



# 인공신경망과딥러닝입문

## Lecture 13. 군집 알고리즘

동덕여자대학교  
데이터사이언스 전공  
권 범

# 목차

- ❖ 01. 과일 사진 데이터 준비하기
- ❖ 02. 픽셀값 분석하기
- ❖ 03. 평균값과 가까운 사진 고르기
- ❖ 04. 비슷한 샘플끼리 모으기
- ❖ 05. 마무리

# 01. 과일 사진 데이터 준비하기

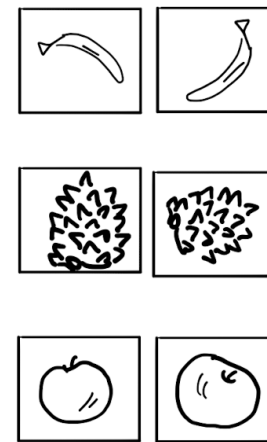
- 02. 픽셀값 분석하기
- 03. 평균값과 가까운 사진 고르기
- 04. 비슷한 샘플끼리 모으기
- 05. 마무리

# 01. 과일 사진 데이터 준비하기

## ❖ 시작하기 전에

### 상황 가정

- ✓ 동덕 마켓은 새 이벤트를 기획하고 있음
- ✓ 고객이 동덕 마켓에서 사고 싶은 과일 사진을 보내면  
그중에서 가장 많이 요청된 과일을 판매 품목으로 선정하려고 함
- ✓ 그런데 고객이 보낸 사진을 사람이 하나씩 분류하기는 어려울 것임
- ✓ 그렇다고 생선처럼 미리 과일 분류기를 학습시키에는  
고객들이 어떤 과일 사진을 보낼지 알 수 없으니 곤란함



← 이렇게 과일 사진을  
자동으로 묶을 수 있죠?

# 01. 과일 사진 데이터 준비하기

## ❖ 타겟을 모르는 비지도 학습

- 이번 수업 시간에는 타겟(레이블)을 모르는 과일 사진을 종류별로 분류하려고 함
- 이렇게 타겟이 없을 때 사용하는 머신러닝 알고리즘이 있음
- 바로 비지도 학습(Unsupervised Learning)임
- 사람이 가르쳐 주지 않아도 데이터에 있는 무언가를 학습함

타겟이 없는 상태에서  
어떻게 과일 사진을 종류별로 분류할 수 있을까?

- ✓ 사진의 픽셀값을 모두 평균 내면 비슷한 과일끼리 모이지 않을까?
- ✓ 정말 그런지 데이터를 준비하고 픽셀값을 이용해서 사진을 분류해 보자

# 01. 과일 사진 데이터 준비하기

## ❖ 실습 Dataset (1/4)

### 과일 데이터셋의 출처

- ✓ 이번 수업에서 우리가 사용할 실습 데이터셋의 캐글(Kaggle)에 공개된 Fruits 360이라고 불리는 데이터셋임
- ✓ (URL) <https://www.kaggle.com/moltean/fruits>

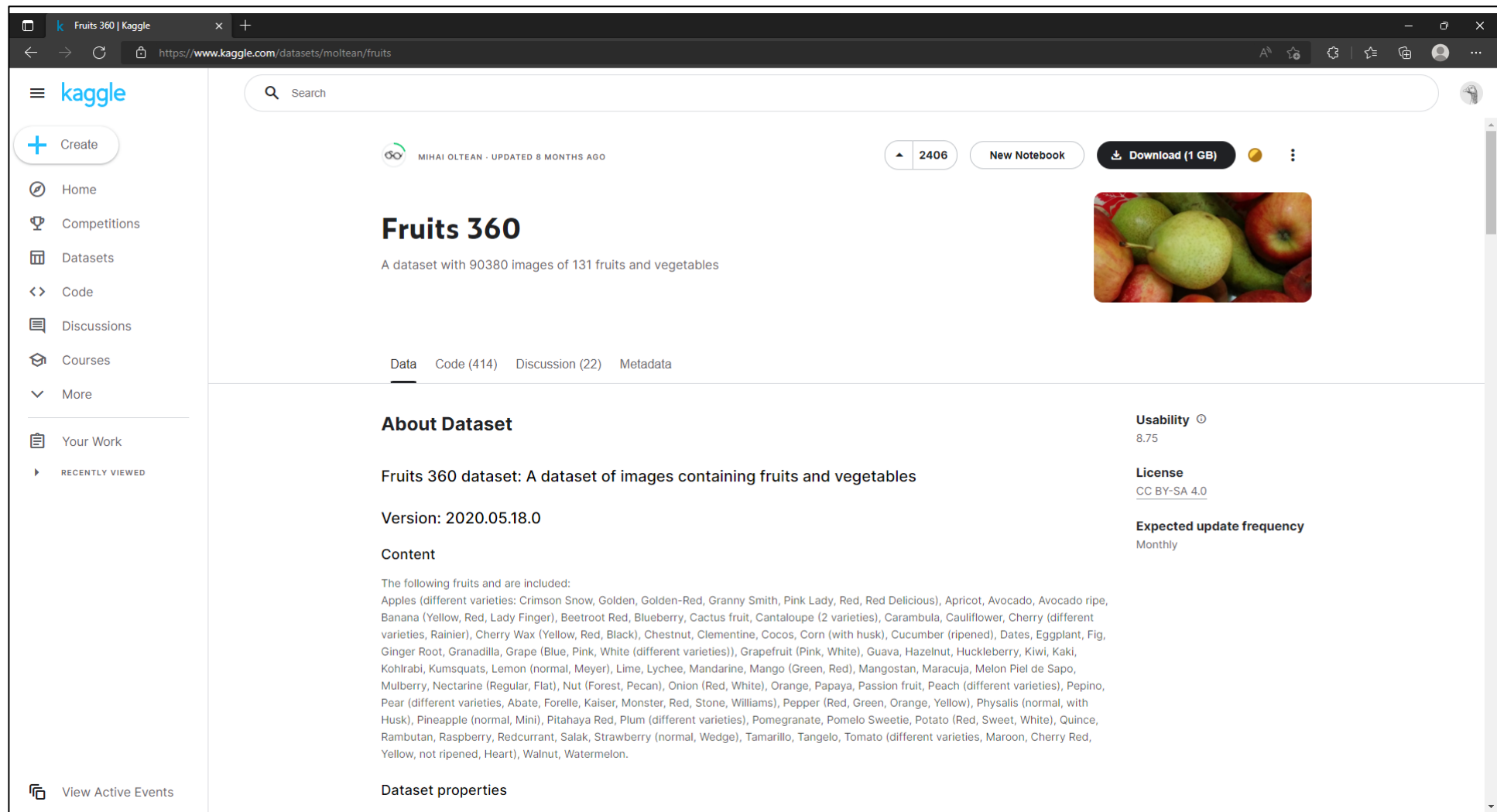
The Kaggle logo, consisting of the word "kaggle" in a lowercase, blue, sans-serif font.

