


## 군집 알고리즘

 [구글 코랩에서 실행하기](#)

### 과일 사진 데이터 준비하기

```
In [1]: !wget https://bit.ly/fruits_300_data -O fruits_300.npy
# 사과, 바나나, 파인애플을 담고 있는 흑백사진 데이터 다운로드
# npy는 numpy 배열의 기본 저장 포맷
# wget 명령은 원격 주소에서 데이터를 다운로드/저장
# -O 옵션에서 저장할 파일 이름을 지정 가능

--2023-10-28 12:44:17-- https://bit.ly/fruits_300_data
Resolving bit.ly (bit.ly)... 67.199.248.10, 67.199.248.11
Connecting to bit.ly (bit.ly)[67.199.248.10]:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://github.com/rickiepark/hg-mlcl/raw/master/fruits_300.npy [following]
--2023-10-28 12:44:17-- https://github.com/rickiepark/hg-mlcl/raw/master/fruits_300.npy
Resolving github.com (github.com)... 140.82.112.3
Connecting to github.com (github.com)[140.82.112.3]:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/rickiepark/hg-mlcl/master/fruits_300.npy [following]
--2023-10-28 12:44:18-- https://raw.githubusercontent.com/rickiepark/hg-mlcl/master/fruits_300.npy
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)[185.199.108.133]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3080128 (2.9M) [application/octet-stream]
Saving to: 'fruits_300.npy'

fruits_300.npy      100%[=====]  2.86M  --.-KB/s   in 0.06s

2023-10-28 12:44:19 (51.8 MB/s) - 'fruits_300.npy' saved [3080128/3080128]
```

```
In [2]: import numpy as np
import matplotlib.pyplot as plt
```

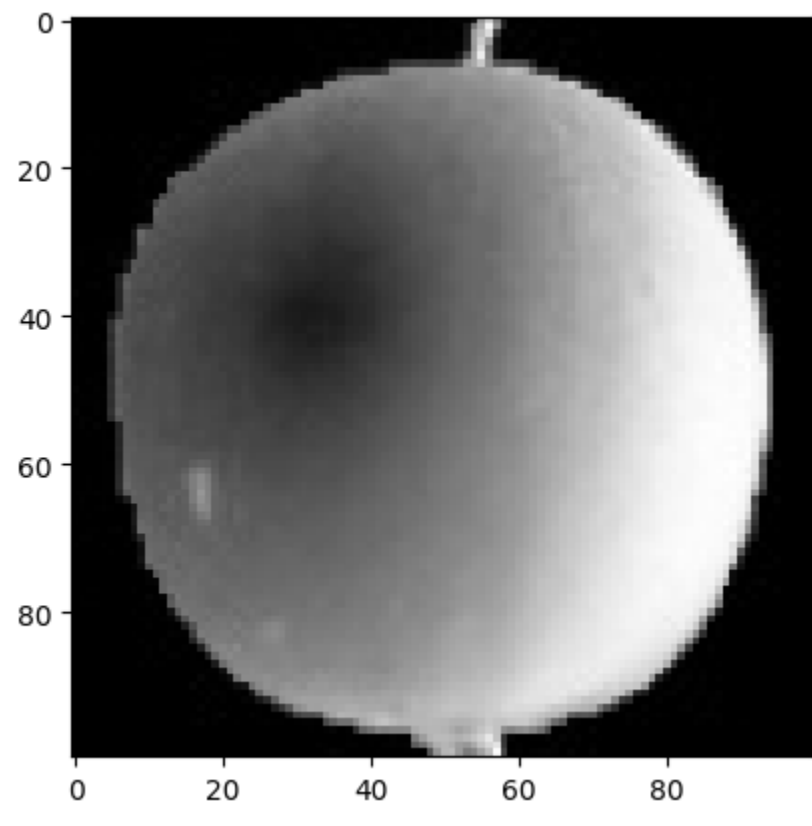
```
In [3]: fruits = np.load('fruits_300.npy')
# fruits_300.npy 파일에 있는 모든 데이터를 fruits에 할당
```

```
