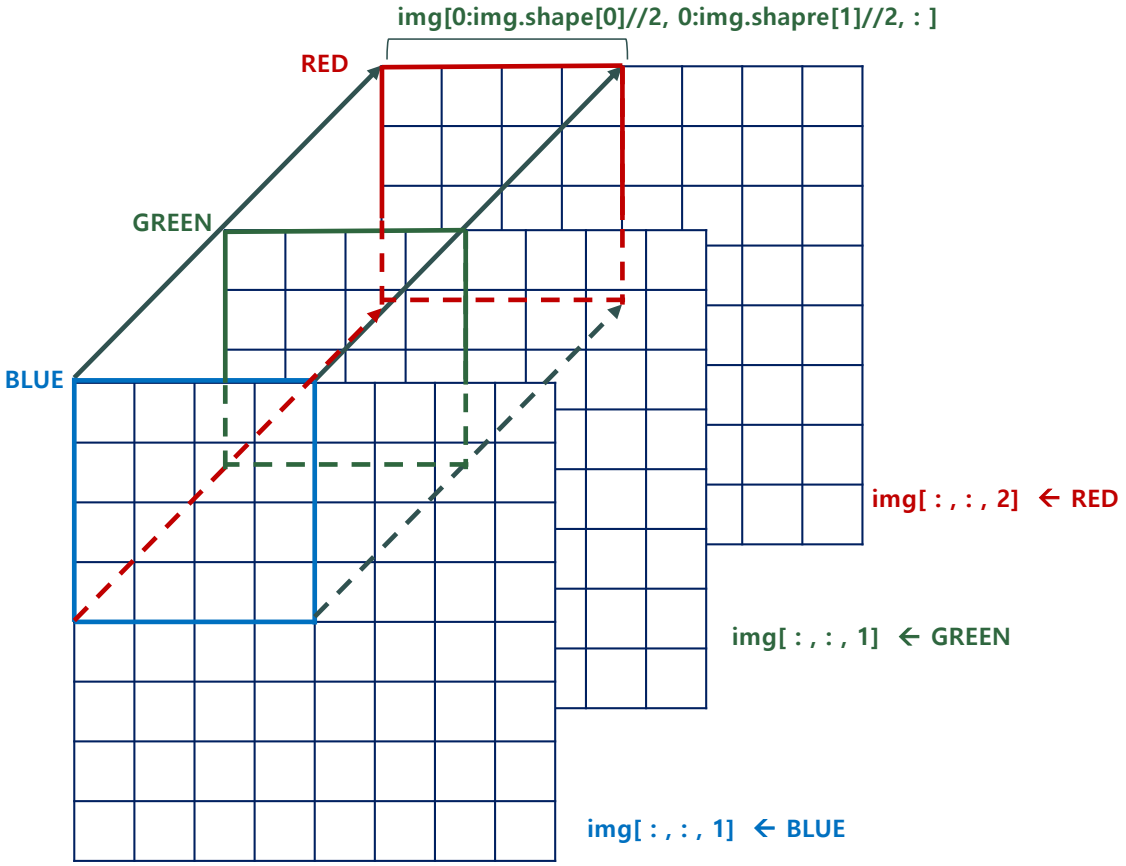
파이썬으로 배우는 딥러닝

이미지 전처리 (OpenCV)