2010년에 설립된 전 세계에서 가장 큰 머신러닝 경연 대회 사이트로  
대회 정보뿐만 아니라 많은 데이터와 참고 자료를 제공함

기업 및 단체에서 데이터와 문제를 등록하면,  
데이터 과학자들이 이를 해결하는 모델을 개발하고 경쟁함

# 01. 과일 사진 데이터 준비하기

## ❖ 실습 Dataset (2/4)



The screenshot displays the Kaggle dataset page for 'Fruits 360'. The page includes a search bar, a sidebar with navigation options like 'Home', 'Competitions', 'Datasets', 'Code', 'Discussions', 'Courses', and 'More', and a main content area. The main content area shows the dataset title 'Fruits 360', a description 'A dataset with 90380 images of 131 fruits and vegetables', and a list of included fruits. The 'About Dataset' section is expanded, showing the version '2020.05.18.0' and the content list. The 'Usability' section shows a score of 8.75, and the 'License' section shows 'CC BY-SA 4.0'.

**Fruits 360**  
A dataset with 90380 images of 131 fruits and vegetables

**About Dataset**

Fruits 360 dataset: A dataset of images containing fruits and vegetables

Version: 2020.05.18.0

**Content**

The following fruits and are included:

Apples (different varieties: Crimson Snow, Golden, Golden-Red, Granny Smith, Pink Lady, Red, Red Delicious), Apricot, Avocado, Avocado ripe, Banana (Yellow, Red, Lady Finger), Beetroot Red, Blueberry, Cactus fruit, Cantaloupe (2 varieties), Carambola, Cauliflower, Cherry (different varieties, Rainier), Cherry Wax (Yellow, Red, Black), Chestnut, Clementine, Cocos, Corn (with husk), Cucumber (ripened), Dates, Eggplant, Fig, Ginger Root, Granadilla, Grape (Blue, Pink, White (different varieties)), Grapefruit (Pink, White), Guava, Hazelnut, Huckleberry, Kiwi, Kaki, Kohlrabi, Kumsquats, Lemon (normal, Meyer), Lime, Lychee, Mandarine, Mango (Green, Red), Mangostan, Maracuja, Melon Piel de Sapo, Mulberry, Nectarine (Regular, Flat), Nut (Forest, Pecan), Onion (Red, White), Orange, Papaya, Passion fruit, Peach (different varieties), Pepino, Pear (different varieties, Abate, Forelle, Kaiser, Monster, Red, Stone, Williams), Pepper (Red, Green, Orange, Yellow), Physalis (normal, with Husk), Pineapple (normal, Mini), Pitahaya Red, Plum (different varieties), Pomegranate, Pomelo Sweetie, Potato (Red, Sweet, White), Quince, Rambutan, Raspberry, Redcurrant, Salak, Strawberry (normal, Wedge), Tamarillo, Tangelo, Tomato (different varieties, Maroon, Cherry Red, Yellow, not ripened, Heart), Walnut, Watermelon.

**Dataset properties**

**Usability**  
8.75

**License**  
CC BY-SA 4.0

**Expected update frequency**  
Monthly

# 01. 과일 사진 데이터 준비하기

## ❖ 실습 Dataset (3/4)

**Fruits 360**

Data Code (414) Discussion (22) Metadata

2406 New Notebook Download (1 GB)

Food Image Data Multiclass Classification

**fruits-360\_dataset** (1 directories)

fruits-360  
4 directories, 2 files

**Data Explorer**  
Version 9 (1.38 GB)

- fruits-360-original-size
- fruits-360\_dataset
  - fruits-360
    - Test
    - Training
    - papers
    - test-multiple\_fruits
    - LICENSE
    - readme.md

