준비된 이미지를 읽어 보겠습니다. 구글드라이버의 data 폴더에 개미와 벌의 이미지를 10장씩 복사해주세요.

```
import cv2
```

```
import os
import numpy as np
dir = ['ants', 'bees']
pixels = []
labels = []
for i, d in enumerate(dir):
  files = os.listdir('./drive/MyDrive/data/' + d)
  for f in files:
    img = cv2.imread('./drive/MyDrive/data/' + d + '/' + f, 0)
    img = cv2.resize(img, (128, 128))
    img = np.array(img).flatten().tolist()
    pixels.append(img)
    labels.append(i)
```

그리고 위 코드를 실행시킵니다. 이미지 20장을 읽을 때, 이미지의 크기를 128X128로 축소하고 1차원으로 자료로 변환합니다. 그러면 1줄로 16384열을 가지는 자료가 되겠죠? 이 데이터를 pixels에 추가합니다. labels에는 0이 개미, 1이 벌인 수치가 추가됩니다.

아래 코드는 위에서 작성된 넘파이 배열을 판다스 데이터프레임으로 변경해줍니다. pixels_df는 pixel과 마찬가지로 16384열을 가지는데, 여기에 labels_df를 마지막 컬럼으로 추가하여 img_set를 만듭니다.

```
import pandas as pd
```

```
pixels_df = pd.DataFrame(pixels)
pixels_df = pixels_df/255
labels_df = pd.DataFrame(labels)
labels_df = labels_df.rename(columns={0: 'label'})
img_set = pd.concat([pixels_df, labels_df], axis=1)
img_set.head()
```

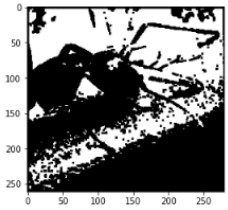
	0	1	2	3	4	5	6	7	8	9	...	16375	16376
0	0.662745	0.674510	0.674510	0.678431	0.670588	0.682353	0.803922	0.800000	0.803922	0.803922	...	0.400000	0.458824
1	0.631373	0.670588	0.662745	0.619608	0.635294	0.647059	0.678431	0.690196	0.686275	0.698039	...	0.227451	0.223529
2	0.949020	0.952941	0.956863	0.952941	0.952941	0.949020	0.956863	0.960784	0.960784	0.952941	...	1.000000	1.000000
3	0.501961	0.478431	0.490196	0.509804	0.513725	0.509804	0.505882	0.509804	0.525490	0.541176	...	0.466667	0.556863
4	0.650980	0.650980	0.658824	0.666667	0.666667	0.647059	0.662745	0.678431	0.650980	0.643137	...	0.717647	0.729412

5 rows × 16385 columns

이렇게 해서 20장의 이미지에 대한 데이터 테이블을 얻을 수 있습니다.

```
import matplotlib.pyplot as plt
# 이미지 침식(erode) 처리
img = cv2.imread(
    './drive/MyDrive/data/ants/swiss-army-ant.jpg',0)
ret, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
kernel = np.ones((3,3), np.uint8)
img_el = cv2.erode(bin_img, kernel, iterations=1)
plt.imshow(img_el, cmap='gray')
```

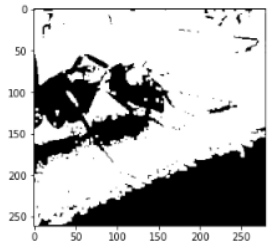
```
<matplotlib.image.AxesImage at 0x7fc1bd72cd90>
```



위 코드는 이미지를 침식(erode)하는 과정을 보여줍니다. 필터로는 3X3의 정사각형을 사용합니다. 필터내의 값이 모두 255에 해당될 때 필터의 중앙값을 255로 살립니다. 그렇지 않으면 모두 0으로 변경합니다. 이렇게 하는 이유는 노이즈를 제거하기 위해서입니다.

```
[40] # 이미지의 팽창(dilate) 처리
     img_dl = cv2.dilate(bin_img, kernel, iterations=1)
     plt.imshow(img_dl, cmap='gray')
```

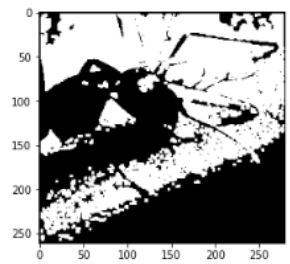
```
<matplotlib.image.AxesImage at 0x7fc1bd1f4790>
```



이미지를 팽창(dilate)시켜 노이즈를 제거할 수도 있습니다. 흰색을 확장하는 거죠.

```
# 침식 후 팽창
img_op = cv2.morphologyEx(bin_img,
                          cv2.MORPH_OPEN, kernel)
plt.imshow(img_op, cmap='gray')
```

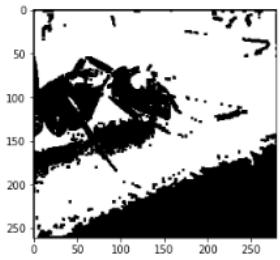
```
<matplotlib.image.AxesImage at 0x7fc1bd174590>
```



침식 후 팽창을 하는 경우도 있습니다. OpenCV의 morphologyEx 함수를 사용합니다.