Ch.03 OpenCV영상처리기능 연습하기

OpenCV의 BGR 데이터 구조

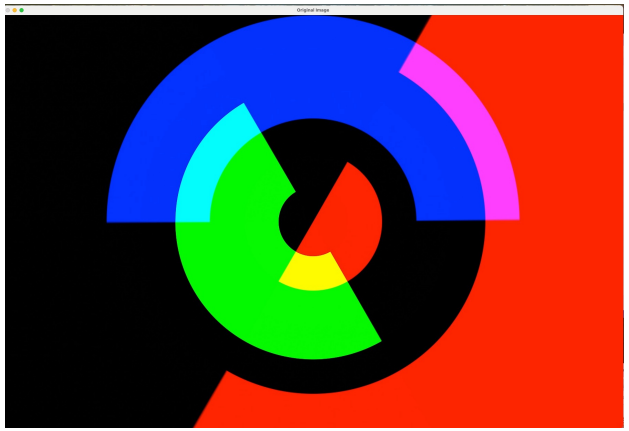
◆ **BGR 채널 구조**



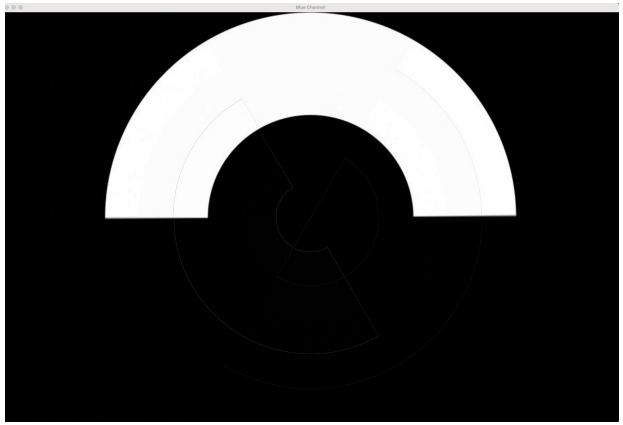
[OpenCV 예제1] BGR채널별 이미지 표현

◆ **컬러 이미지를 컬러 채널별로 표시하기**

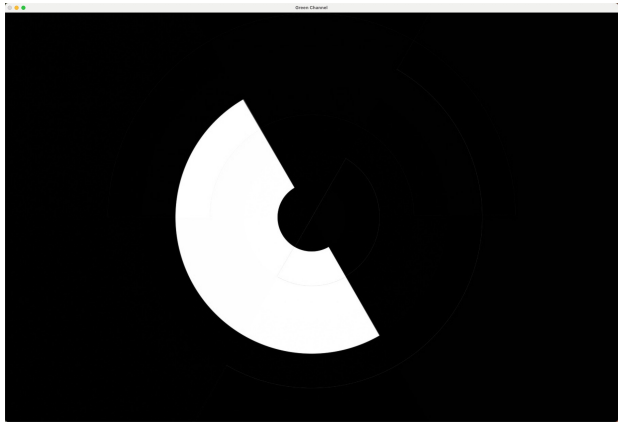
Original image



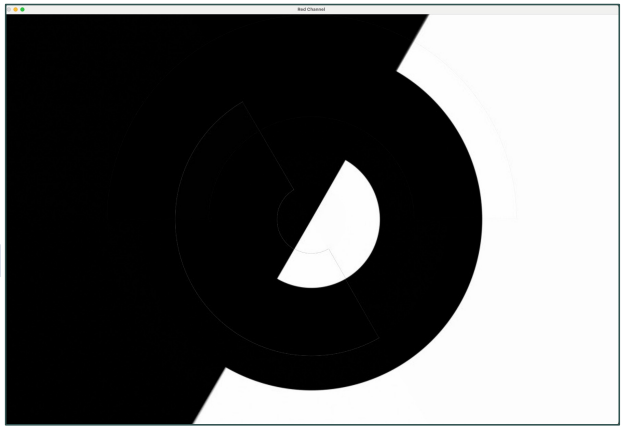
Blue channel



Green channel



Red channel



[OpenCV 예제1] BGR채널별 이미지 표현

◆ **Ch.03₩3-1.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv
import numpy as np

# 1. 이미지 불러오기
img = cv.imread('Ch.03/RGB.jpg')  # 이미지 경로 지정
if img is None:
    raise FileNotFoundError("이미지 파일을 찾을 수 없습니다.")

# 2. 이미지 출력 (OpenCV는 BGR 그대로 출력)
cv.imshow('Original Image', img)
cv.imshow('Blue Channel', img[:, :, 0])
cv.imshow('Green Channel', img[:, :, 1])
cv.imshow('Red Channel', img[:, :, 2])

# 6. 키 입력 대기 후 창 닫기
cv.waitKey(0)
cv.destroyAllWindows()
```

이진화 알고리즘 – 오츄(OTSU) 알고리즘

◆ 오추 알고리즘이란?

- 정의 : 이미지의 히스토그램을 분석하여 두 개의 클래스(배경과 객체)간의 분산이 최대가 되는 임계값을 자동으로 찾아주는 방법
- 방법 : 픽셀값을 기준으로 두 그룹(배경/객체)로 나누되 두 그룹간의 분산(분리도)가 가장 큰 지점을 임계값으로 설정
- 적용 단계
  - ✓ 입력 : 그레이스케일 이미지
  - ✓ 이미지 히스토그램 계산 : 각 밝기 값(0~255)의 빈도 측정
  - ✓ 클래스(배경/객체)간 분산을 최대화 하는 t(Threashold)를 선택
  - ✓ 최적의 t를 기준으로 이미지를 이진화 – t보다 작으면 0, 크면 255

[OpenCV 예제2] 이진화 알고리즘 – 오츄(OTSU) 알고리즘

◆ **Ch.03₩3-2.py에 아래 코드 작성해 넣기**

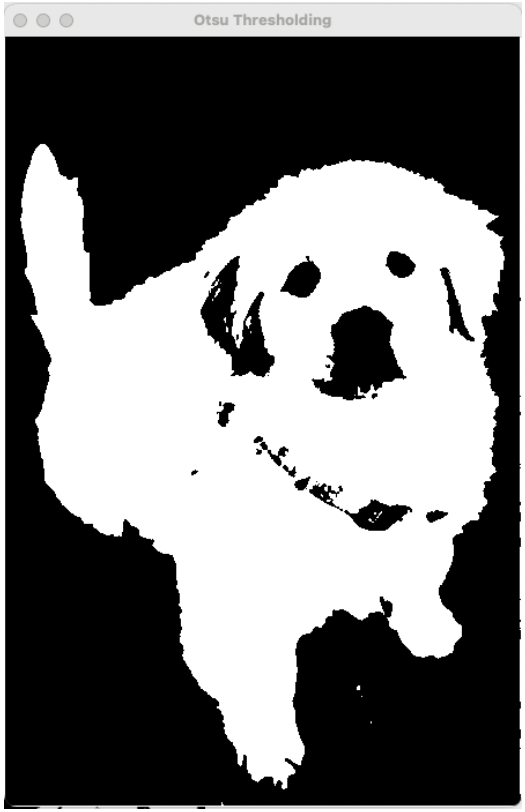
```python
import cv2 as cv