**Summary**

- 103k files
  - 103k
  - 1
  - 2
  - 26

**캐글이 아닌, 스마트클래스에서 다운로드하자!**

Activity Overview

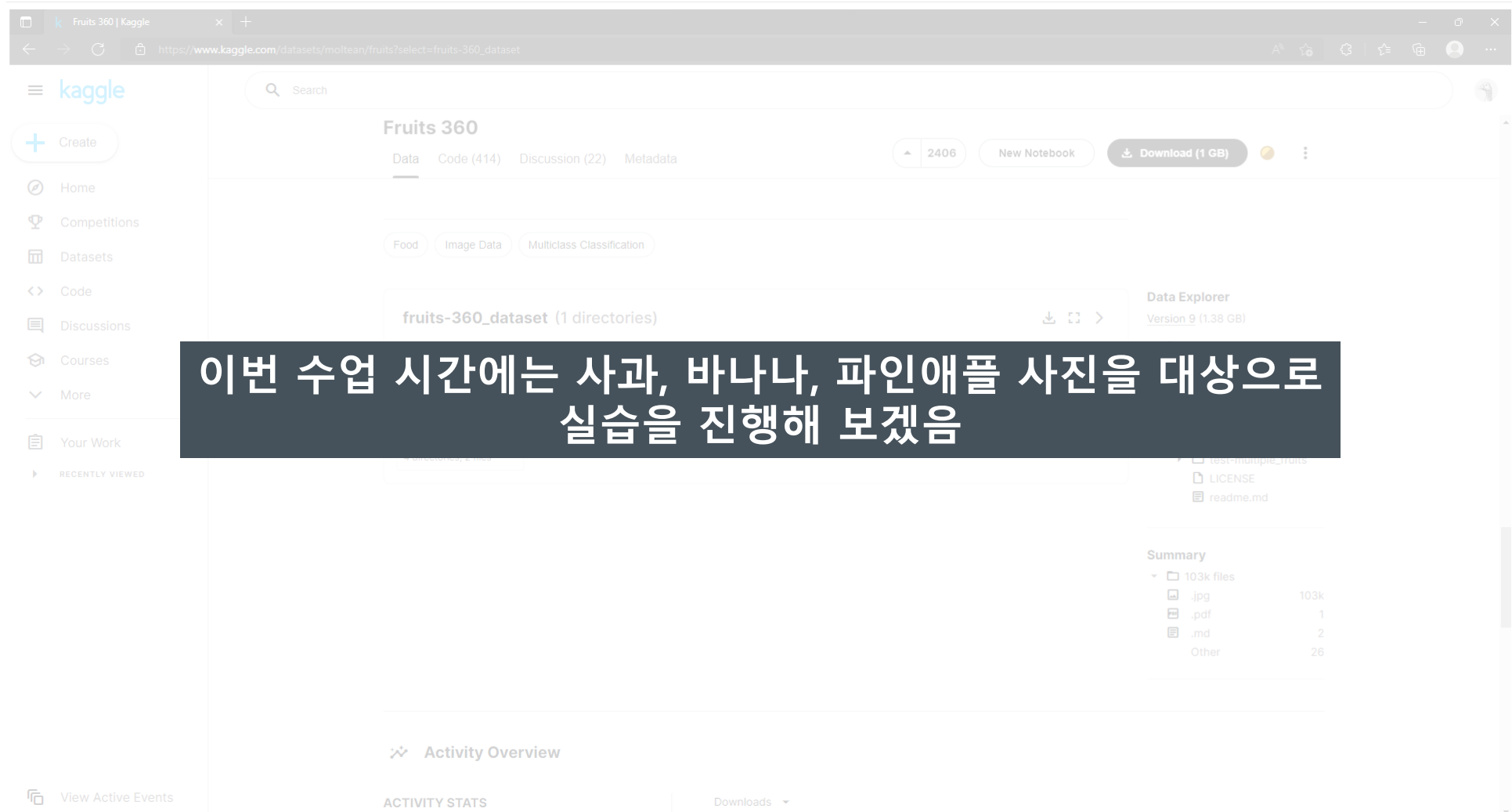
View Active Events

ACTIVITY STATS Downloads



# 01. 과일 사진 데이터 준비하기

## ❖ 실습 Dataset (4/4)



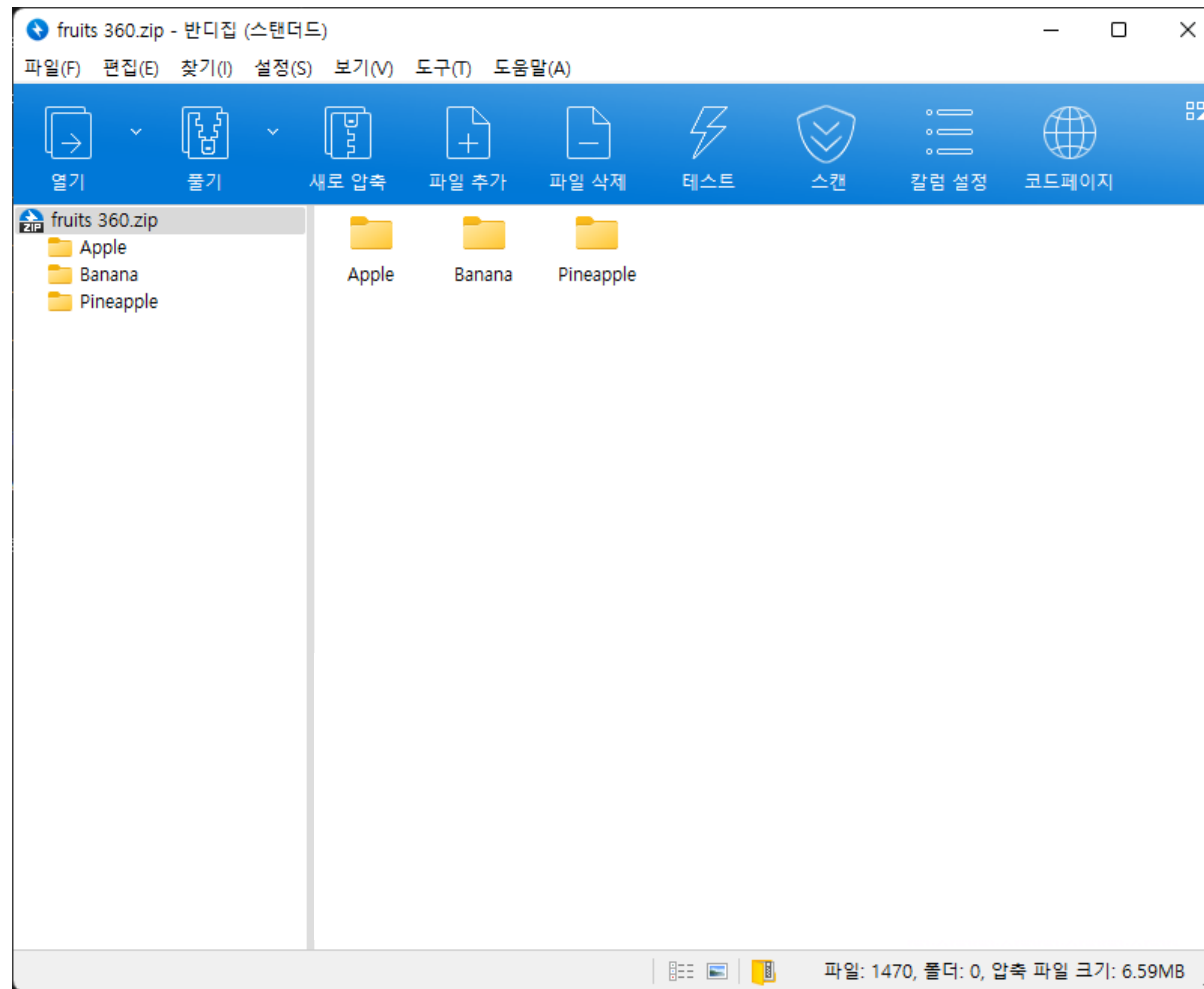
The screenshot shows the Kaggle interface for the 'Fruits 360' dataset. The page includes a sidebar with navigation options like 'Create', 'Home', 'Competitions', 'Datasets', 'Code', 'Discussions', 'Courses', 'More', 'Your Work', and 'RECENTLY VIEWED'. The main content area displays the dataset name 'Fruits 360' with tabs for 'Data', 'Code (414)', 'Discussion (22)', and 'Metadata'. Below these are filters for 'Food', 'Image Data', and 'Multiclass Classification'. A central box shows 'fruits-360\_dataset (1 directories)' with download and view icons. To the right, the 'Data Explorer' section indicates 'Version 9 (1.38 GB)'. A 'Summary' table lists file counts: 103k files, 103k .jpg files, 1 .pdf file, 2 .md files, and 26 Other files. At the bottom, there is an 'Activity Overview' section with 'ACTIVITY STATS' and a 'Downloads' dropdown.