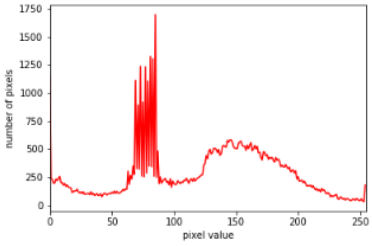
```
# 팽창 후 침식
img_cl = cv2.morphologyEx(bin_img,
                          cv2.MORPH_CLOSE, kernel)
plt.imshow(img_cl, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7fc1bd0dbbd0>
```



팽창후 침식을 하는 경우도 있구요.

```
hist_gr, bins = np.histogram(img.ravel(), 255, [0,256])
plt.xlim(0,255)
plt.plot(hist_gr, '-r')
plt.xlabel('pixel value')
plt.ylabel('number of pixels')
plt.show()
```



히스토그램을 그려서 이미지 픽셀수가 많은 픽셀 구간을 확인할 수도 있습니다.

t-SNE 분석으로 이미지의 컬럼수를 대폭 축소할 수도 있습니다.

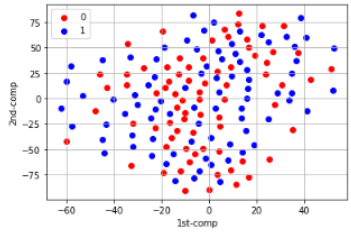
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
pixels_tsne = tsne.fit_transform(pixels_df)
print(pixels_df.shape)
print(pixels_tsne.shape)
img_set_tsne = pd.concat([pd.DataFrame(pixels_tsne),
                          labels_df], axis=1)
img_set_tsne.head()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will
  FutureWarning,
(157, 16384)
(157, 2)
```

	0	1	label
0	11.475740	-84.330757	0
1	-16.928295	16.541561	0
2	-10.026407	-90.761909	0
3	-3.587237	-16.084433	0
4	-5.901059	-54.740585	0

산포도를 그려볼 수도 있습니다.

```
# 컬럼별 산포도
img_set_tsne_0 = img_set_tsne[img_set_tsne['label'] == 0]
img_set_tsne_0 = img_set_tsne_0.drop('label', axis=1)
plt.scatter(img_set_tsne_0[0], img_set_tsne_0[1], c='red'
, label=0)
img_set_tsne_1 = img_set_tsne[img_set_tsne['label'] == 1]
img_set_tsne_1 = img_set_tsne_1.drop('label', axis=1)
plt.scatter(img_set_tsne_1[0], img_set_tsne_1[1], c='blue'
, label=1)
plt.xlabel('1st-comp')
plt.ylabel('2nd-comp')
plt.legend()
plt.grid()
plt.show()
```



다음 코드는 1차원배열이던 20장의 이미지를 128X128의 2차원으로 변경합니다.

```
pixels = np.array(pixels)/255
pixels = pixels.reshape([-1, 128, 128, 1])
labels = np.array(labels)
print(pixels[0].shape)
print(labels[0])
```

```
(128, 128, 1)
0
```

그리고 훈련셋과 테스트셋을 구합니다. 80:20의 분포를 가지도록 합니다.

```
from sklearn import model_selection
trainX, testX, trainY, testY = model_selection.train_test_split(
    pixels, labels, test_size=0.2)
print(len(trainY))
print(len(testY))
```

```
125
32
```

이렇게 해서 이미지 전처리가 완료되었습니다만, 될 수 있으면 많은 이미지를 훈련하는 것이 좋은 결과를 가지고 옵니다. 그런데 일반적으로 보면 이미지 데이터가 부족한 것이 사실이죠.

```
# 이미지 반전으로 이미지 수 증가
img = cv2.imread('./drive/MyDrive/data/ants/swiss-army-ant.jpg'
, 0)
x_img = cv2.flip(img, 0)
y_img = cv2.flip(img, 1)
xy_img = cv2.flip(img, -1)
cv2.imwrite('./drive/MyDrive/data/ants/x_img.jpg', x_img)
cv2.imwrite('./drive/MyDrive/data/ants/y_img.jpg', y_img)
cv2.imwrite('./drive/MyDrive/data/ants/xy_img.jpg', xy_img)
```

```
True
```

위 코드는 이미지를 반전시켜 또다른 이미지를 만들어 냅니다. x, y 축으로 각각 반전시키는가하면 xy축을 기준으로 회전시킨 이미지도 생성합니다.

```
# 블러 처리(흐릿하게)로 이미지 증가
blur_img = cv2.blur(img, (5,5))
gau_img = cv2.GaussianBlur(img, (5,5), 0)
med_img = cv2.medianBlur(img, 5)
cv2.imwrite('./drive/MyDrive/data/ants/blur_img.jpg', blur_img)
cv2.imwrite('./drive/MyDrive/data/ants/gau_img.jpg', gau_img)
cv2.imwrite('./drive/MyDrive/data/ants/med_img.jpg', med_img)
```

```
True
```

또한 블러 처리로 이미지를 흐릿하게 하여 이미지를 추가할 수도 있습니다.

```
gamma = 0.5
lut = np.zeros((256,1), dtype = 'uint8')
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('./drive/MyDrive/data/ants/gamma_img.jpg', gamma_img)
```

```
True
```

또한 명도를 변경하여 이미지를 생성할 수도 있습니다.