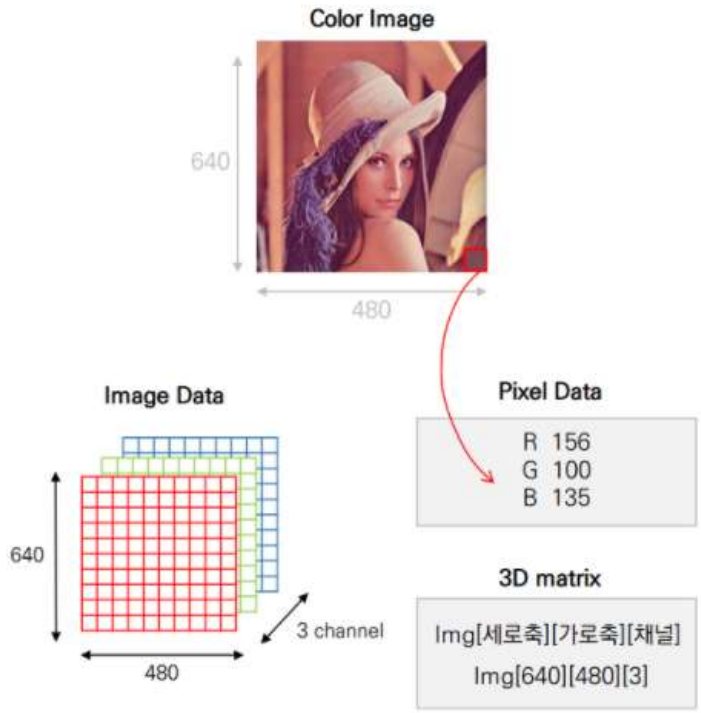
이미지 데이터 처리

OpenCV

-이미지 처리를 위한 다양한 기능 제공

1. 픽셀값 읽기

픽셀 : 디지털 화상을 구성하는 기본적인 단위

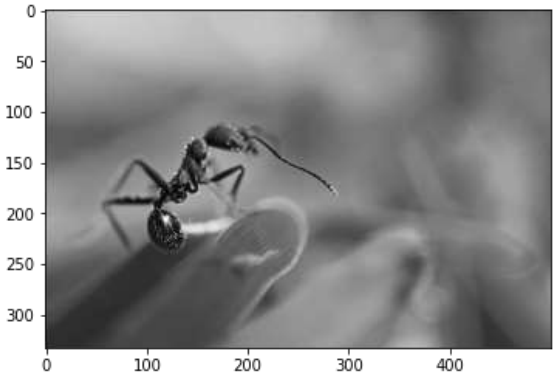


하나의 셀마다 R(0~255), G(0~255), B(0`~255)의 색상을 조합하여 한 셀의 색상을 결정한다.

2. 이미지 종류

1. 컬러이미지 : RGB 컬러로 표현된 이미지

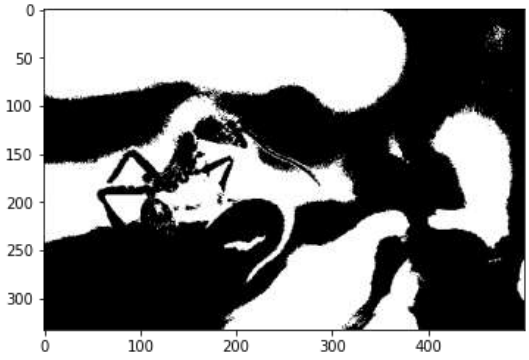
2. 그레이 스케일 이미지 : 휘도(밝기 정도)만으로 표현된 이미지



3. 이진화 이미지 : 그레이스케일 이미지보다 더 정보를 줄여서 특징량을 돋보이게 한 것

→ 흑과 백으로만 표현

→ 경계값을 설정하고 픽셀값이 경계값보다 크면 백(255)을 값을 주고 경계값보다 작으면 흑(0)을 주어서 흑백이미지로 변환



배열 크기 확인
print(len(img)) print(len(img[0])) print(len(img[0][0]))
→배열 전체의 크기를 확인 →첫 번째 행의 배열 크기 확인 →첫번 째 행 첫 번째 열의 배열 크기 확인