**이번 수업 시간에는 사과, 바나나, 파인애플 사진을 대상으로 실습을 진행해 보겠습니다**

# 01. 과일 사진 데이터 준비하기

## ❖ ① Fruits 360 데이터셋 살펴보기 (1/6)

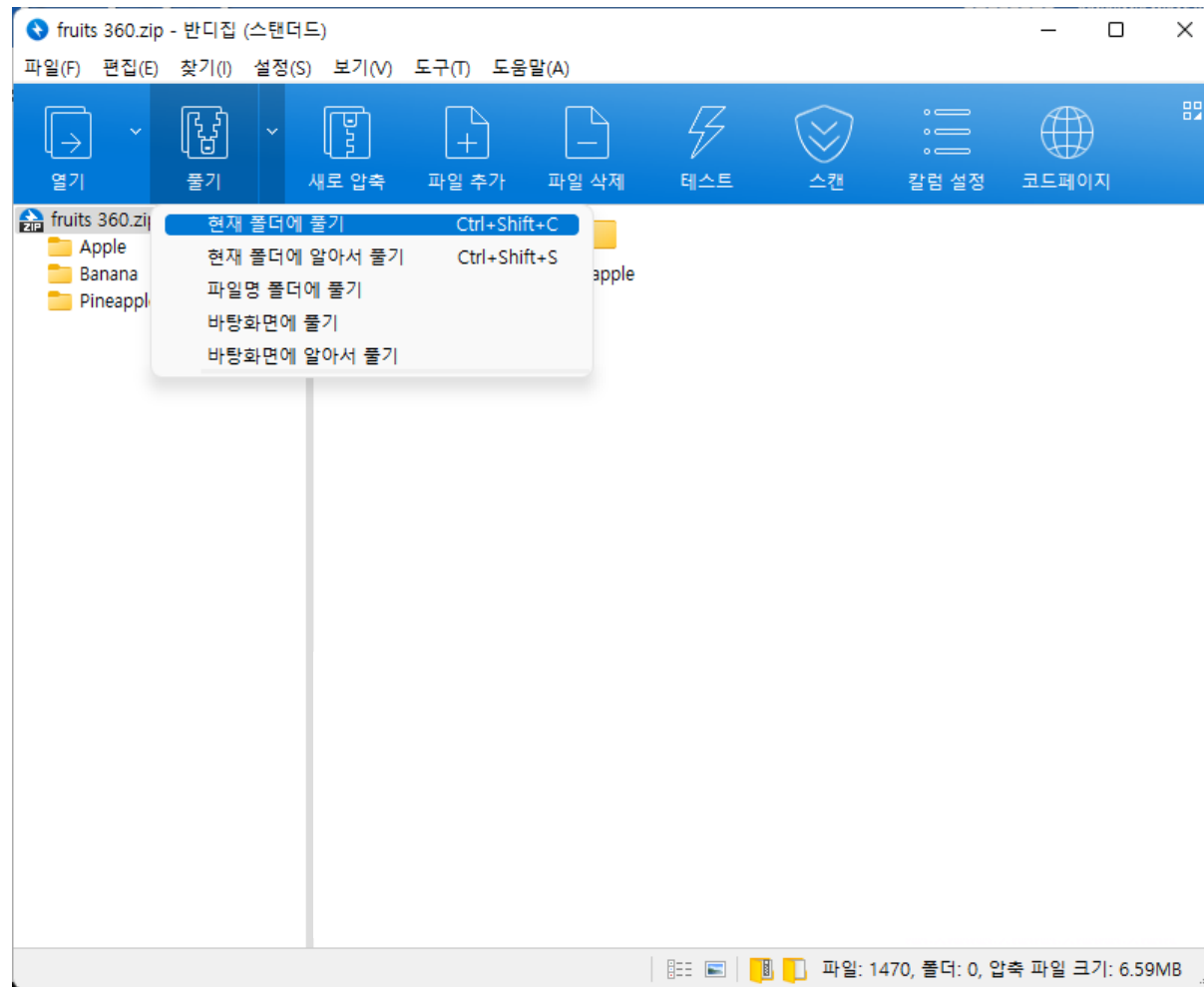
- fruits 360.zip 파일에는 다음과 같이 3개의 폴더가 존재함