In [4]: print(fruits.shape) # 300은 샘플, 100은 이미지 높이, 100은 이미지 너비
(300, 100, 100)
```

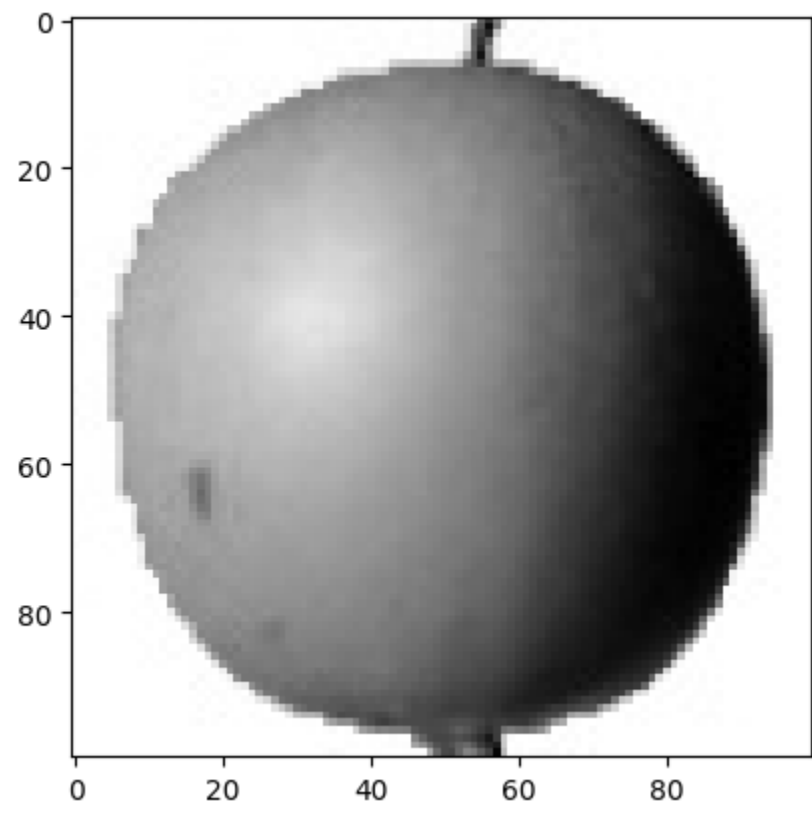
```
In [5]: print(fruits[0, 0, :])
```

```
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  1
  2  2  2  2  2  2  1  1  1  1  1  1  1  1  1  1  2  3  2  1
  2  1  1  1  2  1  3  2  1  3  1  4  1  2  5  5  5  5
18 148 192 117 28  1  1  2  1  4  1  3  1  1  1  1  1
  2  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1]
```

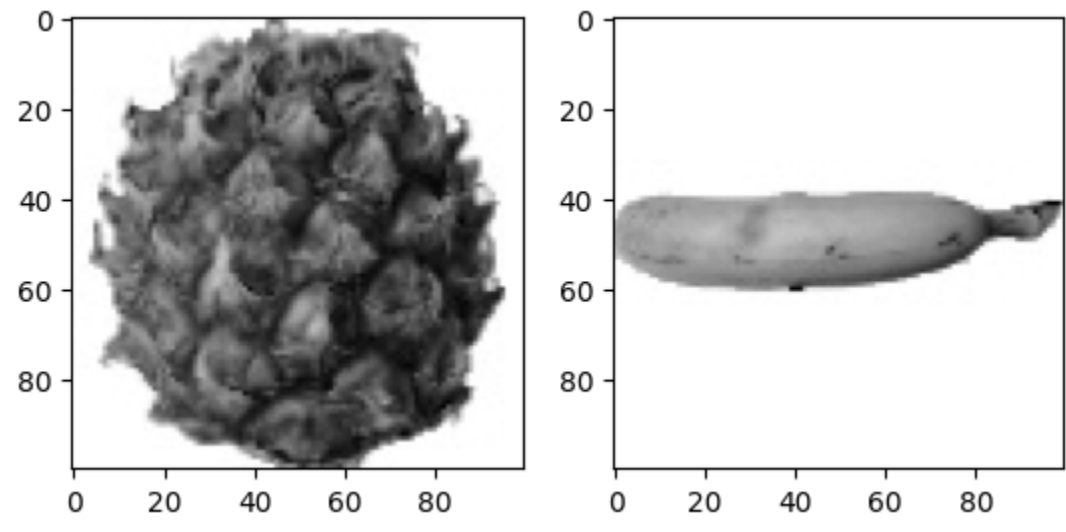
```
In [6]: plt.imshow(fruits[0], cmap='gray') # 흑백 이미지이므로 cmap매개변수를 'gray'로 지정
plt.show() # 0에 가까운 배열 값일수록 검게 나타나고 높은 값은 밝게 표시됨
```



```
In [7]: plt.imshow(fruits[0], cmap='gray_r') # cmap 매개변수를 'gray_r'로 지정하여 색을 반전시킴
plt.show()
```



```
In [8]: fig, axs = plt.subplots(1, 2)