픽셀값 확인

cv2.cvtColor(img, cv2.COLOR_BGRRGB)

→ 픽셀값의 정렬순을 RGB로 바꾼다.

print(img)

```
[[[60   4   33]
 [28   0   10]
 [12   7    9]
 ...
 [ 4    2   0 ]
 [ 13   0   2 ]]
 ...
```

Ex) 만약 픽셀값이 [60 40 33]이 나온다면 RGB 순서대로 60,40,33의 값을 가지는 것

이미지 종류에 따른 픽셀값 확인

컬러이미지 - 데이터 프레임 형식으로 확인

```
B, G, R = cv2.split(img)
B_df = pd.DataFrame(B)
G_df = pd.DataFrame(G)
R_df = pd.DataFrame(R)
```

그레이스케일 이미지로 변환

```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
print(gray_img.shape)

gr_df = pd.DataFrame(gray_img)
```

이진화 이미지로 변환

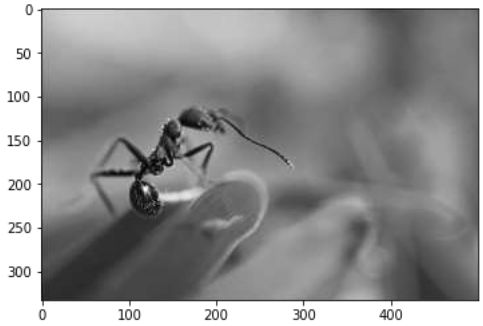
```
ret, bin_img = cv2threadhold(gray_img , 128 255, cv2.THRESH_BINARY)

bin_df = pd.DataFrame(bin_img)
```

2. 머신러닝을 위한 데이터 준비

2-1. 데이터 세트 작성

그레이스케일 이미지, 이진화 이미지로 변환하여 특징을 찾아냈지만 알고리즘에 적합한 형태로 변형해야한다.

2-1-1 모든 픽셀값을 평면에 늘어놓기



	설명변수				목적변수
	픽셀 1	픽셀2	픽셀3	. . .	
이미지1	19	6	8	. . .	개미
이미지2	11	75	6	. . .	벌
. . .	. . .	. . .	. . .	. . . .	. . .
	13.	2	45	. . .	개미

결합 { 설명변수 : 이미지의 픽셀값
목적변수 : 이미지의 피사체 레이블

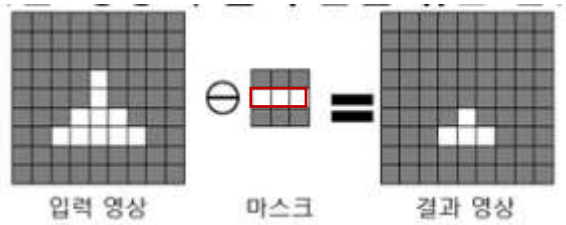
2-2 노이즈 제거

이미지와 같은 차원수가 많은 비정형 데이터의 경우 노이즈(의미가 없는 특징량)를 제거한 특징량을 가져와야 성능이 높은 모델을 만들 수 있다.

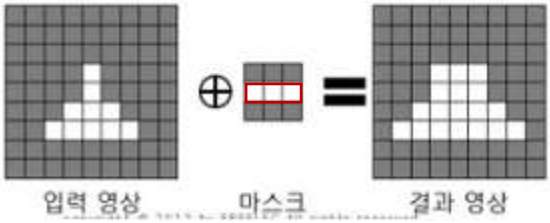
2-2-1 모폴로지 변환

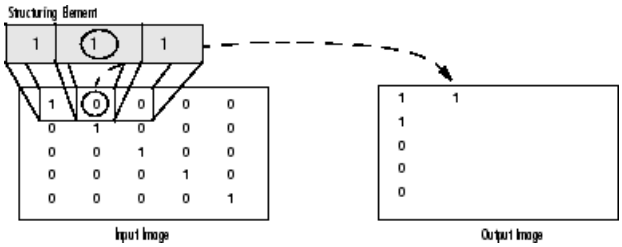
이진화를 했을 때 흰색 영역이나 검은색 영역이 원하는 의도보다 넓거나 좁게 얻어질 수 있는데 이때 최적의 경계값을 사용하면서 잘못 분리된 영역은 후처리 과정을 통해서 결과를 수정하기 위한 기법