# 01. 과일 사진 데이터 준비하기

## ❖ ① Fruits 360 데이터셋 살펴보기 (2/6)

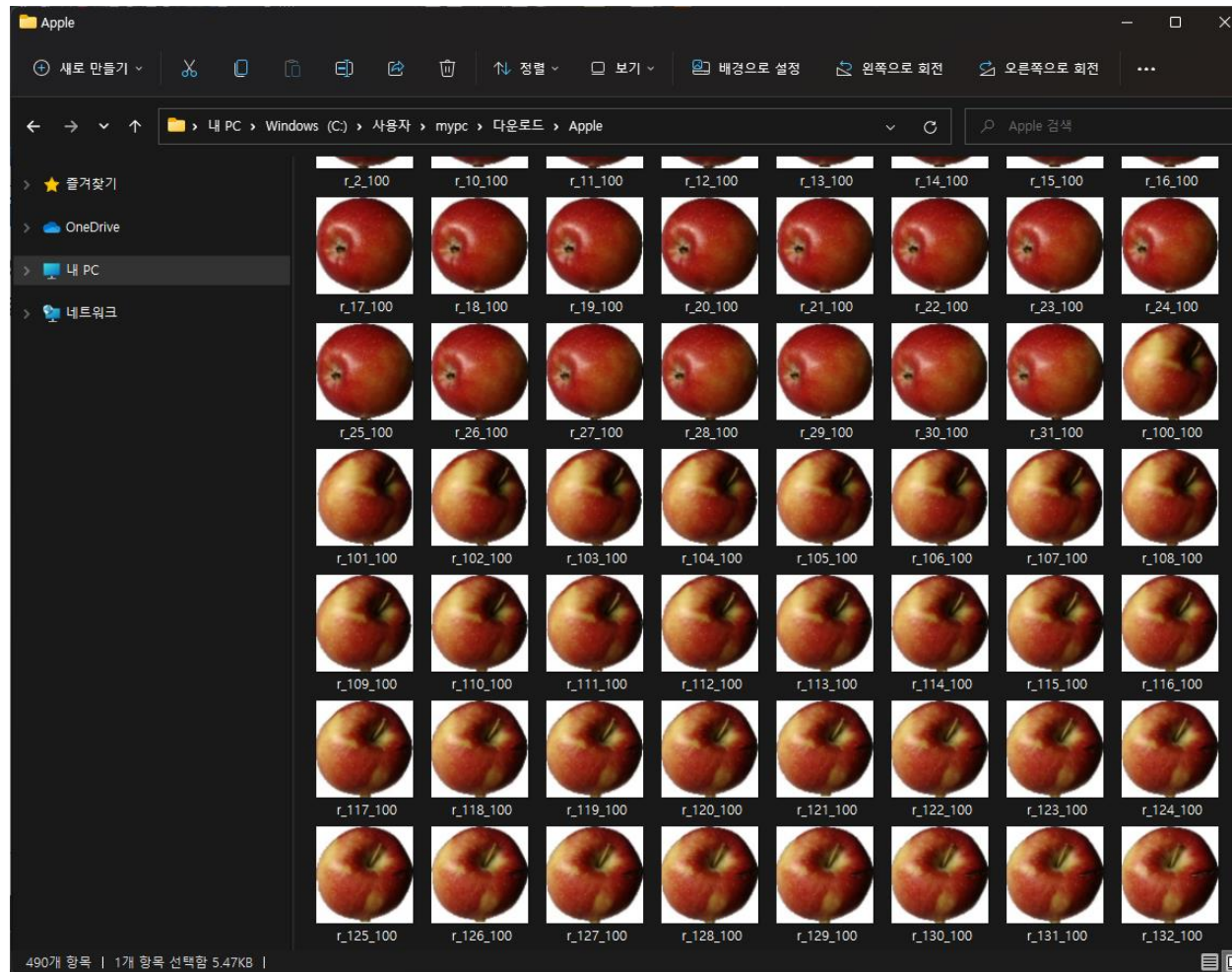
- 다운로드(Downloads) 폴더에 압축 풀기를 수행하자



# 01. 과일 사진 데이터 준비하기

## ❖ ① Fruits 360 데이터셋 살펴보기 (3/6)

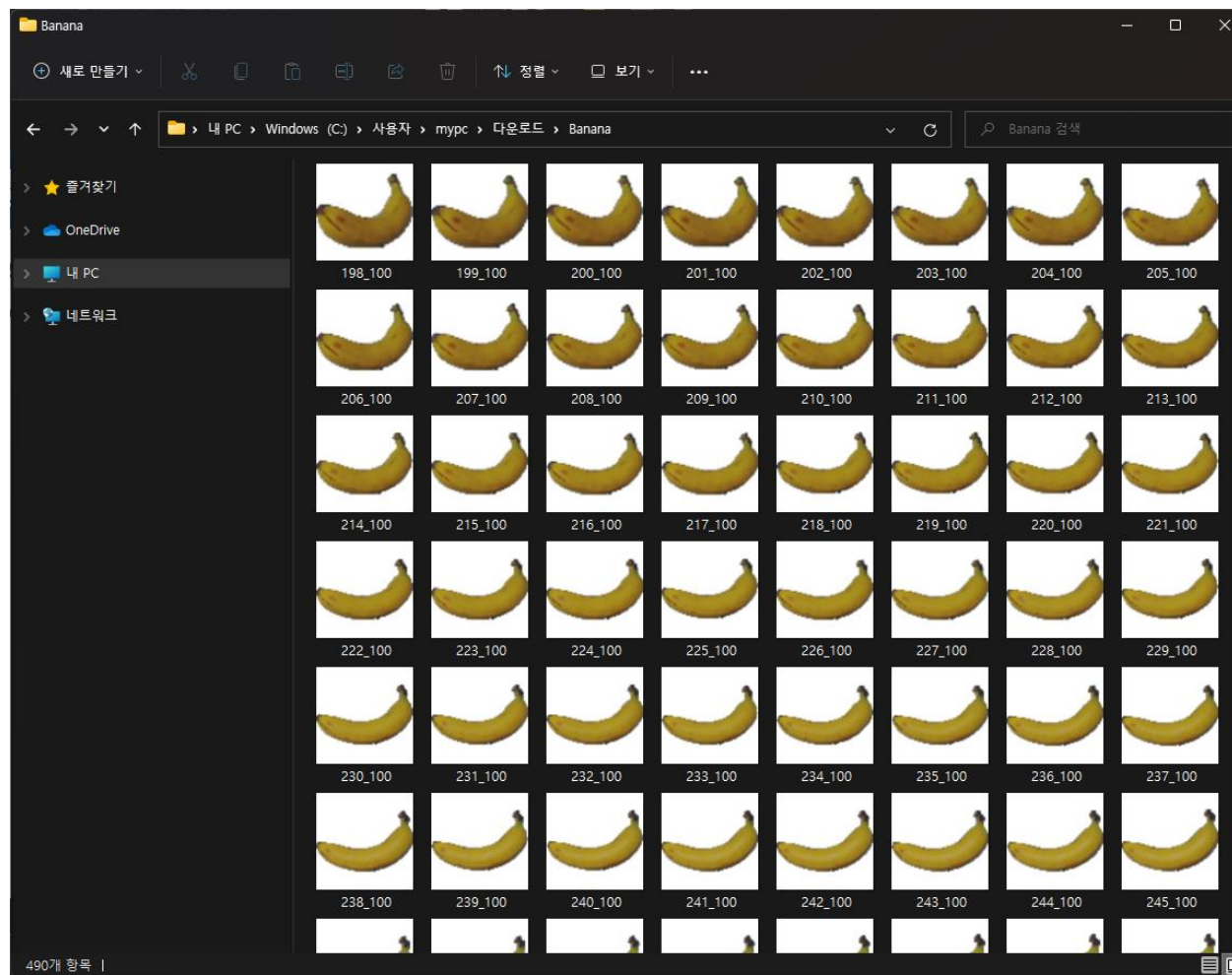
- "Apple" 폴더에 사과 사진이 490장 존재함



# 01. 과일 사진 데이터 준비하기

## ❖ ① Fruits 360 데이터셋 살펴보기 (4/6)

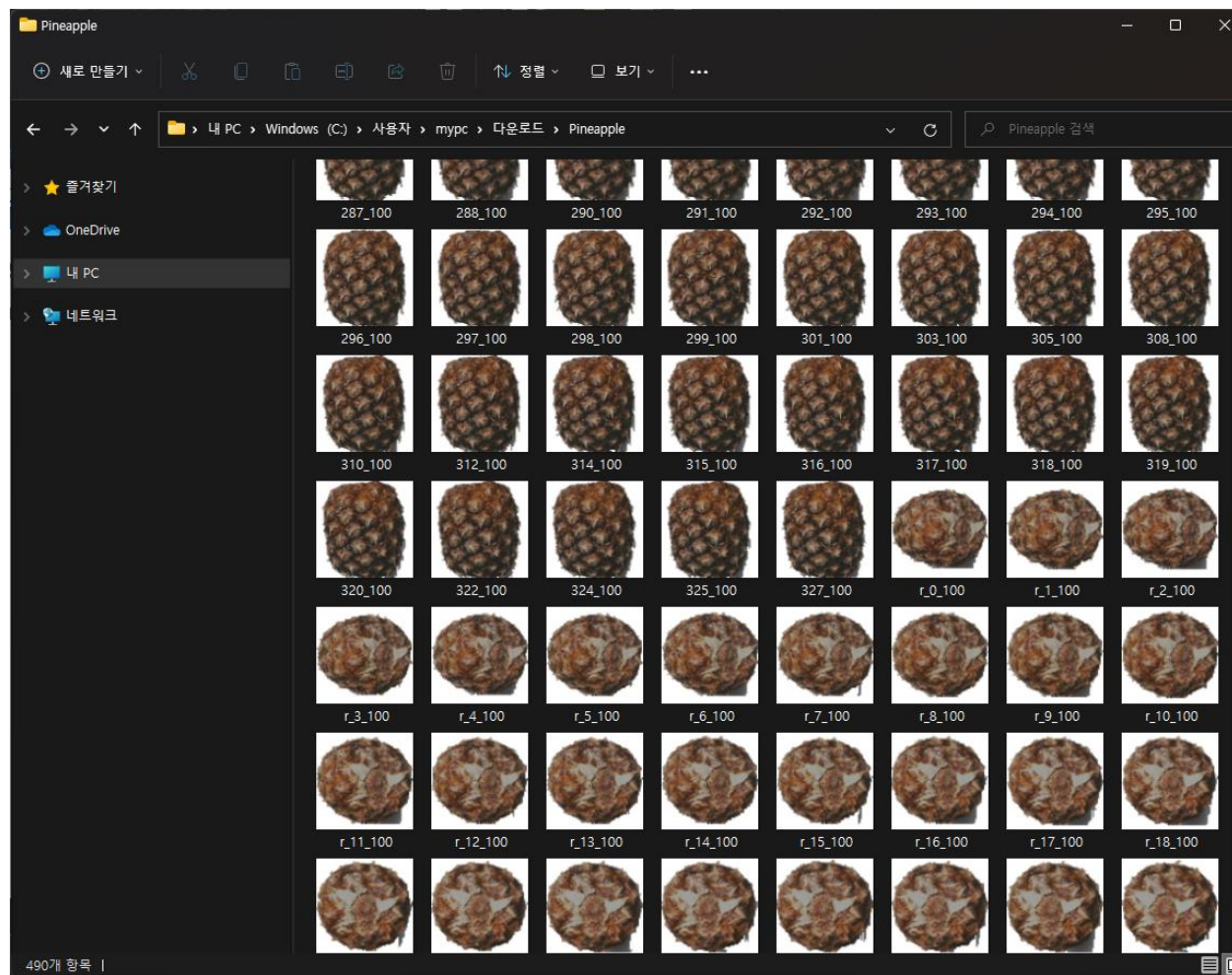
- "Banana" 폴더에 바나나 사진이 490장 존재함