axs[0].imshow(fruits[100], cmap='gray_r')
axs[1].imshow(fruits[200], cmap='gray_r')
plt.show()
# 사과, 바나나, 파인애플이 각각 100개씩 들어있으므로 각 인덱스에 맞춰서 파이나
```



### 픽셀 값 분석하기

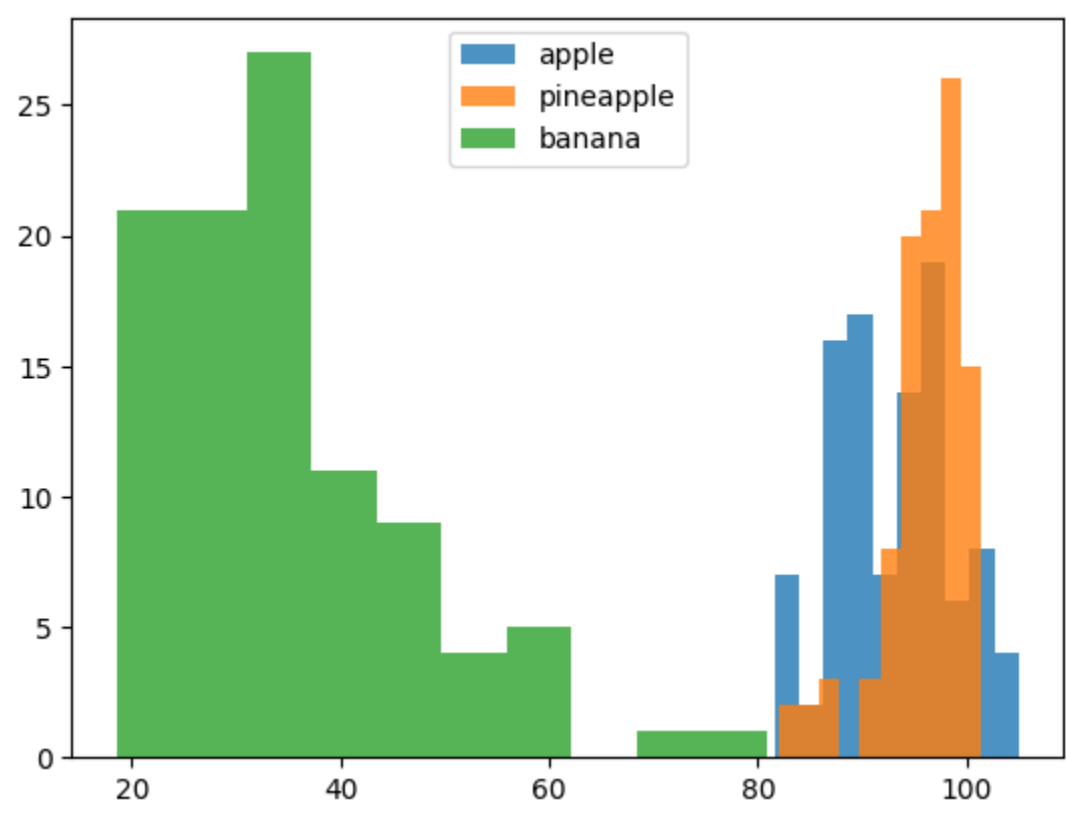
```
In [9]: apple = fruits[0:100].reshape(-1, 100*100)
pineapple = fruits[100:200].reshape(-1, 100*100)
banana = fruits[200:300].reshape(-1, 100*100)
# 각 과일의 픽셀을 분석하기 위해, 100 x 100 이미지를 펼쳐서 길이가 10,000인 1차원 배열로 만들
```

```
In [10]: print(apple.shape)
(100, 10000)
```

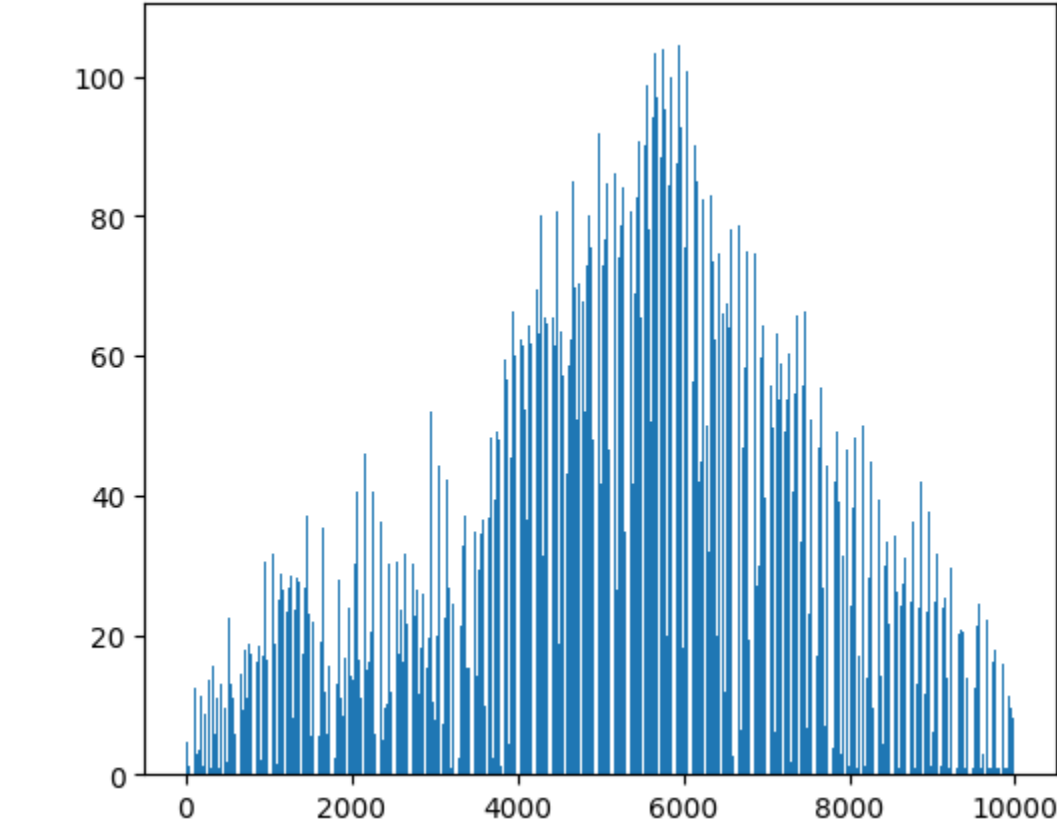
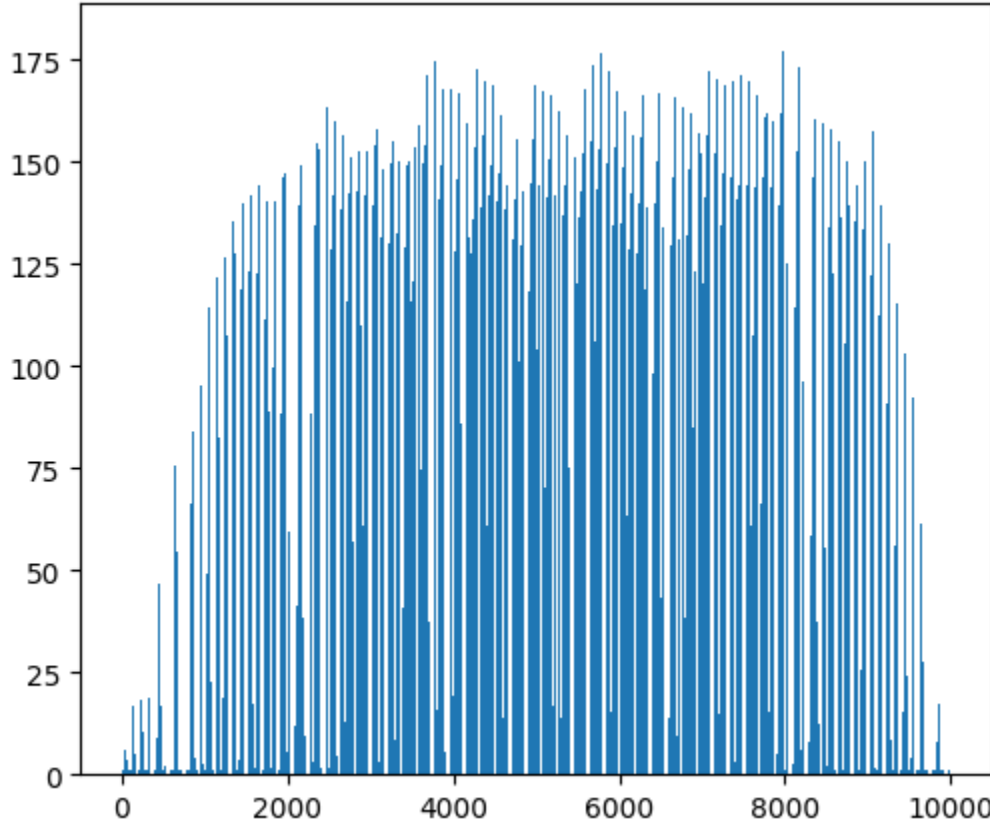
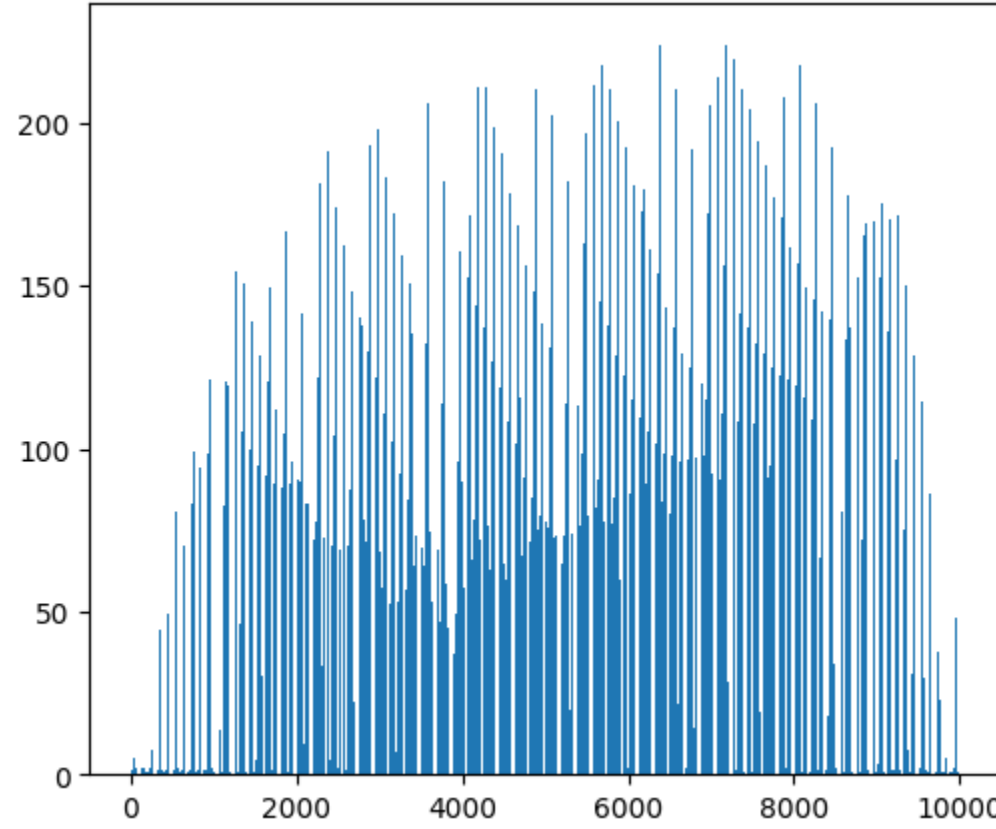
```
In [11]: print(apple.mean(axis=1)) # 두번째 축인 열을 따라 계산
```

```
[ 88.3346  97.9249  87.3709  98.3703  92.8705  82.6439  94.4244  95.5999