압축 : 이미지에 대해서 필터를 슬라이드하면서 필터의 픽셀값이 전부 1(백)일 경우에만 1을 출력하고 그렇지 않으면 0(흑)을 출력하는 처리



팽창 : 압축과 반대되는 처리, 필터의 픽셀값이 하나라도 1(백)이면 1을 출력





오프닝 : 압축 후 팽창

-밝은 영역을 전체적으로 축소한 후 팽창 연산을 뒤이어 수행하여 전체적인 넓이를 원래대로 복구

-밝은 영역에 나타난 미세한 조각을 제거할 수 있도록 하는 연산

클로징 : 팽창 후 압축

-밝은 영역을 넓히고 다시 침식 연산을 수행

-밝은 영역에 생긴 미세한 틈을 메우는 역할

2-2-2 히스토그램 작성

픽셀값 분포를 그래프 형태로 표현

-픽셀과 수의 분포에 대해 쉽게 파악할 수 있음

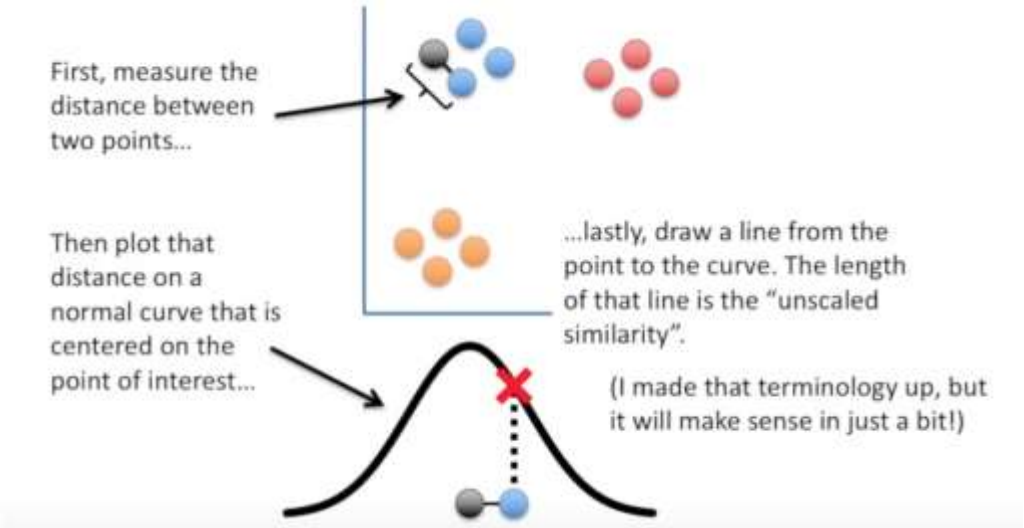
-가로축에 픽셀값, 세로축에 해당 픽셀값을 가지는 픽셀수로 표현

2-2-3 PCA에 의한 차원압축

PCA(주성분 분석)을 통해 이미지 데이터의 차원을 압축하고 특징량을 추출할 수 있음

2-2-4 t-SNE에 의한 차원압축

t-SNE: 데이터 사이의 거리를 확률분포로 표현



-검은 점 하나 선택

-검은 점과 다른 점까지의 거리 측정

-T분포 그래프를 이용하여 검정 점을 T분포 상의 가운데에 위치한다면 기준점으로부터 상대점까지 거리에 있는 T분포의 값(빨간색 X표시의 값)을 선택하여, 이 값을 친밀도(Similarity)라고 하고 이 친밀도가 가까운 값끼리 묶는다.

