents(bin_img)
6
7  print(labeled_img[80, 80], end=" ")
8  print(labeled_img[69, 173], end=" ")
9  print(labeled_img[160, 190], end=" ")
10 print(labeled_img.dtype)
11
12 labeled_img = labeled_img * 50
13 cv2.imshow('Labeling', labeled_img.astype(np.uint8))
14 cv2.waitKey()
15 cv2.destroyAllWindows()
```

```
1 2 3 4
int32
```



1.3. 특징 파라미터

객체의 면적, 주위 길이, 원형도, 중심 위치 등 특징 파라미터들에 대해서 알아보겠습니다.

이미지를 읽어 2진화한 후 라벨링을 수행하고 이를 기반으로 특징 파라미터들을 알 수 있습니다.

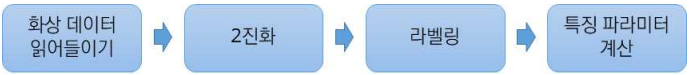


그림 3. 특징 파라미터 계산 절차

1) 면적과 중심

면적은 물체에 포함된 화소 수를 계산합니다. 객체의 면적은 1화소가 면적 1에 해당하는데 같은 레이블의 수를 세면 해당 레이블을 갖는 객의 면적이 계산됩니다.

중심은 중심을 계산할 객체에 포함되는 화소의 위치 (x_i, y_i) $(i = 0, 1, \ldots, n-1)$의 평균값을 계산합니다.

$$\left(\frac{1}{n}\sum_{i=0}^{n-1} x_i,\ \frac{1}{n}\sum_{i=0}^{n-1} y_i\right)$$

다음 함수는 객체의 면적과 중심을 계산합니다.

```
def calc_size(img_label, label):
    tx, ty = 0, 0
    cx, cy = 0, 0 # 반환할 중심점
    total = 0
    height, width = img_label.shape
    for y in range(height):
        for x in range(width):
            if img_label[y,x]==label:
                tx = tx + x
                ty = ty + y
                total = total + 1
    if total==0:
        return 0
    cx = int(tx/total)
    cy = int(ty/total)
    return (cx, cy, total)
```

다음 코드는 앞에서 만든 함수를 테스트합니다.

```
import cv2

img = cv2.imread('shape.png', 0)
_, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
labeled_img, cnt = labeling(bin_img, base=100)
for i in range(cnt):
    cx, cy, area = calc_size(labeled_img, i+100)
    print(cx, cy, area)
```

```
76 79 4880
127 87 1797
192 168 1108
81 163 2759
```

2) 주위 길이

객체의 주위 길이를 계산할 때 수직 수평선분의 경계는 1 화소당 1로 계산됩니다. 그러나 대각선 성분의 경계는 $\sqrt{2}$가 되므로 수평성분의 경계와 수직 성분의 경계 길이를 다르게 계산해야 합니다.

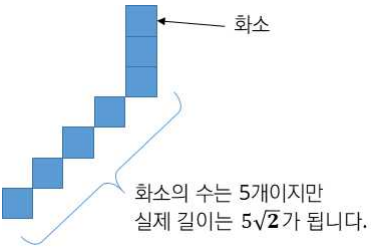


그림 4. 주위 길이 계산

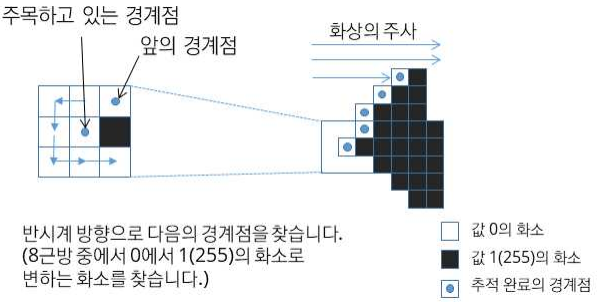


그림 5. 경계부의 추적

주위 길이의 계산은 처음 만나는 경계점으로부터 반시계 방향으로 다음의 경계선을 찾아가는데 주위 8근방 화소를 이전 경계점에서 반시계방향으로 탐색을 하면서 화소의 값이 0에서 255로 변하는 화소를 찾아갑니다.

다음 그림은 주위 길이를 계산하기 위해 처음 만나는 경계점을 현재 주목화소의 오른쪽(3시방향)에 두었을 때 주위 화소를 3시방향이 0이 되게합니다.

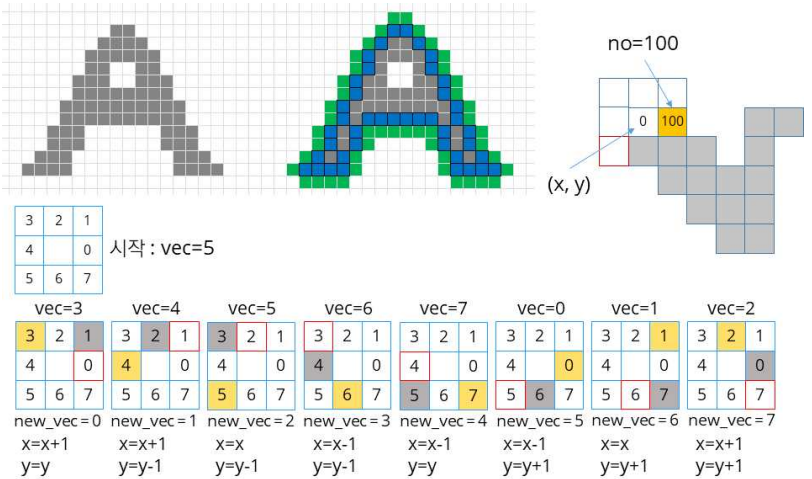


그림 6. 주위 길이 계산

다음 코드는 주위 길이를 계산하는 함수입니다. 앞의 그림을 구현한 코드입니다.