 90.691  81.6226  87.0578  95.0745  93.8416  87.017  97.5978  87.2019
 88.9827 100.9158  92.7823 100.9184 104.9954  88.674  99.5643  97.2495
 94.1179  92.1935  95.1671  93.3222 102.8967  94.6695  90.5285  89.0744
 97.7641  97.2938 100.7504  90.5236 100.2542  85.0452  96.4635  97.1492
 90.731  102.3183  87.1529  89.8751  86.7327  86.3901  95.2865  89.1709
 96.8163  91.6604  96.1065  99.6829  94.9718  87.4812  89.2596  89.5268
 93.799  97.3983  87.151  97.625 103.22  94.4239  83.6657  83.5159
102.8453  87.0379  91.2742 100.4848  93.8388  90.0568  97.4616  97.5022
 82.446  87.1789  96.9286  90.3135  90.565  97.6538  98.0919  93.6252
 87.3867  84.7073  89.1135  86.7646  88.7301  86.643  96.7323  97.2604
 81.9424  87.1687  97.2866  83.4712  95.9781  91.8096  98.4086 100.7823
101.556 100.7827  91.6088  88.8976]
```

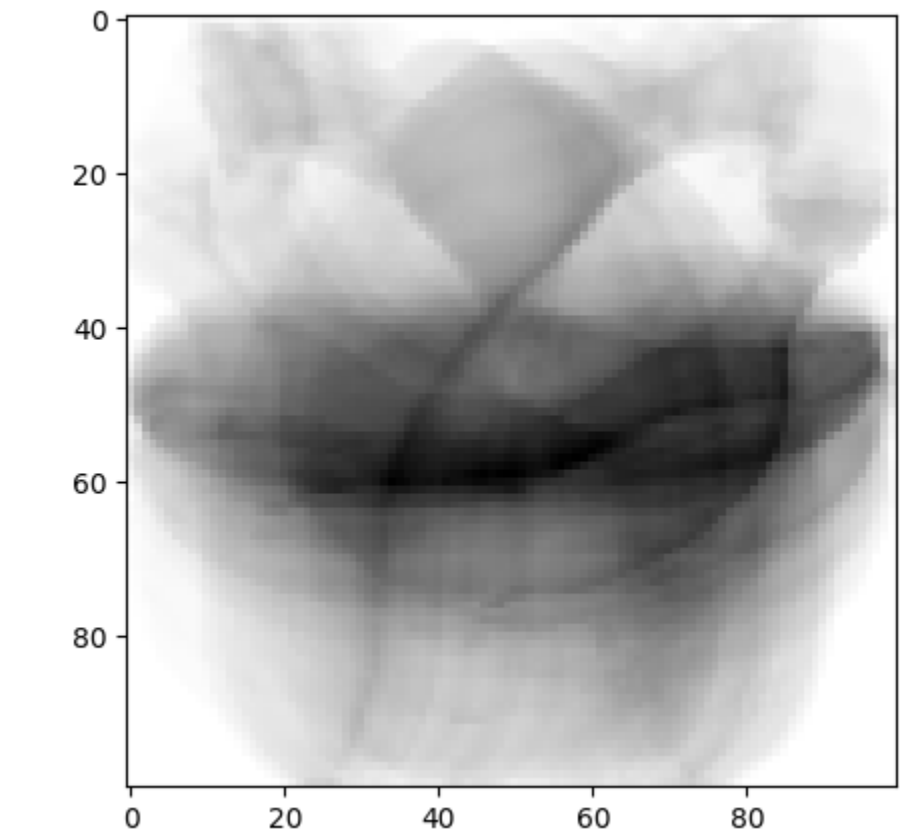
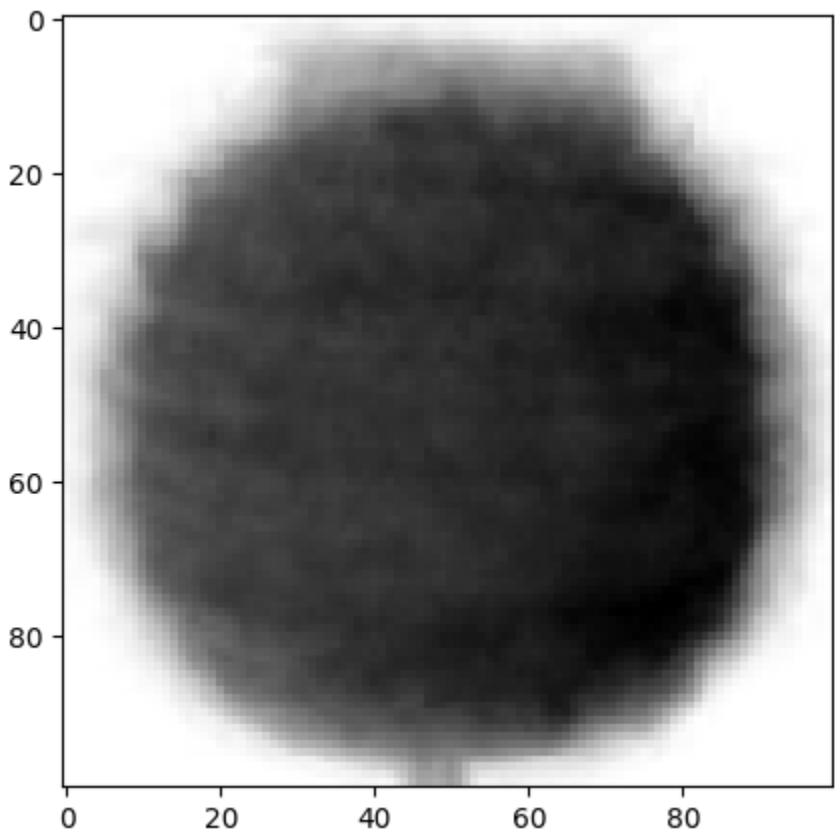
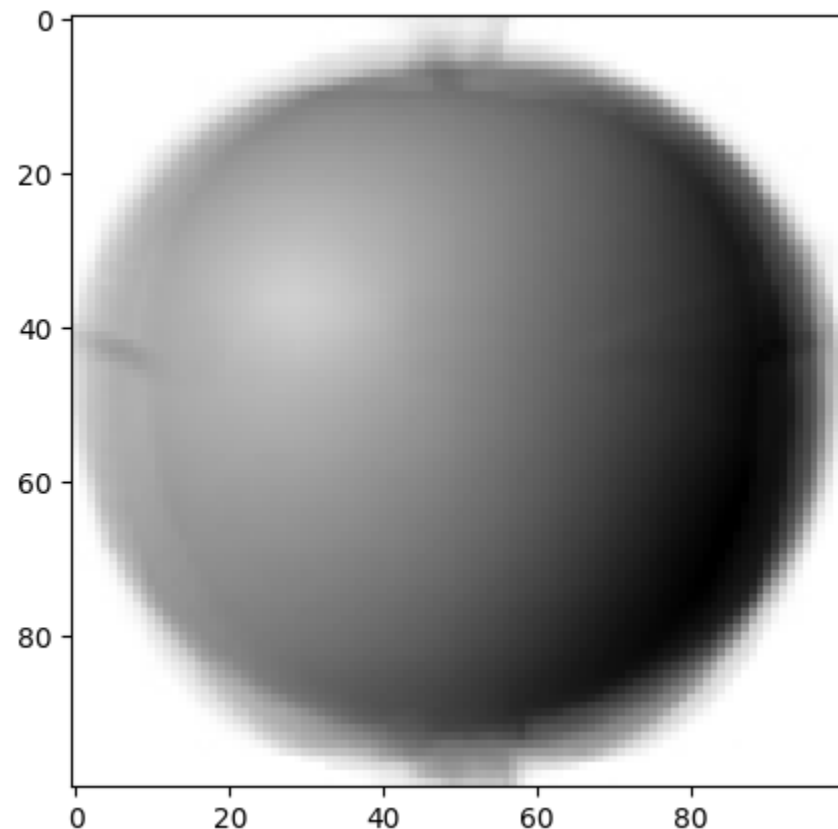
```
In [12]: plt.hist(np.mean(apple, axis=1), alpha=0.8)
plt.hist(np.mean(pineapple, axis=1), alpha=0.8)
plt.hist(np.mean(banana, axis=1), alpha=0.8)
plt.legend(['apple', 'pineapple', 'banana'])
plt.show()
# 히스토그램을 통해서 각 과일의 평균값이 어떻게 분포되어 있는지 출력
# alpha를 1보다 작게해서 투명도를 줄 수 있음
```



```
In [13]: fig, axs = plt.subplots(1, 3, figsize=(20, 5))
axs[0].bar(range(10000), np.mean(apple, axis=0))
axs[1].bar(range(10000), np.mean(pineapple, axis=0))
axs[2].bar(range(10000), np.mean(banana, axis=0))
plt.show()