# 01. 과일 사진 데이터 준비하기

## ❖ ① Fruits 360 데이터셋 살펴보기 (5/6)

- "Pineapple" 폴더에 파인애플 사진이 490장 존재함

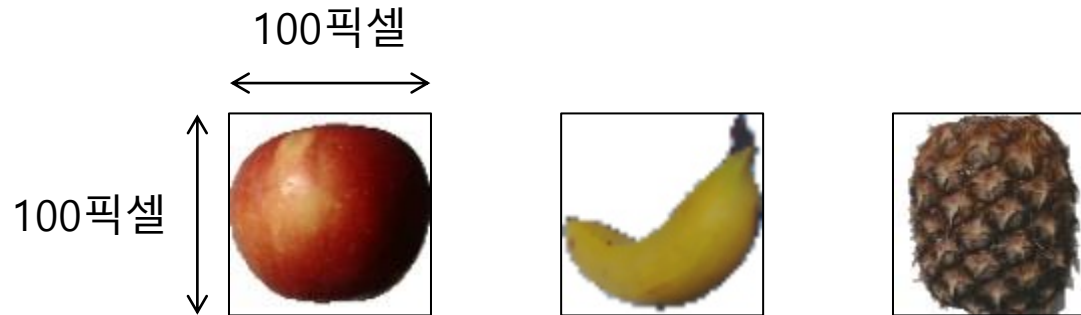




# 01. 과일 사진 데이터 준비하기

## ❖ ① Fruits 360 데이터셋 살펴보기 (6/6)

- 이번 수업을 위해 준비한 과일 데이터는 사과, 바나나, 파인애플을 담고 있는 RGB 이미지임
- 각 이미지의 크기는 100×100 즉, 너비 100픽셀, 높이 100픽셀임
- 각 이미지는 JPG 파일임



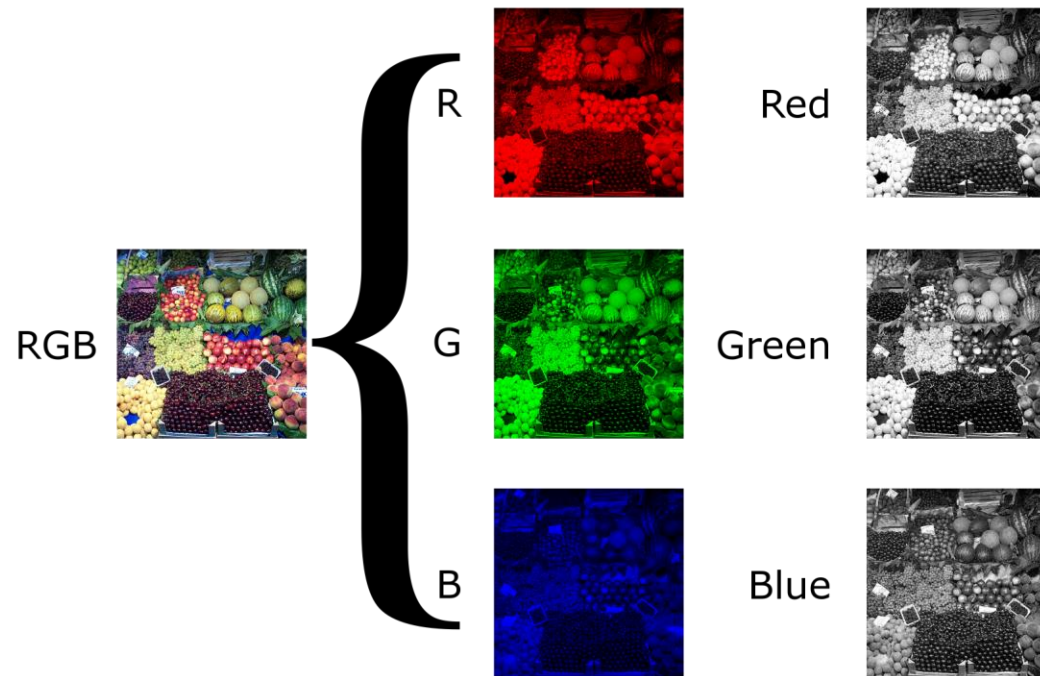
우선 RGB 이미지에서 Grayscale 이미지로 변환하고  
비지도 학습 실습을 진행하도록 하겠음