```
import numpy as np

def trace(img_label, xs, ys):
    length = 0
    x, y = xs, ys
    no = img_label[y, x+1]
```

```
    vec = 5

    while True:
        if (x==xs) & (y==ys) & (length!=0):
            return (length, img_label)
        img_label[y,x] = 255

        if vec==3:
            if (img_label[y,x+1]!=no) & (img_label[y-1,x+1]==no):
                x=x+1; y=y;
                length=length+1
                vec=0
                continue
            else:
                vec=4
        if vec==4:
            if (img_label[y-1,x+1]!=no) & (img_label[y-1,x]==no):
                x=x+1; y=y-1;
                length=length+np.sqrt(2)
                vec=1
                continue
            else:
                vec=5
        if vec==5:
            if (img_label[y-1,x]!=no) & (img_label[y-1,x-1]==no):
                x=x; y=y-1;
                length=length+1
                vec=2
                continue
            else:
                vec=6
        if vec==6:
            if (img_label[y-1,x-1]!=no) & (img_label[y,x-1]==no):
                x=x-1; y=y-1;
                length=length+np.sqrt(2)
                vec=3
                continue
```

```
44              else:
45                  vec=7
46          if vec==7:
47              if (img_label[y,x-1]!=no) & (img_label[y+1,x-1]==no):
48                  x=x-1; y=y;
49                  length=length+1
50                  vec=4
51                  continue
52              else:
53                  vec=0
54          if vec==0:
55              if (img_label[y+1,x-1]!=no) & (img_label[y+1,x]==no):
56                  x=x-1; y=y+1;
57                  length=length+np.sqrt(2)
58                  vec=5
59                  continue
60              else:
61                  vec=1
62          if vec==1:
63              if (img_label[y+1,x]!=no) & (img_label[y+1,x+1]==no):
64                  x=x; y=y+1;
65                  length=length+1
66                  vec=6
67                  continue
68              else:
69                  vec=2
70          if vec==2:
71              if (img_label[y+1,x+1]!=no) & (img_label[y,x+1]==no):
72                  x=x+1; y=y+1;
73                  length=length+np.sqrt(2)
74                  vec=7
75                  continue
76              else:
77                  vec=3
```

다음 코드는 모든 화소에 대해 주위 길이를 계산하고 객체의 테두리가 표시된 이미지를 반환합니다.

```
def calc_length(img_label, label):
    height, width = img_label.shape
    for y in range(height):
        for x in range(width):
            if img_label[y,x] == label:
                length, img_label = trace(img_label, x-1, y)
                return length, img_label
```

* 앞에서 정의된 두 함수는 다음 원형도를 설명하고 테스트합니다.

3) 원형도

면적, 원주 길이를 토대로 형상의 복잡한 정도를 측정하는 특징량입니다. 원형도(e)는 다음 식을 통해 얻을 수 있습니다. 원의 경우, 원주 길이 $2\pi r$, 면적 πr^2이므로, e=1.0이 됩니다. 도형이 복잡할수록 작은 값이 됩니다.

$$e = 4\pi * \text{면적}/(2*\text{주위길이})$$

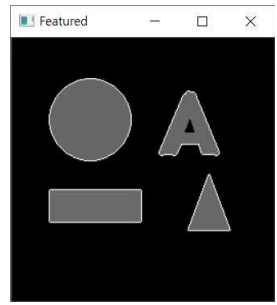
다음 코드는 앞에서 만들었던 면적과 중심을 계산하는 함수, 주위 길이를 계산하는 함수를 실행시키고 면적, 중심, 주위 길이, 원형도, 그리고 라벨링되고 주위 길이가 표시된 이미지 객체를 반환하는 함수입니다.

```
import numpy as np

def features(img_label, cnt):
    areas = []; centers = []; lengths = []; ratios = []
    featured_img = np.zeros(img_label.shape, dtype=np.uint8)
    for i in range(cnt):
        cx, cy, area = calc_size(img_label, i+100)
```

```
        centers.append((cx, cy)); areas.append(area)
        length, featured_img = calc_length(img_label, i+100)
        lengths.append(length)
        ratio = 4*np.pi*area / (length*length)
        ratios.append(ratio)

    return ((areas, centers, lengths, ratios), featured_img)
```

다음 코드는 라벨링과 특징 추출을 테스트합니다. 코드에서 feat 변수는 순서대로 객체의 면적, 중심점, 주위 길이, 원형도입니다.

```
import cv2

img = cv2.imread('shape.png', 0)
_, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
labeled_img, cnt = labeling(bin_img, 100)
feat, featured_img = features(labeled_img, cnt)
areas, centers, lengths, ratios = feat

print(areas)
print(centers)
print(lengths)
print(ratios)