# 이미지마다 값이 높은 구간이 다름
# 사과는 사진 아래쪽으로 갈수록 값이 높아짐
# 파인애플은 전체적으로 고르고 높음
# 바나나는 중앙의 픽셀값이 높음
```



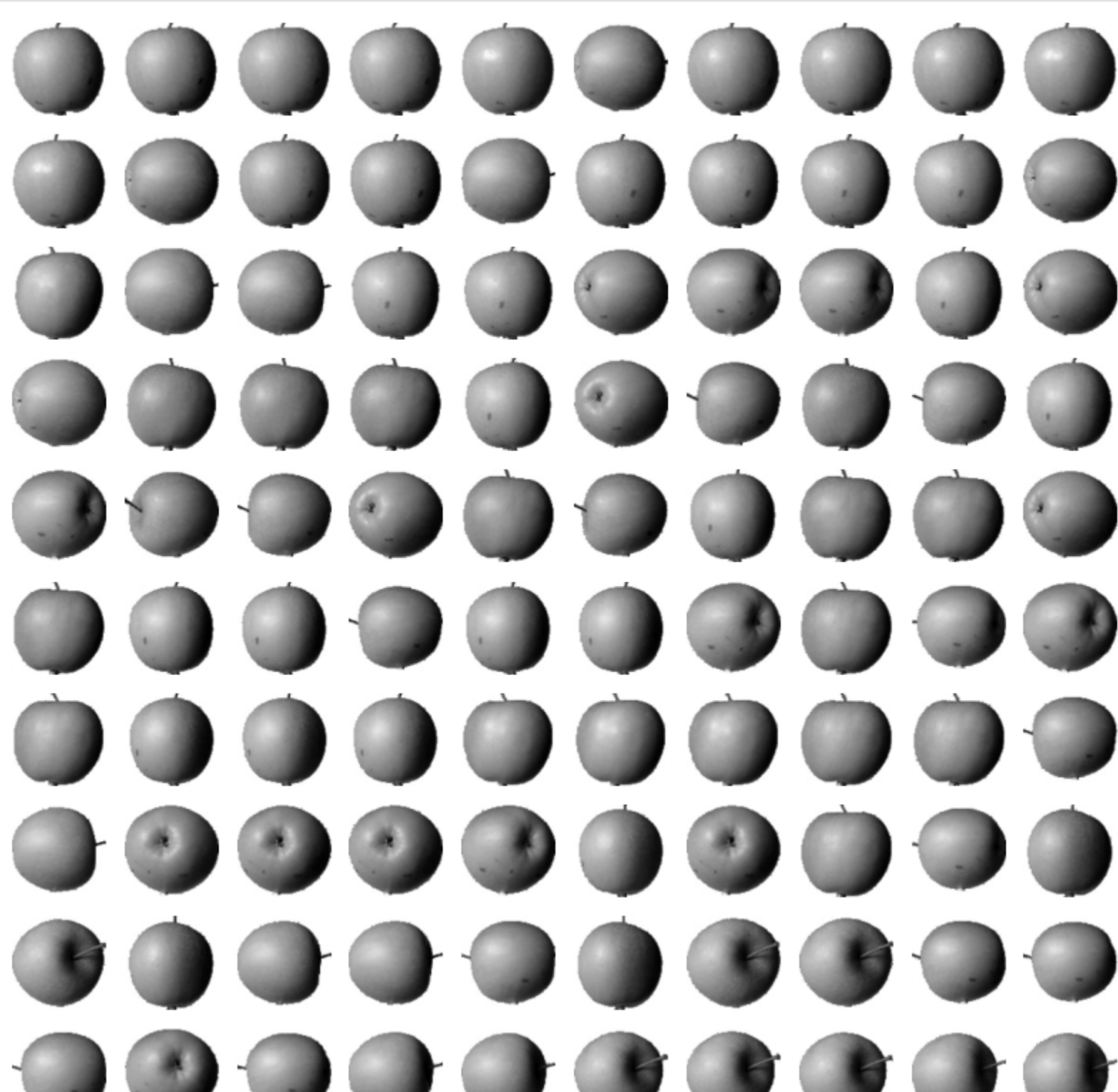
```
In [14]: apple_mean = np.mean(apple, axis=0).reshape(100, 100)
pineapple_mean = np.mean(pineapple, axis=0).reshape(100, 100)
banana_mean = np.mean(banana, axis=0).reshape(100, 100)
# 해당 평균값을 100 x 100 크기인 이미지로 만들
fig, axs = plt.subplots(1, 3, figsize=(20, 5))
axs[0].imshow(apple_mean, cmap='gray_r')
axs[1].imshow(pineapple_mean, cmap='gray_r')
axs[2].imshow(banana_mean, cmap='gray_r')
plt.show()
```



### 평균값과 가까운 사진 고르기

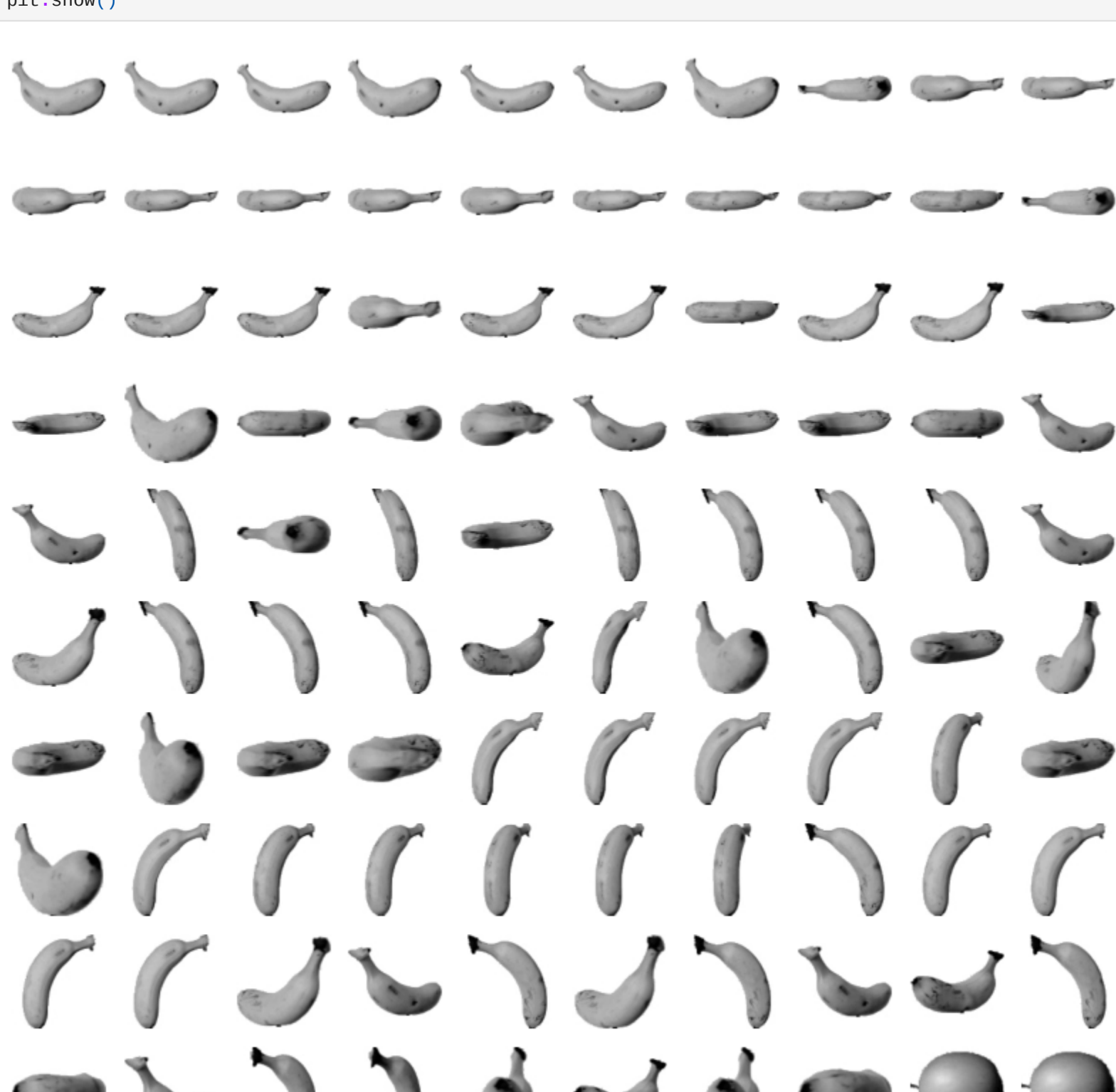
```
In [15]: abs_diff = np.abs(fruits - apple_mean)
abs_mean = np.mean(abs_diff, axis=(1,2))
print(abs_mean.shape)
(300,)
```

```
In [16]: apple_index = np.argsort(abs_mean)[:100]
# np.argsort를 통해 차이가 가장 작은 순서대로 100개의 데이터를 선정
fig, axs = plt.subplots(10, 10, figsize=(10,10))
for i in range(10):
    for j in range(10):
        axs[i, j].imshow(fruits[apple_index[i*10 + j]], cmap='gray_r')
        axs[i, j].axis('off')
plt.show()
```



### 확인문제

```
In [17]: abs_diff = np.abs(fruits - banana_mean)
abs_mean = np.mean(abs_diff, axis=(1,2))
# fruits 배열에 있는 모든 샘플에서 banana_mean를 뺀 절대값의 평균을 계산
# axis(1,2)를 통해 1,2축을 따라서 배열의 산술 평균을 계산함
banana_index = np.argsort(abs_mean)[:100] # 100개 선정
fig, axs = plt.subplots(10, 10, figsize=(10,10))
for i in range(10):
    for j in range(10):
        axs[i, j].imshow(fruits[banana_index[i*10 + j]], cmap='gray_r')
        axs[i, j].axis('off')
plt.show()
```



```
In [ ]:
```