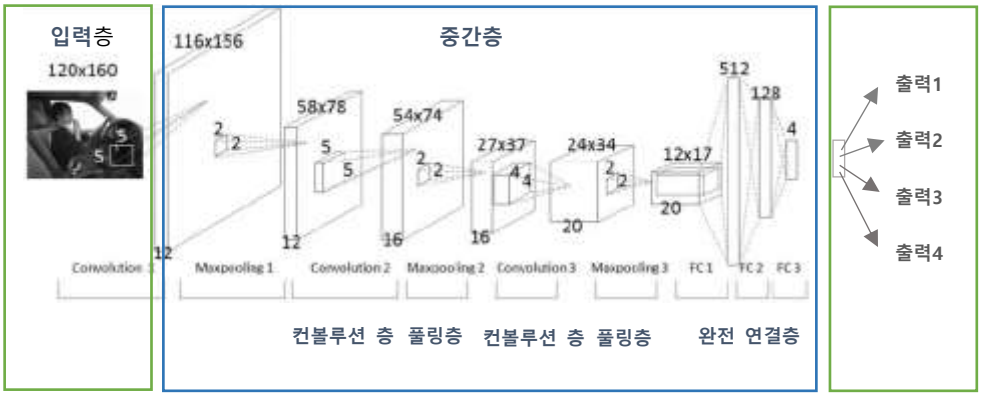
-차원압축 전후, 확률분포의 KL-정보량이 최소가 되는 압축 후의 데이터점을 계산(확률분포의 차이가 최소가 되는 데이터 점)

* KL-정보량: 두개의 확률분포 차이를 측정하는 지표

3. 딥러닝을 위한 데이터 준비

이미지를 대상으로 한 분류 모델에서는 딥러닝 알고리즘 중 CNN을 사용하는 것이 높은 성능을 낼 수 있다.

3-1. CNN의 구조



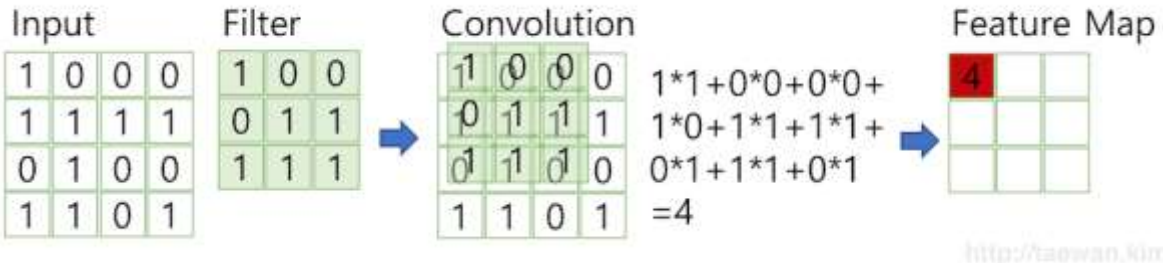
3-1-2. 중간층

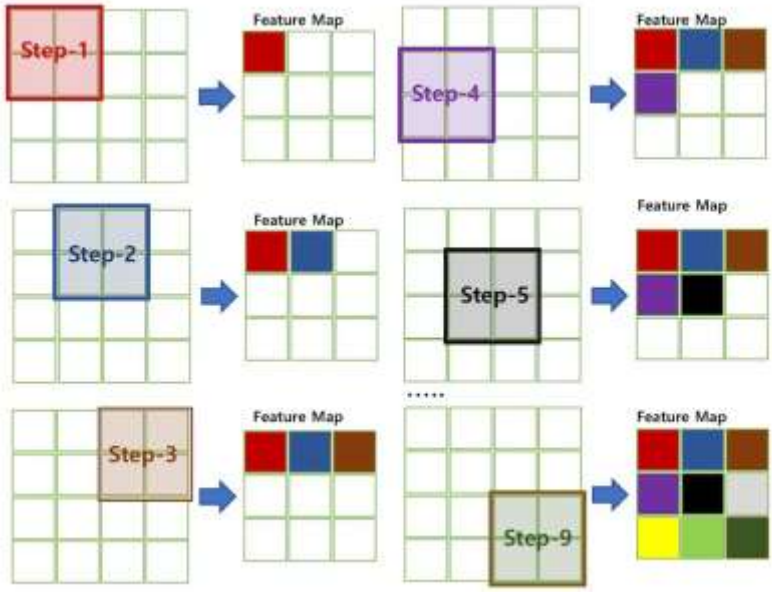
컨볼루션 층 : 입력 데이터에 필터를 적용해 그 데이터가 가지고 있는 특징량을 추출