cv2.imshow('image', featured_img)
cv2.waitKey()
cv2.destroyAllWindows()
```




```
[4880, 1797, 1108, 2759]
[(76, 79), (172, 87), (192, 168), (81, 163)]
[263.7645019878176, 226.99494936611694, 168.2253967444163,
241.6568542494924]
[0.8814486001080429, 0.4382536609116099, 0.49200158512447256,
0.5936949146124189]
```

1.4. OpenCV를 이용한 특징 추출

OpenCV의 함수를 이용하면 영역의 경계선을 연결한 선을 찾아 그리거나 객체의 특징 파라미터를 쉽게 얻을 수 있습니다.

1) cv2.findContours()와 cv2.drawContours()

cv2.findContours(), cv2.drawContours() 함수를 이용하면 동일한 색 또는 동일한 강도를 가지고 있는 영역의 경계선을 찾고 표시할 수 있습니다.

cv2.findContours() 함수는 경계를 찾아줍니다.

cv2.findContours(image, mode, method) -> image, contours, hierarchy

- image – 8비트 단일채널 이미지(바이너리 이미지)입니다.
- mode – 경계(Contour)를 찾는 방법입니다.
  - cv2.RETR_EXTERNAL : 경계선 중 가장 바깥쪽 선만 찾음.
  - cv2.RETR_LIST : 모든 경계선을 찾지만, 계층(Hierarchy) 관계를 구성하지 않음.
  - cv2.RETR_CCOMP : 모든 경계선을 찾으며, 계층 관계는 2레벨로 구성함.
  - cv2.RETR_TREE : 모든 경계선을 찾으며, 모든 계층 관계를 구성함.
- method – Contour를 찾을 때 사용하는 근사치 방법입니다.
  - cv2.CHAIN_APPROX_NONE : 모든 경계점을 저장.
  - cv2.CHAIN_APPROX_SIMPLE : 경계선을 그릴 수 있는 점만 저장.(ex; 사각형이면 4개 포인트)
  - cv2.CHAIN_APPROX_TC89_L1 : 경계점을 찾는 알고리즘
  - cv2.CHAIN_APPROX_TC89_KCOS : 경계점을 찾는 알고리즘

- 반환값 image, contours, hierarchy – 이미지 객체와 경계들, 그리고 계층 구조를 반환합니다.

cv2.drawContours()는 찾은 컨투어 정보를 이용해서 경계선을 그린 객체를 반환합니다.

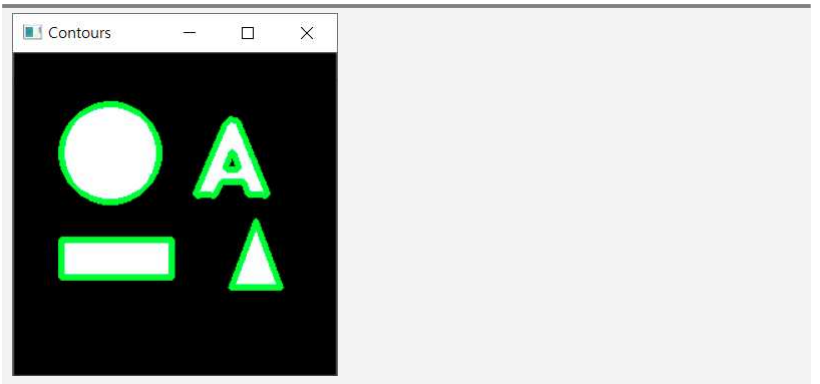
cv2.drawContours(image, contours, contourIdx, color, thickness) -> dst

- image - 원본 이미지입니다.
- contours - 경계(Contour) 정보입니다.
- contourIdx - 경계 리스트에서 몇 번째 경계선을 그릴 것인지 지정합니다. -1이면 전체를 그립니다.
- color - 경계선의 색을 지정합니다.
- thickness - 경계선의 두께를 지정합니다. 음수이면 경계선의 내부를 채웁니다.

- 반환값 dst - 이미지에 경계가 그려진 결과 이미지 객체를 반환합니다.

다음 코드는 cv2.findContours(), cv2.drawContours() 함수를 테스트합니다.