# 이미지 불러오기 (grayscale)
img = cv.imread('Ch.03/dog.jpg', cv.IMREAD_GRAYSCALE)

# 오츄 이진화 적용
ret, binary_img = cv.threshold(img, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)

print("자동 계산된 임계값:", ret)

# 결과 출력
cv.imshow('Original', img)
cv.imshow('Otsu Thresholding', binary_img)
cv.waitKey(0)
cv.destroyAllWindows()
```



이진화 알고리즘 – 오츄(OTSU) 알고리즘

◆ **t, bin_img = cv.threshold(img, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)**

- img: 입력 이미지. 현재는 Grayscale
- 0: 자동으로 임계값을 계산하는 것을 의미. Otsu's 방법을 사용하므로, 직접 임계값을 지정하지 않고 자동으로 선택
- 255: 픽셀 값이 임계값을 넘으면 설정할 최대값. 이 코드에서는 이진화 결과에서 초과한 값은 255(흰색)가 됨.
- cv.THRESH_BINARY + cv.THRESH_OTSU: 이진화 방법. cv.THRESH_BINARY는 기본 이진화를 의미하며, cv.THRESH_OTSU는 Otsu's Thresholding 방법을 추가로 사용하여 임계값을 자동으로 결정

모폴로지

◆ 모폴로지란?

- 디지털 이미지에서는 픽셀 집합의 형태적 특징을 처리하는 기법
- 기본적으로는 "구조 요소(Structuring Element)"를 이용해 픽셀을 변화시키는 방식

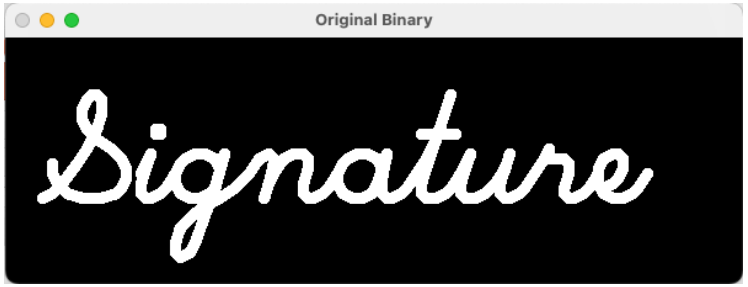
◆ 주요 연산

연산 이름	설명	주요 효과
침식(Erosion)	객체 외곽을 깎음 (배경이 침투)	잡음 제거, 객체 축소
팽창(Dilation)	객체 외곽을 확장 (객체가 배경 침식)	구멍 메움, 객체 확장
열림(Opening)	침식 후 팽창	작은 잡음 제거
닫힘(Closing)	팽창 후 침식	객체의 구멍 제거
그 외	Morphological Gradient, Top-hat, Black-hat 등	외곽 강조, 배경 제거 등

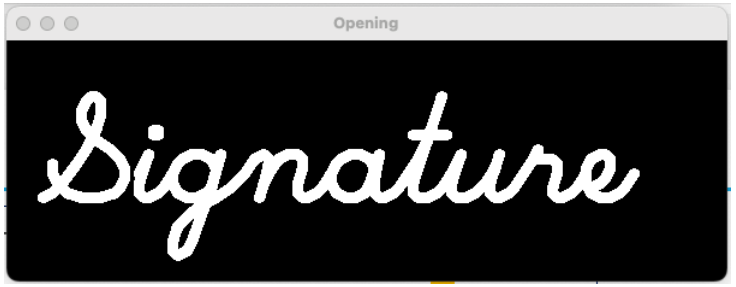
모폴로지

◆ 모폴로지 연산의 예

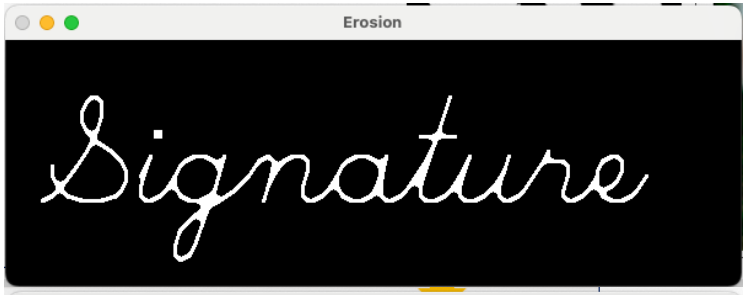
원본



열림



침식



닫힘

팽창

[OpenCV 예제3] 모폴로지

◆ **Ch.03₩3-3.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv
import numpy as np

# 1. 빈 흰 캔버스 생성
canvas = np.ones((200, 600), dtype=np.uint8) * 255

# 2. 'Signature' 텍스트를 붓글씨 느낌으로 그림
cv.putText(canvas, 'Signature', (30, 130), cv.FONT_HERSHEY_SCRIPT_SIMPLEX, 4, (0,), 8, cv.LINE_AA)

# 3. 이미지 이진화 (글씨 흰색으로, 배경 검정색으로 반전)
_, binary = cv.threshold(canvas, 127, 255, cv.THRESH_BINARY_INV)

# 4. 모폴로지용 커널 생성 (원형, 크기 7x7)
kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (7,7))

# 5. 모폴로지 연산 수행
erosion = cv.erode(binary, kernel, iterations=1)
dilation = cv.dilate(binary, kernel, iterations=1)
opening = cv.morphologyEx(binary, cv.MORPH_OPEN, kernel)
closing = cv.morphologyEx(binary, cv.MORPH_CLOSE, kernel)

# 6. 결과 출력
cv.imshow('Original Binary', binary)
cv.imshow('Erosion', erosion)
cv.imshow('Dilation', dilation)
cv.imshow('Opening', opening)
cv.imshow('Closing', closing)

cv.waitKey(0)
cv.destroyAllWindows()
```