-가중치가 있는 필터를 슬라이드하면서 적용

-노드값과 가중치를 곱하고 그 결과들을 더해서 특징량을 추출해나감

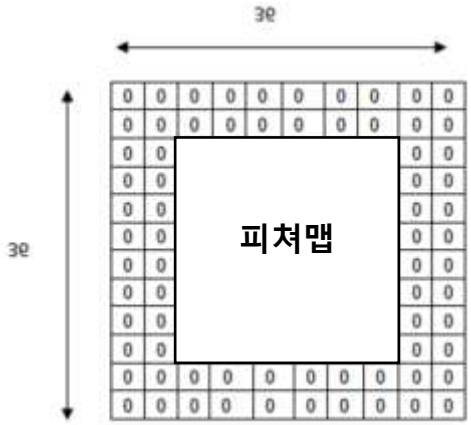
-추출된 특징량을 맵이라고 함





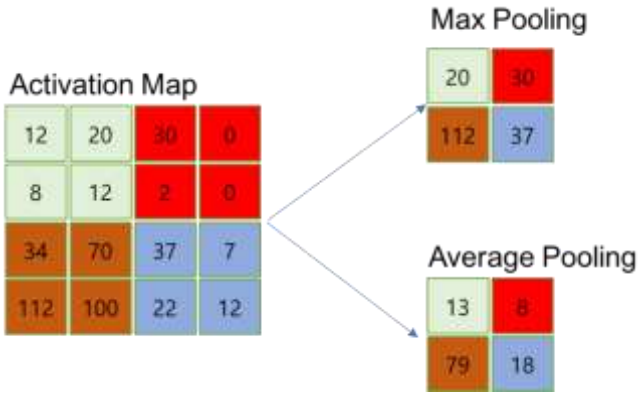
-컨볼루션 연산을 하고 난 출력인 피쳐맵의 크기는 입력데이터보다 작기 때문에 패딩을 하여 컨볼루션 레이어의 출력 데이터가 줄어드는 것을 방지하기 위해 패딩을 함.

-패딩 : 데이터의 외각에 지정된 크기 만큼 특정 값으로 채워 넣는 것



풀링층 : 컨볼루션 레이어의 출력을 입력으로 받아서 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용

1. 맥스 풀링 : 특정 영역 안에 값의 최댓값을 모음
2. 평균 풀링 : 특정 영역 안에 값의 평균을 모음



입력데이터는 중간층에서 컨볼루션 층과 풀링층을 차례로 거치면서 중요한 특징량만 남도록 압축되고 결국엔 완전연결층에 도달

3-2. 데이터 세트 작성

딥러닝의 입력층 형태는 2차원 배열이므로 데이터 세트의 형태도 이와 동일하게 2차원 배열로 만들어준다.

3-2-1. 이미지의 설명변수와 목적변수의 세트 작성

```
pixels = np.array(pixels)/255
pixels = pixels.reshape([-1,128,128,1)
labels = np.array(labels)
```

➔ 픽셀값을 Numpy 배열로 변환하고 255로 나누어서 정규화함

➔ reshape를 사용해 픽셀값을 1차원 배열에서 128 X 128의 2차원 배열로 변환

3-2-2. 데이터 세트의 분할

데이터 세트를 훈련데이터와 테스트용 데이터로 분할한다.

```
from sklearn import model_selection

trainX, testX, trainY, testY =
    model_selection.train_test_split(pixels, labels, test_size = 0.2)
```

➔ model_selection에 포함되어 있는 train_test_split함수를 사용해 80:20의 비율로 훈련데이터와 테스트 데이터로 나눔

3-3. 데이터 수 증가

딥러닝에서 모델을 작성하는 경우 학습에는 좀 더 많은 이미지가 필요하다. 사용할 수 있는 이미지가 적을 때 원래 있던 이미지를 조금 변형해서 이미지 수를 늘린다.

3-3-1. 이미지 반전

원래의 이미지를 상하, 좌우, 상하좌우로 반전시켜 이미지 수를 늘린다.