# 01. 과일 사진 데이터 준비하기

## ❖ ② RGB 이미지와 Grayscale 이미지의 이해 (1/2)

- RGB 이미지는 3개의 채널(Red, Green, Blue)로 구성되어 있음
- RGB 값들은 다음 공식에 의해 Grayscale 값으로 변환될 수 있음

$$Y = 0.299 \times \text{Red} + 0.587 \times \text{Green} + 0.114 \times \text{Blue}$$





# 01. 과일 사진 데이터 준비하기

## ❖ ② RGB 이미지와 Grayscale 이미지의 이해 (2/2)

### Grayscale 이미지

- ✓ Grayscale 이미지는 밝기 정보를 256단계로 구분하여 표현함
- ✓ 즉, Grayscale 이미지에서 하나의 픽셀은 0부터 255 사이의 정숫값을 가질 수 있음
- ✓ 0은 가장 어두운 검은색을 표현하고, 255는 가장 밝은 흰색을 표현함

# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (1/8)

- numpy 라이브러리를 импорт하자
- matplotlib 라이브러리에 존재하는 pyplot 모듈과 image 모듈을 импорт하자
- image 모듈에서 제공하는 imread( ) 함수를 이용하여 RGB 이미지를 읽자

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.image as mpimg
4
5 img = mpimg.imread("Pineapple\\0_100.jpg")
6
7 print(type(img))
8 print(img.shape)
```

### 실행결과

```
<class 'numpy.ndarray'>
(100, 100, 3)
```

- ✓ img는 넘파이 배열임
- ✓ 이 배열의 첫 번째 차원(100)은 이미지의 높이, 두 번째 차원(100)은 이미지의 너비임
- ✓ 세 번째 차원(3)은 RGB 채널에 대응됨

# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (2/8)

- 각 RGB 채널에 저장된  $100 \times 100$  데이터를 분리해 내자
- RGB-to-Grayscale 공식을 이용하여 Grayscale 값을 계산하자

```
1 R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
2 imgGray = 0.299 * R + 0.587 * G + 0.114 * B
3 imgGray = np.array(imgGray, dtype="int")
4
5 print(imgGray.shape)
6 print(imgGray[50, :])
```

$$Y = 0.299 \times \text{Red} + 0.587 \times \text{Green} + 0.114 \times \text{Blue}$$

### 실행결과

```
(100, 100)
[255 252 254 254 255 255 253 253 140 113  96  64 59 51 48 52 47 51
 58  52  49  51  61  62  45  43  72 113 145 115 39 47 52 54 54 67
 62  52  45  36  42  56  55  92  68  45  45  40 78 78 64 78 88 86
 94 114 151 133  80  54  56  67  82  70  48  38 36 37 76 45 44 44
 54  60  54  54  55  49  48  46  41  36  41  39 37 47 56 55 56 59
114 254 253 254 255 253 255 255 255 255]
```

- ✓ imgGray 배열에 저장된 값을 숫자로 보니 잘 와닿지 않음
- ✓ 이미지로 보면 좋을 것 같음

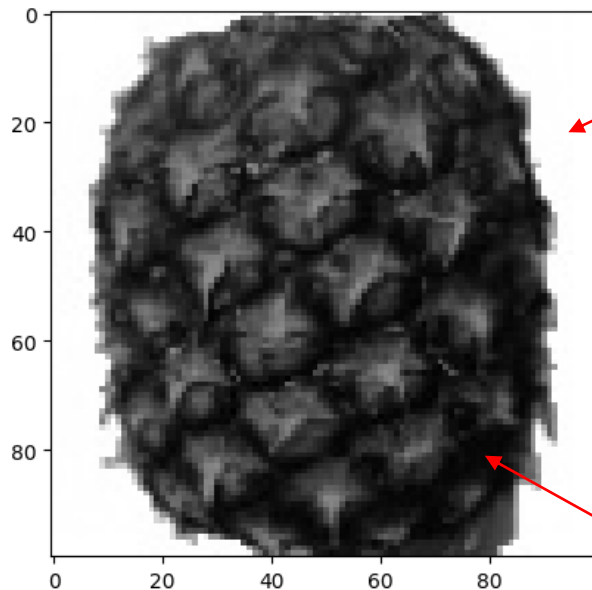
# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (3/8)

- matplotlib의 imshow( ) 함수를 사용하면 넘파이 배열로 저장된 이미지를 쉽게 그릴 수 있음
- Grayscale 이미지이므로, cmap 매개변수를 "gray"로 지정하자