모폴로지

◆ **erosion = cv.erode(binary, kernel, iterations=1)**

- 이미지에서 흰색(전경) 영역을 점점 줄여 나가는 연산
- 모폴로지 연산 중 가장 기본적인 것 중 하나이며, 노이즈 제거, 객체 분리 등에 사용됨.
- 원리 : 커널 중심을 기준으로 주변 픽셀들을 검사해서 커널 크기만큼 모두 흰색(255)이 아니면, 중심 픽셀을 검정(0)으로 바꿈
- binary : 입력 이진 이미지 (보통 0 또는 255 값으로 구성)
- kernel : 침식 연산을 수행할 때 사용할 구조 요소(커널)
- 보통 3x3, 5x5 같은 사각형, 원형, 십자형 등이 쓰임
- 이 커널이 이미지 위를 이동하며 픽셀 주변 상태를 검사함
- iterations=1 : 침식 연산을 몇 번 반복할지 지정 (기본값 1)

모폴로지

◆ **dilation = cv2.dilate(binary, kernel, iterations=1)**

- **팽창(Dilation)**은 이진 이미지에서 흰색(전경) 영역을 확장시키는 모폴로지 연산
- 객체의 경계가 바깥쪽으로 넓어지고, 끊어진 부분이 연결될 수 있음.
- 노이즈를 보완하거나 객체를 두껍게 할 때 쓰임.
- 원리 : 커널이 이미지 위를 이동하며 각 위치에서 커널 영역 내에 흰색(255)이 하나라도 있으면 중심 픽셀을 흰색(255)으로 변경
- binary: 입력 이진 이미지 (0과 255 값으로 된 흑백 이미지)
- kernel: 팽창 연산에 사용할 구조 요소(커널) — 보통 사각형, 원형, 십자형 모양의 작은 행렬
- iterations=1: 팽창 연산 반복 횟수 (기본값 1)

모폴로지

◆ **opening = cv.morphologyEx(binary, cv.MORPH_OPEN, kernel)**

- 여러 모폴로지 연산(열림, 닫힘, 그라디언트 등)을 수행할 수 있는 함수
- cv2.MORPH_OPEN은 열림 연산을 지정하는 옵션
- 열림은 침식(Erosion) 후 팽창(Dilation)을 수행하는 연산입니다.
- 주로 **작은 노이즈(흰 점)**를 제거하거나, 이미지 내에서 객체의 경계를 조금 부드럽게 만들 때 사용됨.
- 원리 : 입력 이미지에 대해 먼저 침식을 해서 흰색 영역이 줄어들고, 그 다음 팽창을 해서 본래 크기로 어느 정도 복구시킴 → 작고 독립된 노이즈는 제거됨.
- binary: 입력 이진 이미지
- cv2.MORPH_OPEN: 열림 연산 지정
- kernel: 구조 요소(커널), 보통 3x3 ~ 7x7 크기의 사각형, 원형, 십자형 등