```
1  import cv2
2
3  img = cv2.imread('shape.png')
4  imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
5
6  ret, bin_img  = cv2.threshold(imgray,127,255,0)
7
8  contours, hierachy = cv2.findContours(bin_img,
                                         cv2.RETR_TREE,
                                         cv2.CHAIN_APPROX_SIMPLE)
9
10 image = cv2.drawContours(img, contours, -1, (0,255,0), 3)
11
12 cv2.imshow('image', image)
13 cv2.waitKey()
14 cv2.destroyAllWindows()
```



2) cv2.moments()

이미지 모멘트(Moment)는 대상을 구분할 수 있는 특징(Area, Perimeter, 중심점)을 의미합니다. 대상을 구분한 후, 다른 대상과 구분하기 위해 대상을 설명(describe)하는 자료로 사용되기도 합니다.

cv2.moments() 함수는 객체의 여러 특징을 추출해 줍니다.

cv2.moments(array, binaryImage) -> retval

- ▶ array - 8비트 단일채널 이미지 또는 1xN 또는 Nx1 2D 점들입니다.
- ▶ binaryImage - True이면 0이 아닌 모든 이미지 픽셀은 1로 처리됩니다. 매개변수는 이미지에만 사용됩니다.

▶ 반환값 retval - dict_item 유형의 모멘트 정보를 반환합니다.

모멘트는 크게 3가지 정보를 포함합니다.

- m_{ji} : 공간 모멘트이며 다음에 의해 계산됩니다.

$$m_{ji} = \sum_{x,y}(array(x,y) \cdot x^j \cdot y^i)$$

- mu_{ji} : 중심 모멘트이며 다음에 의해 계산됩니다.

$$\mu_{ji} = \sum_{x,y}(array(x,y) \cdot (x-\bar{x})^j \cdot (y-\bar{y})^i)$$

$$\bar{x} = \frac{m_{10}}{m_{00}}, \ \bar{y} = \frac{m_{01}}{m_{00}}$$

- nu_{ji} : 정규화된 중심 모멘트이며 다음에 의해 계산됩니다.

$$nu_{ji} = \frac{mu_{ji}}{m_{00}^{(i+j)/2+1}}$$

3) cv2.contourArea()

cv2.contourArea()는 컨투어 영역을 계산합니다. 모멘트의 결과인 dict_items의 m00과 같습니다.

cv2.contourArea(contour, oriented) -> retval

- contour - 컨투어 객체입니다.
- oriented - True이면 함수는 윤곽선 방향(시계 방향 또는 시계 반대 방향)에 따라 부호 있는 영역 값을 반환합니다. 이 기능을 사용하면 영역의 부호를 사용하여 윤곽선의 방향을 결정할 수 있습니다. 기본값은 False이며 절댓값이 리턴됨을 의미합니다.

- 반환값 retval - 컨투어 영역입니다. 모멘트의 결과인 dict_items의 m00과 같습니다.

4) cv2.arcLength()

cv2.arcLength()는 둘레 길이를 구합니다.

cv2.arcLength(curve, closed) -> retval

- curve - 2D 점의 입력 벡터입니다.

- close – True이면 폐곡선 도형의 둘레 길이를 구하고, False 이면 시작점과 끝 점을 연결하지 않고 둘레 길이를 구합니다.

- 반환값 retval – 둘레 길이입니다.

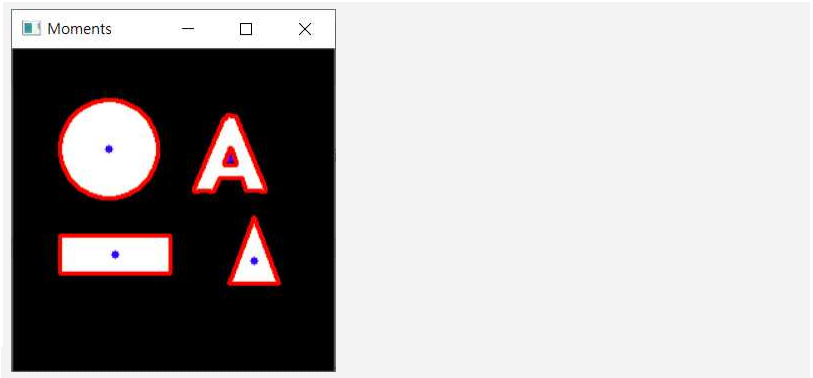
cv2.arcLength(cont, True)는 폐곡선 도형을 만들어 둘레 길이를 구하며, cv2.arcLength(cont, False)는 시작점과 끝점을 연결하지 않고 둘레 길이를 구합니다.

다음 코드는 cv2.moments(), cv2.contourArea(), cv2.arcLength() 함수를 테스트합니다.

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread('shape.png')
5  imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
6  ret, bin_img = cv2.threshold(imgray,127,255,0)
7
8  contours, hierarchy = cv2.findContours(bin_img,
                                          cv2.RETR_TREE,
                                          cv2.CHAIN_APPROX_SIMPLE)
9
10 for cont in contours:
11     M = cv2.moments(cont)
12     print(M.items())
13     print(cv2.contourArea(cont))
14     print(cv2.arcLength(cont, True)
15
16     center_x = int(M['m10'] / M['m00'])
17     center_y = int(M['m01'] / M['m00'])
18     cv2.circle(img, (center_x, center_y), 3, (255, 0, 0), -1)
19     cv2.drawContours(img, [cont], 0, (0, 0, 255), 2)
20
21 cv2.imshow('Moments', img)
22 cv2.waitKey()
23 cv2.destroyAllWindows()
```

```
dict_items([('m00', 2640.0), ('m10', 213840.0), ('m01', 430320.0),
('m20', 19024720.0), ('m11', 34855920.0), ('m02', 70340160.0),
('m30', 1816998480.0), ('m21', 3101029360.0), ('m12',
5697552960.0), ('m03', 11529994080.0), ('mu20', 1703680.0),
('mu11', 0.0), ('mu02', 198000.0), ('mu30', 0.0), ('mu21', 0.0),
('mu12', 0.0), ('mu03', 0.0), ('nu20', 0.24444444444444446),
('nu11', 0.0), ('nu02', 0.02840909090909091), ('nu30', 0.0),
('nu21', 0.0), ('nu12', 0.0), ('nu03', 0.0)])
2640.0
236.0
dict_items([('m00', 1034.0), ('m10', 198528.0), ('m01',
174221.3333333333), ('m20', 38184238.0), ('m11', 33450496.0),
('m02', 29513652.333333332), ('m30', 7357048704.0), ('m21',
6434229694.833333), ('m12', 5666621248.0), ('m03', 5025178420.0),
('mu20', 66862.0), ('mu11', 3.725290298461914e-09), ('mu02',
158649.44315495715), ('mu30', 0.0), ('mu21', 468711.58639460243),
('mu12', -1.1920928955078125e-06), ('mu03', -1115676.79678154),
('nu20', 0.0625371788588382), ('nu11', 3.4843281040951126e-15),
('nu02', 0.14838755350478056), ('nu30', 0.0), ('nu21',
0.013633405192293172), ('nu12', -3.467438387074438e-14), ('nu03',
-0.03245167022894321)])
1034.0
162.56854152679443
dict_items([('m00', 1768.0), ('m10', 305196.5), ('m01', 155029.0),
('m20', 52964544.83333333), ('m11', 26755447.833333332), ('m02',
13991422.333333332), ('m30', 9239798139.25), ('m21',
4645066860.316667), ('m12', 2414167327.5833335), ('m03',
1294375620.7), ('mu20', 280775.8218796998), ('mu11',
-6038.704279795289), ('mu02', 397536.11104826257), ('mu30',
-8027.87162399292), ('mu21', 2898060.4606841505), ('mu12',
-7320.3711812496185), ('mu03', -2194212.1245110035), ('nu20',
0.08982457805676196), ('nu11', -0.0019318759724780699), ('nu02',
```

```
0.12717802123480487), ('nu30', -6.107940070894423e-05), ('nu21',
0.022049654559477404), ('nu12', -5.5696441804255144e-05), ('nu03',
-0.016694482407127943)])
1768.0
221.33809351921082
dict_items([('m00', 86.0), ('m10', 14887.0), ('m01',
7466.666666666666), ('m20', 2577449.0), ('m11',
1292483.6666666665), ('m02', 649313.0), ('m30', 446320691.5),
('m21', 223768503.36666667), ('m12', 112393755.5), ('m03',
56554003.6), ('mu20', 440.05813953466713), ('mu11',
-31.062015504110605), ('mu02', 1044.2661498708185), ('mu30',
-70.75635474920273), ('mu21', 739.4959258552699), ('mu12',
48.85385497435345), ('mu03', -1788.3516054749489), ('nu20',
0.05949947803335141), ('nu11', -0.004199839846418416), ('nu02',
0.1411933680193102), ('nu30', -0.0010316189770650094), ('nu21',
0.010781759932073825), ('nu12', 0.0007122832157332513), ('nu03',
-0.026073947144562615)])
86.0
40.97056245803833
dict_items([('m00', 4769.0), ('m10', 362652.5), ('m01', 376915.0),
('m20', 29386788.333333332), ('m11', 28660331.75), ('m02',
31600039.5), ('m30', 2509869073.25), ('m21', 2322307277.5833335),
('m12', 2402708186.25), ('m03', 2783722821.5), ('mu20',
1809343.2177430615), ('mu11', -1686.92005661875), ('mu02',
1810792.8602432348), ('mu30', 9996.168998241425), ('mu21',
-3016.630687713623), ('mu12', -9715.580042093992), ('mu03',
3201.03875875473), ('nu20', 0.07955478602054734), ('nu11',
-7.417197733522105e-05), ('nu02', 0.07961852517063045), ('nu30',
6.364514551103399e-06), ('nu21', -1.920674801580091e-06), ('nu12',
-6.185864860947758e-06), ('nu03', 2.03808656719635e-06)])
4769.0
258.1076476573944
```



5) 컨투어 속성

다음은 컨투어 속성들에 대한 설명과 계산하는 방법입니다.