이미지 반전을 통한 이미지 수 증가
img = cv2.imread('불러올 데이터 경로',0) x_img = cv2. flip(img, 0) y_img = cv2.flip(img, 1) z_img = cv2.flip(img, -1) cv2.imwrite('x_img.jpg', x_img) cv2.imwrite('y_img.jpg', y_img) cv2.imwrite('z_img.jpg', z_img)
→flip을 사용해 두번째 인자에 0을 지정한 뒤 x축을 중심으로 이미지를 반전시킨다.(상하반전) →flip을 사용해 두번째 인자에 1을 지정한 뒤 y축을 중심으로 이미지를 반전시킨다.(좌우반전) →flip을 사용해 두번째 인자에 -1을 지정한 뒤 x축과 y축 중심으로 이미지를 반전시킨다.(상하좌우반전)




3-3-2. 이미지의 블러 처리

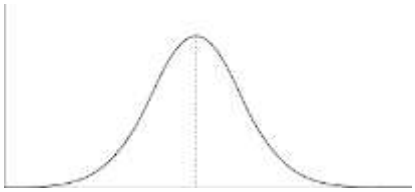
블러 : 이미지를 흐릿하게 만드는 것

원래의 이미지에 블러 처리를 해서 이미지 수를 증가

1. Averaging : 필터 박스내의 평균을 이용한 방법

2. Gaussian Blur : 2차원 가우시안 분포를 이용한 방법

-박스내의 픽셀값을 가우시안 분포의 가중치로 평균을 내어 현재 픽셀값을 업데이트 해주는 방법



3. Median Blurring : 중간값을 이용한 방법

-현재 픽셀값을 박스내의 픽셀 값들의 중간값으로 대체한다.

-박스내 픽셀값들을 크기순으로 나열하여 중간값을 고르고 이것을 현재 픽셀값으로 주는 것

블러 처리를 통한 이미지 수 증가
blur_img = cv2. blur(img, (5,5)) gau_img = cv2.GaussianBlur(img, (5,5), 0) med_img = cv2.medianBlur(img, 5) cv.imwrite('blur_img.jpg', blur_img) cv.imwrite('gau_img.jpg', gau_img) cv.imwrite('med_img.jpg', med_img)
→5X5 크기의 필터를 준비해서 컨볼루션 연산을 한다. ➔ 가중치가 균일한 필터를 사용해서 영역 내의 픽셀수 평균을 계산한다. ➔ 주목하는 픽셀과의 거리에 따라서 가우스분포(정규분포)에 따라 가중치를 부여하고, 픽셀값의평균을 계산한다. ➔ 영역 내의 픽셀값의 중앙치를 계산한다.

3-3-3. 이미지의 명도 변경

이미지의 명도를 변경해서 이미지 수를 늘린다.

감마보정 :

Output = Input^gamma

-감마값이 1을 기준으로 높으면 어두워지고

-낮으면 밝아지는 구조

-openCV에서는 단순히 원래 화소에 감마값을 곱해버리면 오버플로우가 발생할 수도 있다.

-openCV에서 감마보정을 위해 정규화를 하고 (모든 픽셀값에 255를 나눠줌) (1/감마)만큼을 제곱

-(1/감마)를 제곱하는 이유 →값이 높아질수록 밝기가 같이 올라가도록 하기위해

-다시 255를 곱해서 원래의 형식으로 바꿔준다.

명도 변경을 통한 이미지 수 증가

```
gamma = 0.5

lut = np.zeros((256, 1) , dtype = 'unit8')
for I in range(len(lut)):
    lut[i][0] = 255 * pow((float(i)/255, (1.0/gamma))

gamma_img = cv2.LUT(img, lut)

cv2.imwrite('gamma_img/jpg' , gamma_img)
```

➔ 명도를 조정하는 계수를 설정
➔ 명도의 조정 결과를 저장할 배열을 작성
➔ LUT를 사용해서 이미지를 보정하여 명도를 변경

*LUT : 모든 픽셀의 경우에 대해서 미리 계산을 해서 LUT에 저장→ 연산이 빨라짐

ex)

LUT사용 X → 픽셀을 돌면서 밝기가 100인 픽셀 300개가 있다면 100개 각각마다 +10을 해서 110의 결과를 낸다.

LUT사용 O → 픽셀을 돌기 전에 밝기가 100일 경우에 110으로 리턴 할 수 있도록 한다.