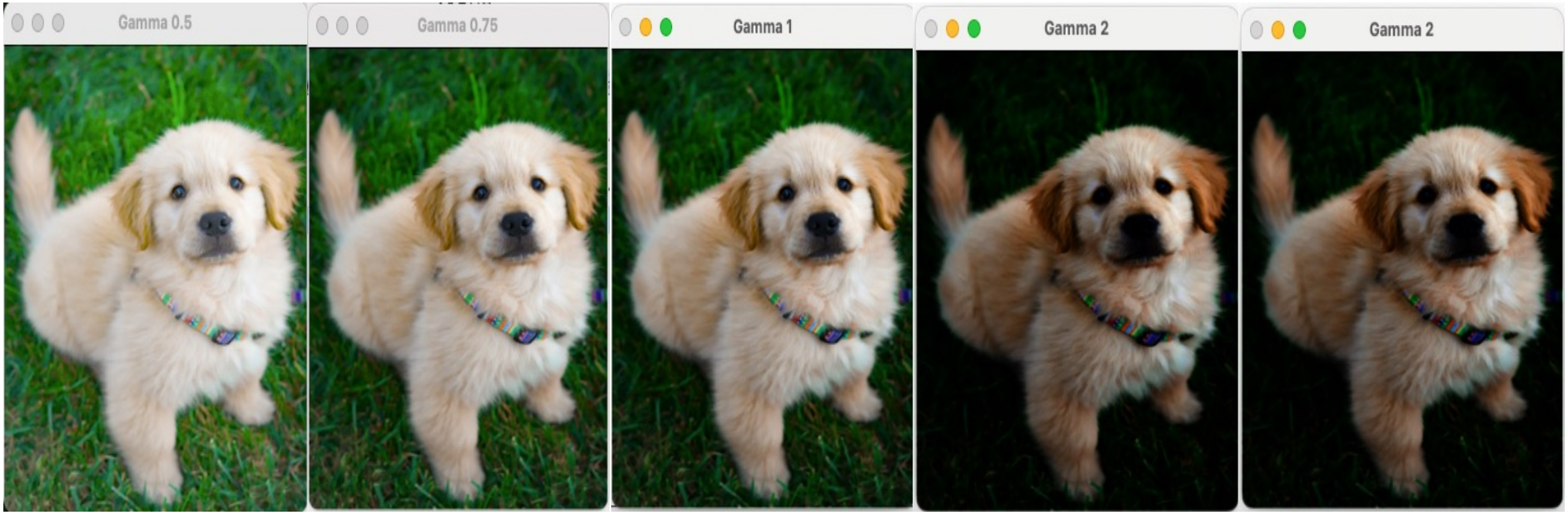
모폴로지

◆ **closing = cv.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)**

- 여러 모폴로지 연산(열림, 닫힘, 그라디언트 등)을 수행할 수 있는 함수
- cv2.MORPH_CLOSE 옵션은 닫힘(Close) 연산을 의미
- 원리 : 팽창 후 침식하여 결과적으로, 작은 구멍은 메워지고, 선이 끊어졌던 부분이 연결되어 더 매끄러운 형태가 됨
- binary : 이진화된 입력 이미지 (글씨는 보통 흰색(255), 배경 검정색(0))
- kernel : 커널(구조 요소), 모양과 크기에 따라 결과가 달라짐 (예: 원형, 사각형)

감마(밝기) 보정

◆ **감마 보정 예시**



[OpenCV 예제4] 감마(밝기) 보정

◆ **Ch.03₩3-4.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv
import numpy as np

# 감마 보정 함수 (단일 이미지 처리)
def adjust_gamma(image, gamma):
    image1 = image / 255.0
    return np.uint8(255*(image1**gamma))

# 이미지 불러오기
img = cv.imread('Ch.03/dog.jpg')  # 이미지 경로 수정
img = cv.resize(img, (300, 300))  # 보기 좋게 리사이즈

# 감마 값 리스트
gammas = [0.5, 0.75, 1, 2, 3]

# 각 감마값에 대해 이미지 처리 및 표시
for gamma in gammas:
    adjusted = adjust_gamma(img, gamma)
    window_name = f'Gamma {gamma}'
    cv.imshow(window_name, adjusted)

# 아무 키나 누를 때까지 대기 후 창 닫기
cv.waitKey(0)
cv.destroyAllWindows()
```

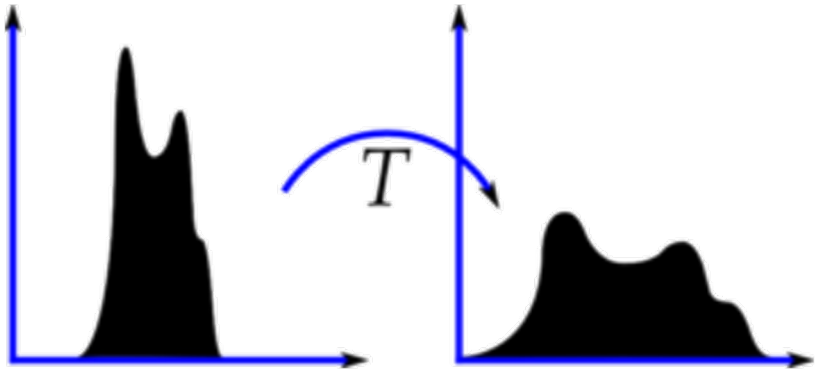
히스토그램 평활화(Equalization)

◆ **정의**

- 이미지의 픽셀 값 분포(히스토그램)를 균등하게(평탄하게) 만들어서, 전체 밝기 범위를 고르게 활용하게 하는 기법

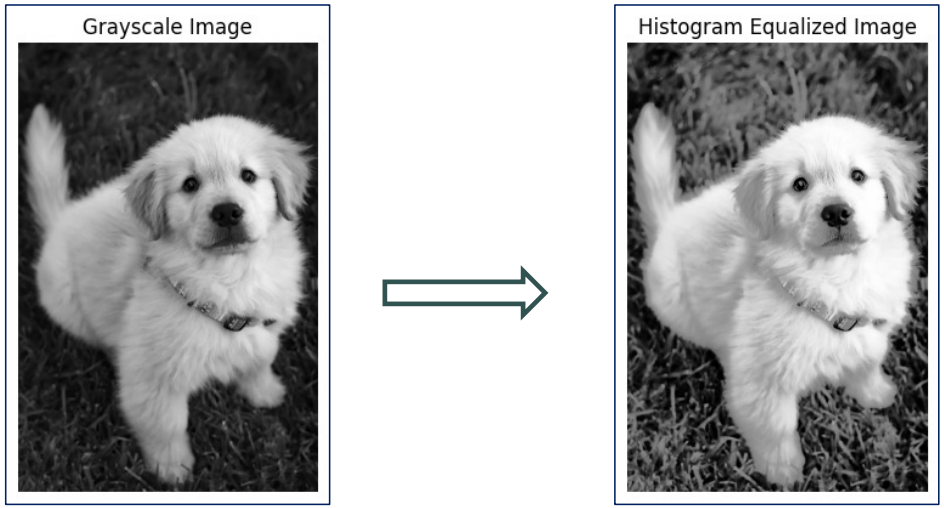
◆ **활용방안**

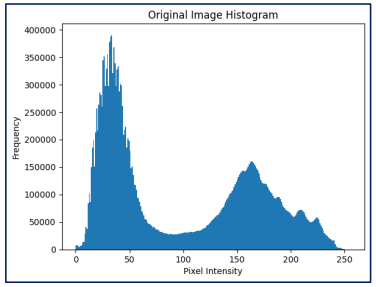
- 어두운 사진이나 밝은 사진은 특정 밝기 영역에만 픽셀이 몰려 있음.
- 이럴 때 히스토그램 평활화를 하면 픽셀들이 전체 밝기 영역에 골고루 분포하게 되어, 디테일이 살아나고 명암 대비가 개선됨.

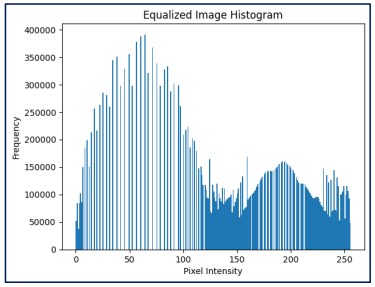


히스토그램 평활화(Equalization)

◆ 그레이스케일(Grayscale)과 평활화(Equalization)의 비교







[OpenCV 예제5] 히스토그램 평활화(Equalization)

◆ **Ch.03₩3-5.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv

# 이미지 불러오기 (흑백)
img = cv.imread('Ch.03/dog.jpg', cv.IMREAD_GRAYSCALE)

# 히스토그램 평활화
equalized = cv.equalizeHist(img)

# 결과 디스플레이
cv.imshow('Original Image', img)
cv.imshow('Histogram Equalized Image', equalized)

# 아무 키나 누르면 종료
cv.waitKey(0)
cv.destroyAllWindows()
```