Aspect Ratio : Contours Line의 가로 세로 비율입니다.

- AspectRatio=Width/Height

- cv2.boundingRect() 함수를 이용하여 가로/세로 크기를 구한 후에 사용합니다.
- x, y, w, h = cv2.boundingRect(cont)
- aspect_ratio = w/h

Extend : Contour Line을 포함하는 사각형 면적대비 Contour의 면적 비율입니다.

- Extend=ObjectArea / BoundingRectagleArea

- area = cv2.contourArea(cont) # Contour Line의 면적
- x, y, w, h = cv2.boundingRect(cont)
- rect_area = w * h # 사각형 면적
- extend = float(area) / rect_area

Solidity : Solidity Ratio(고형비)는 Convex hull 면적 대비 Contour의 면적 비율입니다.

- Solidity=ContourArea / ConvexHullArea

- area = cv2.contourArea(cont) # Contour Line면적
- hull = cv2.convexHull(cont) # Convex hull line
- hull_area = cv2.contourArea(hull) # Convex hull 면적
- solidity = float(area) / hull_area

Extream Points : Contour Line의 좌/우/상/하의 끝점을 찾는 방법입니다.

- leftmost = tuple(cont[cont[:,:,0].argmin()][0])
- rightmost = tuple(cont[cont[:,:,0].argmax()][0])
- topmost = tuple(cont[cont[:,:,1].argmin()][0])
- bottommost = tuple(cont[cont[:,:,1].argmax()][0])

6) cv2.approxPolyDP()

cv2.findContours() 함수에 의해서 찾은 경계선들은 각각의 경계점을 가지고 있습니다. 이 점을 연결하여 선을 그리게 되는데, 이때 이 점의 수를 줄여 근사(Approximation)한 선을 그리는 것을 경계 근사(Contour Approximation)라고 합니다.

cv2.approxPolyDP() 함수는 경계 근사를 찾는데 사용합니다. 이 함수에서 점의 수를 줄이기 위해 더글라스-퓨커(Douglas-Peucker) 알고리즘을 사용합니다.

cv2.approxPolyDP(curve, epsilon, closed) -> approxCurve

- curve - 경계점 배열입니다.
- epsilon - 원본 경계점과 근사치의 최대거리입니다. 최대거리가 클수록

더 먼 곳의 점까지 고려하기 때문에 점의 수가 줄어듭니다.

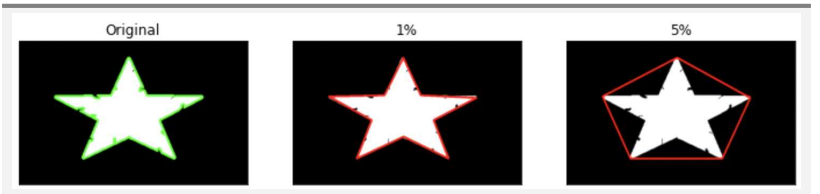
- closed – True이면 폐곡선, False이면 처음과 끝을 연결하지 않습니다.

- 반환값 approxCurve – 근사치가 적용된 경계점들의 배열입니다.

다음 코드는 cv2.approxyPolyDP() 함수를 테스트하는 예입니다.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('star.png')
6  img1 = img.copy()
7  img2 = img.copy()
8
9  imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
10 ret, bin_img  = cv2.threshold(imgray,127,255,0)
11
12 contours, hierachy = cv2.findContours(bin_img, cv2.RETR_TREE,
                                         cv2.CHAIN_APPROX_SIMPLE)
13 cont = contours[0]
14
15 epsilon1 = 0.01*cv2.arcLength(cont, True)
16 epsilon2 = 0.05*cv2.arcLength(cont, True)
17
18 approx1 = cv2.approxPolyDP(cont, epsilon1, True)
19 approx2 = cv2.approxPolyDP(cont, epsilon2, True)
20
21 cv2.drawContours(img, [cont], 0, (0,255,0), 5)
22 cv2.drawContours(img1, [approx1], 0,(255,0,0), 5)
23 cv2.drawContours(img2, [approx2], 0,(255,0,0), 5)
24
25 titles = ['Original', '1%', '5%']
26 images = [img, img1, img2]
27
28 plt.figure(figsize=(12, 4))
29 for i in range(3):
30     plt.subplot(1,3,i+1)
```

```
    plt.title(titles[i])
    plt.imshow(images[i])
    plt.xticks([])
    plt.yticks([])
plt.show()
```



결과에서 보듯이 apploxyPolyDP() 함수의 epsilon 매개변수의 값이 클수록 안쪽으로 들어간 경계점들이 줄어들어 더 원에 가까운 경계선을 찾습니다. 입실론이 0.01인 경우 별 도형의 균열 부분들은 경계선에서 제외되고, 0.05인 경우 별의 바깥 꼭지점들을 연결한 경계선이 찾아집니다.

2절. 객체 탐지

2.1. 경계상자

1) 사각형으로 둘러쌓기

객체의 경계선을 둘러싸는 사각형을 그리는 방법은 2가지가 있습니다.

cv2.boundingRect() 함수는 대상 객체를 포함하는 사각형을 그립니다. 대상의 회전(Rotation)은 무시한 사각형 모양입니다.

cv2.boundingRect(contour)

- contour : cv2.findContours()로 찾은 경계점들입니다.

- 코드 예시

```
x, y, w, h = cv2.boundingRect(contour)
img1 = cv2.rectangle(img1, (x,y), (x+w,y+h), (0,255,0), 3)
```

cv2.minAreaRect() 함수는 컨투어 라인을 모두 포함하면서, 최소한의 영역을 차지하는 기울어진 사각형을 그립니다.

cv2.minAreaRect(contour)

- contour : cv2.findContours()로 찾은 경계점들입니다.

- 코드 예시

```
rect = cv2.minAreaRect(contour)
box = cv2.boxPoints(rect)
box = np.int0(box)
img1 = cv2.drawContours(img1, [box], 0, (0,0,255), 3)
```

2) 원으로 둘러쌓기

cv2.minEnclosingCircle() 함수는 컨투어 라인을 완전히 포함하는 원 중 가장 작은 원을 그립니다.

cv2.minEnclosingCircle(contour)

- contour : cv2.findContours()로 찾은 경계점들입니다.

- 코드 예시

```
(x,y), radius = cv2.minEnclosingCircle(contour)
center = (int(x), int(y))
radius = int(radius)
img1 = cv2.circle(img1, center, radius, (255,255,0), 3)
```

cv2.fitEllipse() 함수는 컨투어 라인을 완전히 포함하는 타원을 그립니다.

cv2.fitEllipse(contour)

- contour : cv2.findContours()로 찾은 경계점들입니다.

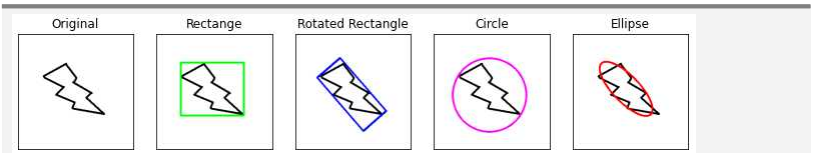
- 코드 예시

```
ellipse = cv2.fitEllipse(contour)
img1 = cv2.ellipse(img1, ellipse, (255,0,0), 3)
```

다음 코드는 cv2.rectangle(), cv2.circle(), cv2.ellipse()를 이용하여 객체의 경계선을 둘러싸는 코드입니다.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('lightning.png')
6  imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7  ret, bin_img = cv2.threshold(imgray, 127, 255, 0)
```

```
8
9  contours, hierachy = cv2.findContours(bin_img, cv2.RETR_TREE,
                                         cv2.CHAIN_APPROX_SIMPLE)
10 cont = contours[1] # 0번째 컨투어는 이미지 전체
11
12 x, y, w, h = cv2.boundingRect(cont)
13 img_rect = cv2.rectangle(img.copy(), (x,y), (x+w,y+h),
                            (0,255,0), 3)
14
15 rect = cv2.minAreaRect(cont)
16 box = np.int_(cv2.boxPoints(rect))
17 img_rotated_rect = cv2.drawContours(img.copy(), [box], 0,
                                       (0,0,255), 3)
18
19 (x,y), radius = cv2.minEnclosingCircle(cont)
20 center = (int(x), int(y))
21 radius = int(radius)
22 img_circle = cv2.circle(img.copy(), center, radius,
                           (255,0,255), 3)
23
24 ellipse = cv2.fitEllipse(cont)
25 img_ellipse = cv2.ellipse(img.copy(), ellipse, (255,0,0), 3)
26
27 titles = ['Original', 'Rectangle',
             'Rotated Rectangle','Circle','Ellipse']
28 images = [img, img_rect, img_rotated_rect, img_circle,
             img_ellipse]
29 plt.figure(figsize=(12,4))
30 for i in range(5):
31     plt.subplot(1, 5, i+1), plt.imshow(images[i]),
32     plt.title(titles[i]), plt.xticks([]), plt.yticks([])
33 plt.show()
```



2.2. cv2.convexHull()

cv2.convexHull() 함수는 컨벡스 헐(Convex Hull; 볼록한 경계선)을 찾아줍니다. 컨벡스 헐은 경계점을 모두 포함하는 볼록한 외관선을 의미합니다. 경계 근사(Contour Approximation)과 유사한 결과지만, 방법은 전혀 다릅니다.

cv2.convexHull(contour)

- contour : cv2.findContours()로 찾은 경계점들입니다.
- 반환값 retval – 컨벡스 헐을 반환합니다.

다음 그림에서 초록 선이 Convex Hull을 나타내고 화살표의 차이가 볼록성 결함(Convexity defect)이라고 합니다. Convexity defect는 경계선과와 볼록선과의 최대차이를 나타냅니다.

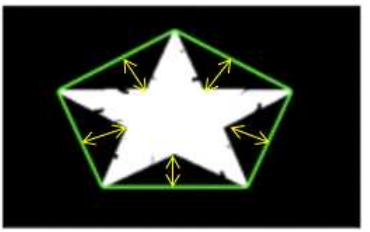
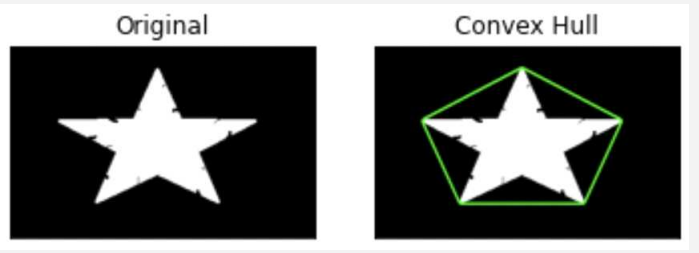


그림 9. 컨벡스 헐

다음 코드는 컨벡스 헐을 그립니다. 별모양의 도형에서 컨벡스 헐을 그리면 앞서 경계선 근사를 이용한 방법과 같은 결과를 얻을 수 있습니다. 경계선 근사는 도형에 따라 입실론 매개변수를 직접 조정해야 하지만 컨벡스 헐은 매개변수를 조정할 필요가 없습니다.

```
1  import cv2
2  import numpy as np
```

```
 3  from matplotlib import pyplot as plt
 4
 5  img = cv2.imread('star.png')
 6  img1 = img.copy()
 7
 8  imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
 9  ret, bin_img  = cv2.threshold(imgray, 127, 255,
                                  cv2.THRESH_BINARY_INV)
10
11  contours, hierachy = cv2.findContours(bin_img, cv2.RETR_TREE,
                                          cv2.CHAIN_APPROX_SIMPLE)
12
13  contour = contours[1]    # 1이 별 모양 주변의 contour
14  hull = cv2.convexHull(contour)
15
16  cv2.drawContours(img1, [hull], 0, (0,255,0), 5)
17
18  titles = ['Original','Convex Hull']
19  images = [img, img1]
20
21  for i in range(2):
22      plt.subplot(1, 2, i+1)
23      plt.title(titles[i])
24      plt.imshow(images[i])
25      plt.xticks([])
26      plt.yticks([])
27
28  plt.show()
```



3절. OpenCV와 머신러닝을 이용한 필기체 숫자 인식

이 절은 손글씨 숫자 데이터(숫자별로 20x20 이미지 500개씩)를 머신러닝을 이용해서 학습시킨 후 웹 카메라에 촬영된 손글씨 숫자를 실시간으로 인식하는 프로그램을 설명합니다.