히스토그램 평활화(Equalization)

◆ **equalized = cv.equalizeHist(img)**

- OpenCV를 사용하여 그레이스케일 이미지의 **히스토그램 평활화(histogram equalization)**를 수행하는 코드
- 히스토그램 평활화는 이미지의 대비를 높여 어두운 부분과 밝은 부분을 더 뚜렷하게 보이도록 하는 기법 이는 픽셀 값의 분포가 특정 범위에 몰려 있을 때, 이를 균등하게 분포시키기 위해 사용

가우시안 블러링

◆ 정의

- 가우시안 함수를 이용해 주변 픽셀의 값을 가중 평균하여 이미지를 부드럽게(smooth) 만드는 필터

◆ 사용 목적

목적	설명
노이즈 제거	사진의 작은 점이나 잡음을 줄이기 위해
에지 검출 전처리	Sobel, Canny 등 에지 검출 전에 적용하면 **의미 없는 에지(노이즈)**를 억제
이미지 디테일 억제	너무 날카로운 이미지나 세세한 구조를 흐려서 **전반적인 구조 파악**에 도움
배경 흐림 효과	심도 효과(fake depth-of-field) 등 시각적 효과
다운샘플링 전처리	이미지 크기를 줄이기 전에 블러링하여 **aliasing** 방지

가우시안 블러링

◆ 가우시안 블러링 예시

[OpenCV 예제6] 가우시안 블러링

◆ **Ch.03₩3-6.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv

# 이미지 불러오기
img = cv.imread('Ch.03/dog.jpg')  # 이미지 경로를 맞게 수정하세요

# 가우시안 블러 적용 (커널 크기 5x5, 시그마 0은 자동 계산)
blurred = cv.GaussianBlur(img, (5, 5), 0)

# 원본 이미지와 블러 이미지 출력
cv.imshow('Original Image', img)
cv.imshow('Gaussian Blurred Image', blurred)

cv.waitKey(0)
cv.destroyAllWindows()
```

가우시안 블러링

◆ **cv.GaussianBlur(gray, (5,5), 0.0)**

- Image에 노이즈를 줄이고, 부드럽게 만드는 필터
- 선명한 경계를 부드럽고 흐리게 만들어주는 효과가 있어 영상 전처리나 윤곽선 검출전에 자주 사용됨.
- gray : 입력영상(흑백 영상 또는 컬러 영상도 가능)
- (5,5) : 가로, 세로 방향 필터 크기
- 0.0 : X 방향 가우시안 표준 편차

필터 크기	결과
작을 때	**살짝 흐림**, 세부 구조 유지
클 때	**더 많이 흐려짐**, 윤곽이 부드러워짐
매우 클 때	거의 전체가 **부드럽게 퍼짐**, 경계 사라짐

σ 값	시각적 효과
작을 때 (예: 0.5)	흐림 효과가 **약함**, 중심 위주로 평균
클 때 (예: 5.0)	흐림 효과가 **강해짐**, 넓은 범위까지 퍼짐
매우 클 때 (예: 20.0)	**전체적으로 매우 부드러움**, 윤곽이 흐려짐

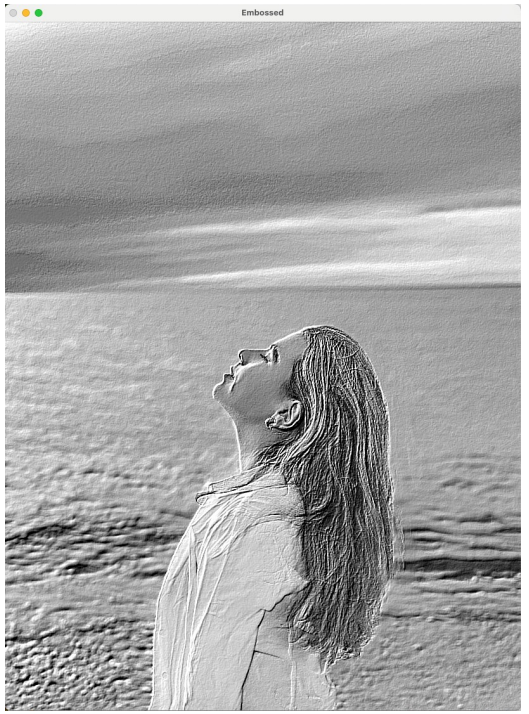
엠보싱 기법

◆ **정의**

- 엠보싱(Embossing)은 이미지의 윤곽(에지, 경계선)을 강조하면서 음영을 주어 입체감을 표현하는 기법

◆ 특징

- 회색톤 이미지로 바뀌며, 밝고 어두운 음영을 통해 입체감 표현함.
- 텍스처 강조, 엣지 스타일 필터, 예술 효과 등
- 방향성 있는 필터로써 빛이 한 방향에서 오는 것처럼 보이게 함



[OpenCV 예제7] 엠보싱 기법

◆ **Ch.03₩3-7.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv
import numpy as np

# 이미지 불러오기 (그레이스케일)
img = cv.imread('Ch.03/girl.jpg', cv.IMREAD_GRAYSCALE)

# 엠보싱 커널 정의
kernel = np.array([[-2, -1, 0],
                   [-1,  1, 1],
                   [ 0,  1, 2]], dtype=np.float32)

# 필터 적용 (엠보싱)
embossed = cv.filter2D(img, -1, kernel)

# 원래 이미지와 엠보싱 이미지 디스플레이
cv.imshow('Original', img)
cv.imshow('Embossed', embossed)

cv.waitKey(0)
cv.destroyAllWindows()
```

엠보싱 기법

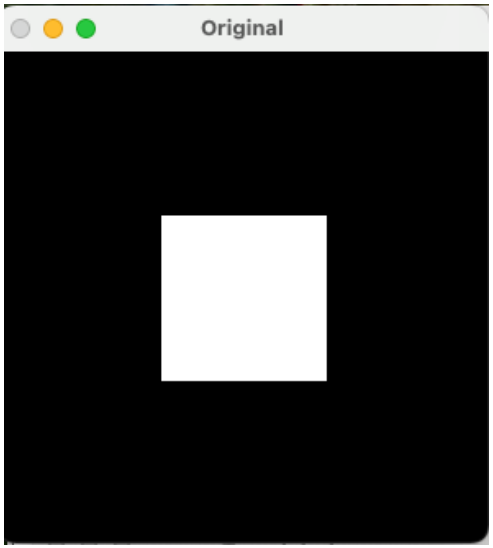
◆ **embossed = cv.filter2D(img, -1, kernel)**

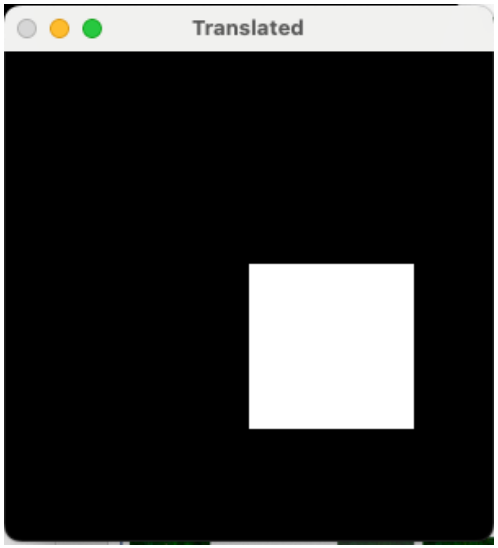
- img 이미지의 모든 픽셀에 대해 kernel을 슬라이딩 적용(컨볼루션) 하여 새로운 이미지를 생성
- 커널의 중심을 기준으로 주변 픽셀과 가중치 곱을 한 뒤 더해서 새로운 값을 계산
- 이때 -1은 출력 이미지가 입력과 동일한 데이터 타입을 유지하라는 의미
- 엠보싱 커널 예시 : 대각선방향 밝기 변화 강조

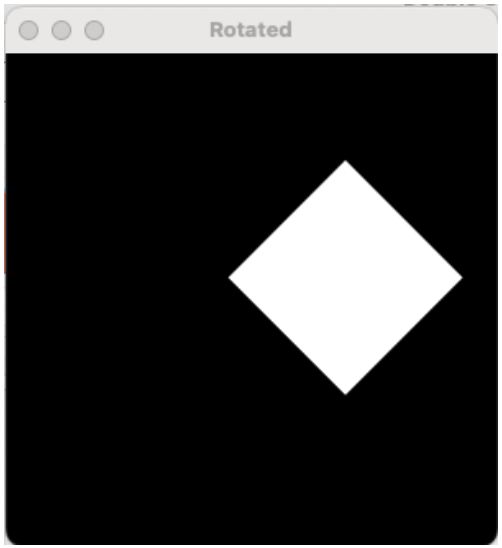
```
kernel = np.array([[-2, -1, 0],
                   [-1,  1, 1],
                   [ 0,  1, 2]], dtype=np.float32)
```