3.1. 필기체 숫자 데이터 불러오기

다음 그림은 학습에 사용할 숫자 데이터입니다. 0부터 9까지의 숫자가 한 장의 이미지로 되어있으며, 숫자의 크기는 20x20픽셀입니다.

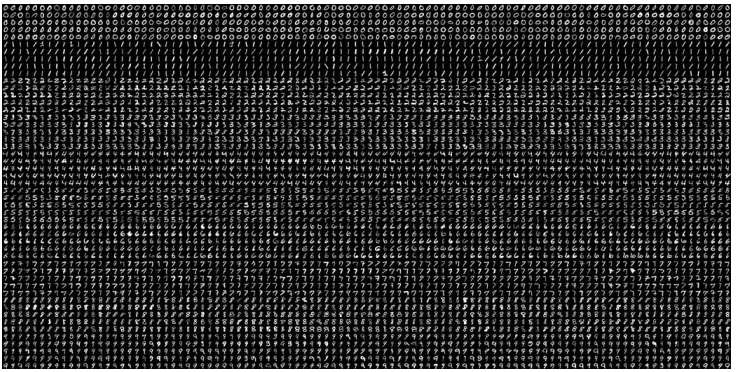


그림 10. 학습에 사용할 숫자 이미지

https://cdn.rawgit.com/opencv/opencv/4.5.0/samples/data/digits.png

다음 코드는 숫자 이미지를 불러옵니다.

```
import cv2
import numpy as np

img = cv2.imread('digits.png', cv2.IMREAD_GRAYSCALE)
print(img.shape)
```

```
(1000, 2000)
```

다음 코드는 불러온 숫자 이미지를 각 숫자 크기로 잘라 20x20 크기의 배열로 만들어 놓습니다.

```
cells = [np.hsplit(row,100) for row in np.vsplit(img,50)]
x = np.array(cells)
print(x.shape)
```

```
(50, 100, 20, 20)
```

코드에서 배열 x의 x[0,0]은 숫자 0이 저장되어 있습니다.

```
[[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   9,  33,   5,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  41, 177, 249, 178,  29,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,  33, 198, 255, 240, 255, 107,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   1,  70, 199, 255, 255, 197, 154, 253,  98,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,  45, 238, 255, 205, 224, 222,  83, 224, 128,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,  25, 202, 255, 193,  40,  99,  54,   0, 190, 197,  16,   0,   0,   0],
 [  0,   0,   0,   0,   0,  20, 163, 246, 152,  72,   0,   0,   0,   0, 184, 252,  74,   0,   0,   0],
 [  0,   0,   0,   0,   0,  97, 255, 118,   0,   1,   0,   0,   0,   0, 184, 255,  82,   0,   0,   0],
 [  0,   0,   0,   0,  20, 218, 216,  17,   0,   0,   0,   0,   0,   0, 183, 255,  78,   0,   0,   0],
 [  0,   0,   0,   0,  67, 255, 138,   0,   0,   0,   0,   0,   0,  24, 215, 188,  22,   0,   0,   0],
 [  0,   0,   0,   0,  69, 247,  87,   0,   0,   0,   0,   0,  31, 185, 212,  54,   0,   0,   0,   0],
 [  0,   0,   0,   0,  71, 237,  68,   0,   0,   0,   0,  94, 211, 177,  17,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,  69, 255, 170,  39,  39, 115, 183, 238, 185,  37,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,  57, 245, 255, 229, 233, 255, 216, 117,  13,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   7, 114, 230, 255, 234, 137,  38,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,  26,  38,  27,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0]]
```

다음 코드는 20x20 크기 배열로 저장돼있는 배열을 400개 데이터를 갖는 1차원 배열로 만들어 독립변수 X로 사용하고, 숫자 레이블 y는 0부터 9까지 500개씩 만듭니다. train_test_split() 함수는 원본 데이터를 학습(train) 데이터와 검증(test) 데이터로 나눕니다. test_size 매개변수는 검증데이터의 비율을 지정합니다. test_size=0.3이면 검증데이터는 전체의 30%를 사용하고 학습데이터는 전체의 70%를 사용하겠다는 의미입니다.

```
X = x[:,:].reshape(-1,400).astype(np.float32)
y = np.repeat(np.arange(10),500)

from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y,
            test_size=0.3)
```

3.2. 머신러닝 분류모형 만들기

싸이킷런(Scikit-learn)은 머신러닝을 위한 라이브러리입니다. 다음 코드는 싸이킷런의 랜덤포레스트 분류기(RandomForestClassifier)를 이용해서 분류 모형을 만들고 학습시킵니다. score() 함수는 검증데이터를 이용해서 예측 모형의 정확도(Accuracy)를 계산합니다.

```
1  from sklearn.ensemble import RandomForestClassifier
2  model = RandomForestClassifier()
3  model.fit(train_X, train_y)
4  print(model.score(test_X, test_y))
```

```
0.9413333333333334
```

3.3. 머신러닝 모형 저장하고 불러오기

학습된 모형을 파일에 저장해 두면 다음 예측 시 다시 학습시키는 과정을 반복하지 않아도 됩니다. 다음 코드처럼 피클을 이용해서 예측 모형을 파일에 저장하고 불러올 수 있습니다.

```
1  import pickle
2  with open("number.model", "wb") as f:
3      pickle.dump(model, f)
```

저장한 모델을 불러와 테스트 데이터셋의 정확도를 확인해 봅니다.

```
1  import pickle
2  with open("number.model", "rb") as f:
3      model = pickle.load(f)
4
5  print(model.score(test_X, test_y))
```

```
0.9413333333333334
```

3.4. 웹 카메라를 이용한 실시간 필기체 숫자 인식하기

다음은 웹 카메라를 이용해서 직접 쓴 글씨를 실시간으로 인식시키기 위한 코드입니다. 카메라를 실행시키기 전에 학습시켜 저장했던 모델을 불러와야 합니다.

```
1  import pickle
2  with open("number.model", "rb") as f:
3      model = pickle.load(f)
```

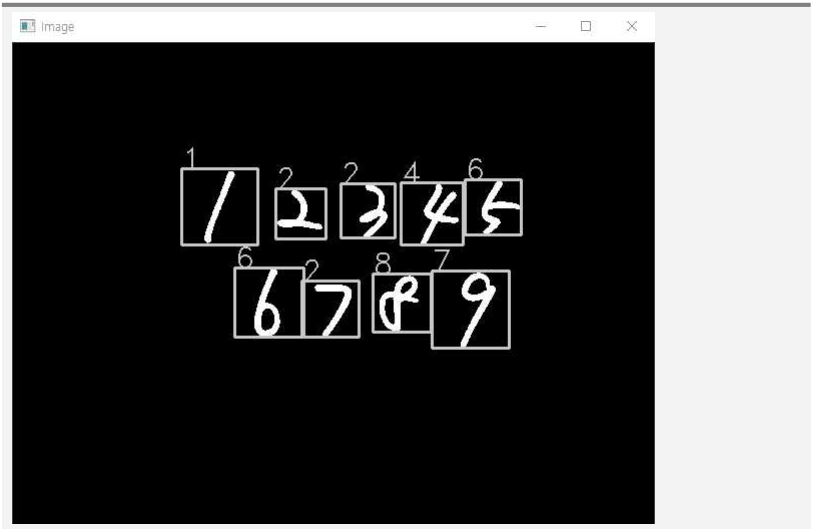
다음 코드의 전체적인 흐름을 아래와 같습니다.

- 카메라에 연결해서 프레임을 읽어온다.
- 프레임을 그레이스케일 이미지로 변환한다.
- 이미지를 이진화한다.(THRESH_BINARY_INV를 이용해서 흰색은 검은색 배경이 되도록 하고 검은색은 흰색이 되도록 한다.)
- 모든 객체의 컨투어를 찾는다.
- 컨투어를 완전히 포함하는 가장 작은 원을 찾는다.
- 반지름이 5보다 큰 객체들에 대해서면 처리한다.
- 반지름과 중심점을 이용해서 사각형의 왼쪽/상단과 오른쪽/하단의 위치를 계산하고 이를 이용해서 사각형을 그린다.
- 사각형 영역을 ROI로 분리해 낸다.
- 학습시킨 이미지와 같은 크기가 되도록 ROI 이미지의 크기를 조정한다. 이때 원본 이미지에서 숫자는 위/아래/좌/우 약 2~3픽셀 여유가 있으므로 ROI 크기를 조정할 때 이를 고려하여 조정한다.
- 크기가 조정된 이미지를 이용해서 숫자를 예측하고 이를 화면에 텍스트로 출력한다.

```
1  import cv2
2  import numpy as np
3
4  cap = cv2.VideoCapture(0)
```

```
5  if cap.isOpened():
6      while True:
7          ret, img = cap.read()
8          if ret:
9              g_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10             _, bin_img = cv2.threshold(g_img, 110, 255,
                                          cv2.THRESH_BINARY_INV)
11             contours, hierarchy = cv2.findContours(bin_img,
                      cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
12             try:
13                 for contour in contours:
14                     (x,y), radius = cv2.minEnclosingCircle(contour)
15                     if radius > 5:
16                         xs, xe = int(x-radius), int(x+radius)
17                         ys, ye = int(y-radius), int(y+radius)
18                         cv2.rectangle(bin_img, (xs,ys),
                                         (xe,ye), (255,0,0), 1)
19                         roi = bin_img[ys:ye, xs:xe]
20                         dst = cv2.resize(roi, dsize=(50, 50),
                                   interpolation=cv2.INTER_AREA)
21                         dst = cv2.resize(dst, dsize=(16, 16),
                                   interpolation=cv2.INTER_AREA)
22                         A = np.zeros((20,20))
23                         A[2:-2,2:-2] = dst[:,:]
24                         A = A.reshape(-1,400)
25                         num = model.predict(A)
26                         cv2.putText(bin_img, str(num), (xs, ys),
                                       cv2.FONT_HERSHEY_PLAIN,
                                       2, (255,0,0))
27             except Exception as e:
28                 print(e)
29             cv2.imshow("Image", bin_img)
30             if cv2.waitKey(1)&0xFF == 27: # ESC
31                 break
32         else:
33             print("No Frame")
34             break
```

```
else :
    print("Camera not opened")

cap.release()
cv2.destroyAllWindows()
```



앞에서 테스트셋을 이용해서 평가한 평가 점수는 정확도가 94% 정도 나왔었습니다. 이 정확도는 상당히 높은 수치입니다. 그런데 이것을 너무 믿으면 안 됩니다. 실제 웹 카메라를 이용해 숫자를 인식시켜보면 숫자의 크기, 굵기 등에 민감하고, 전체적으로 인식률이 떨어집니다. 테스트은 학습에 사용한 데이터처럼 숫자의 크기와 굵기가 대부분 일정하므로 정확도가 높게 나오는 것입니다. 실제 여러분이 손으로 숫자를 쓰고 인식시킬 경우는 숫자의 굵기, 크기 등이 학습시켰던 이미지와 다르면 예측 정확도가 현저하게 떨어질 수 있습니다.

여러분이 이 예제의 실행 결과에 만족하지 않을 수 있습니다. 객체 탐지를 위한 기술이 이것이 전부는 아닙니다. 여러분이 인공신경망 딥

러닝을 배운다면 더 높은 정확도로 객체를 탐지하는 기술을 배울 수 있습니다.

이 책은 영상처리를 위한 기본 서적입니다. 머신러닝이나 인공신경망 딥러닝은 관련 서적을 참고하시길 바랍니다.