이미지의 이동과 회전

◆ **도형의 이동과 회전 예시**







[OpenCV 예제8] 이미지의 이동과 회전

◆ **Ch.03₩3-8.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv
import numpy as np

# 1. 검정 배경 생성 (300x300 크기)
img = np.zeros((300, 300, 3), dtype=np.uint8)

# 2. 사각형 좌표 정의: 좌상단 (100, 100) → 우하단 (200, 200)
start_point = (100, 100)
end_point = (200, 200)

# 흰색 사각형 그리기
cv.rectangle(img, start_point, end_point, (255, 255, 255), -1)

# 3. 이동 (translation): 오른쪽 50픽셀, 아래 30픽셀
tx, ty = 50, 30
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
translated = cv.warpAffine(img, translation_matrix, (300, 300))

# 4. 회전 (rotation): 중심 기준 45도 회전
center = (150, 150)  # 이미지 중심
angle = 45
scale = 1.0
rotation_matrix = cv.getRotationMatrix2D(center, angle, scale)
rotated = cv.warpAffine(translated, rotation_matrix, (300, 300))

# 5. 결과 출력
cv.imshow('Original', img)
cv.imshow('Translated', translated)
cv.imshow('Rotated', rotated)

cv.waitKey(0)
cv.destroyAllWindows()
```

이미지의 이동과 회전

◆ **translated = cv.warpAffine(img, translation_matrix, (300, 300))**

- 이미지를 이동시키는 함수
- Affine 변환은 직선성과 평행성을 보존하면서 이미지를 변형하는 기법
- translation_matrix: 이동 행렬 (2×3 affine 변환 행렬)
- (300, 300): 출력 이미지의 크기 (너비, 높이)
- translated: 변환(이동)된 결과 이미지

이미지의 이동과 회전

◆ **rotation_matrix = cv.getRotationMatrix2D(center, angle, scale)**

- OpenCV에서 회전 변환 행렬(rotation affine matrix) 을 생성하는 함수
- center : 회전의 중심점(예: (150, 150)
- angle : 회전각도(도 단위)
- scale : 확대 축소 비율(eg. 1.0은 크기 유지)

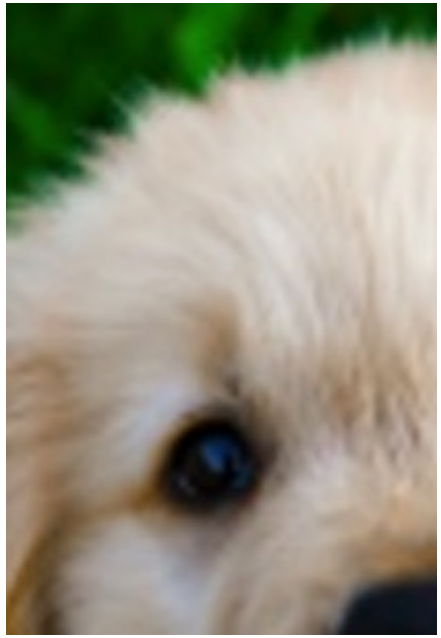
영상 보간

◆ **확대(Resize) 후 보간 옵션에 따른 영상의 차이**

Original

NEAREST

LINEAR



CUBIC

[OpenCV 예제9] 영상 보간

◆ **Ch.03₩3-9.py에 아래 코드 작성해 넣기**

```python
import cv2 as cv

# 이미지 불러오기
img = cv.imread('Ch.03/dog.jpg')  # 이미지 경로에 맞게 수정하세요

# 새 크기 지정 (예: 원본보다 2배 확대)
new_size = (img.shape[1]*2, img.shape[0]*2)

# 서로 다른 보간 방법으로 리사이즈
res_nearest = cv.resize(img, new_size, interpolation=cv.INTER_NEAREST)
res_linear = cv.resize(img, new_size, interpolation=cv.INTER_LINEAR)
res_cubic = cv.resize(img, new_size, interpolation=cv.INTER_CUBIC)

# 결과 출력
cv.imshow('Original', img)
cv.imshow('INTER_NEAREST', res_nearest)
cv.imshow('INTER_LINEAR', res_linear)
cv.imshow('INTER_CUBIC', res_cubic)

cv.waitKey(0)
cv.destroyAllWindows()
```

영상 보간

◆ **res_nearest = cv.resize(img, new_size, interpolation=cv.INTER_NEAREST)**

- OpenCV를 사용하여 이미지 크기를 확장하는 코드. 여기서 이미지 보간(interpolation) 방식을 지정하여 이미지를 크기 조정
- img: 원본 이미지(패치)
- new_size: 출력 이미지의 크기를 직접 지정
- cv.INTER_NEAREST: 최근방 이웃 보간법, 빠르지만 이미지가 확대될 때 계단 현상발생
- cv.INTER_LINEAR: 선형 보간법, 더 부드러운 이미지 확장을 제공
- cv.INTER_CUBIC: 3차 보간법으로, 이미지 확장 시 매우 부드러운 결과를 얻을 수 있지만 계산량이 많음