```
1 plt.figure()  
2 plt.imshow(imgGray, cmap="gray")  
3 plt.show()
```

### 실행결과



255에 가까운 값

- ✓ 파인애플 사진이 맞음
- ✓ 0에 가까울 수록 검게 나타나고, 255에 가까울수록 밝게 표시됨

0에 가까운 값

# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (4/8)

- 우리의 관심 대상은 바탕이 아니라 파인애플임
- 흰색 바탕은 우리에게 중요하지 않지만, 컴퓨터는 255에 가까운 바탕에 집중할 것임
- 따라서 바탕을 0에 가깝게 만들고, 사진에 짙게 나온 파인애플을 255에 가깝게 만들어야 함

### 컴퓨터는 왜 255에 가까운 바탕에 집중하나요?

- ✓ 알고리즘은 어떤 출력을 만들기 위해 곱셈, 덧셈을 함
- ✓ 픽셀값이 0이면 출력도 0이 되어 의미가 없음
- ✓ 픽셀값이 높으면 출력값도 커지기 때문에 의미를 부여하기 좋음

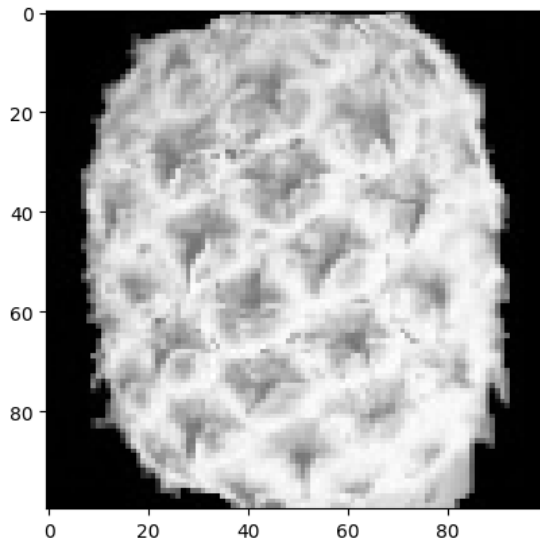
# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (5/8)

- 관심 대상의 영역을 높은 값으로 바꾸었지만 imshow( ) 함수로 출력할 때, 바탕이 검게 나오므로 보기에 썩 좋지 않음

```
1 imgGray2 = 255 - imgGray
2 plt.figure()
3 plt.imshow(imgGray2, cmap="gray")
4 plt.show()
```

### 실행결과



우리가 보는 것과 컴퓨터가 처리하는 방식이 다르기 때문에,  
종종 Grayscale 이미지를 이렇게 반전하여 사용함

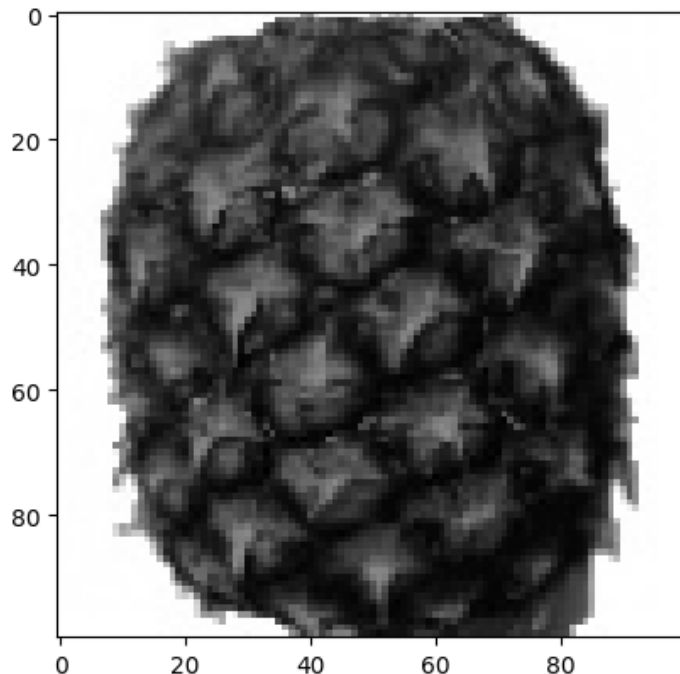
# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (6/8)

- cmap 매개변수를 "gray\_r"로 지정하면, 다시 반전하여 우리 눈에 보기 좋게 출력함

```
1 plt.figure()  
2 plt.imshow(imgGray2, cmap="gray_r")  
3 plt.show()
```

### 실행결과



이 그림에서 밝은 부분은 0에 가깝고,  
짙은 부분은 255에 가까운 값이라는 것을 꼭 기억하자!

# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (7/8)

- Apple 폴더와 Banana 폴더에 있는 RGB 이미지들도 Grayscale 이미지로 변환해 보자

```
1  img = mpimg.imread("Apple\\0_100.jpg")
2
3  R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
4  imgGray = 0.299 * R + 0.587 * G + 0.114 * B
5  imgGray = np.array(imgGray, dtype="int")
6
7  img_apple = 255 - imgGray
8
9  img = mpimg.imread("Banana\\0_100.jpg")
10
11 R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
12 imgGray = 0.299 * R + 0.587 * G + 0.114 * B
13 imgGray = np.array(imgGray, dtype="int")
14
15 img_banana = 255 - imgGray
```



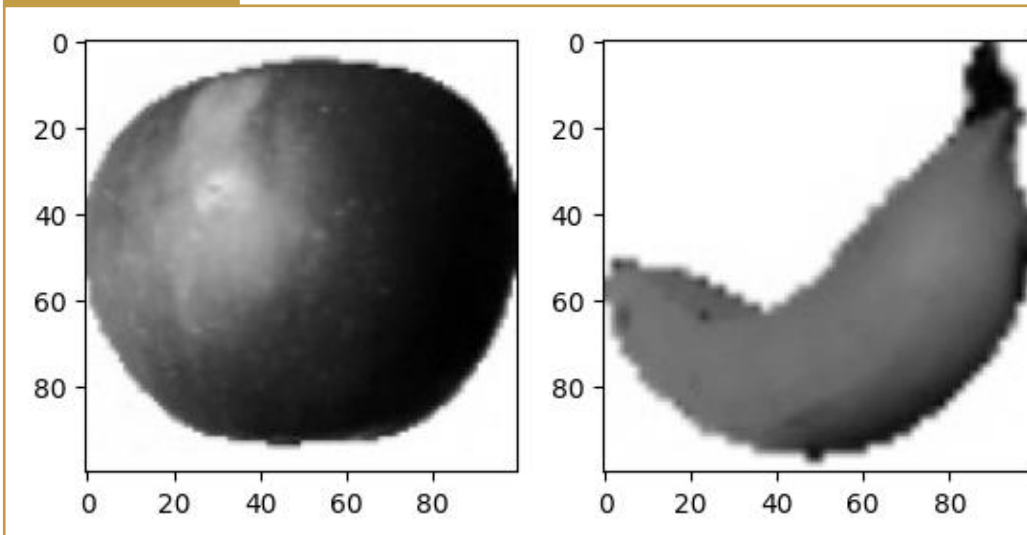
# 01. 과일 사진 데이터 준비하기

## ❖ ③ RGB 이미지를 Grayscale 이미지로 변환하기 (8/8)

- matplotlib 라이브러리에서 제공하는 `subplots()` 함수를 사용하면 여러 개의 그래프를 배열처럼 쌓을 수 있도록 도와줌
- `subplots()` 함수의 두 매개변수는 그래프를 쌓을 행과 열을 지정함

```
1 fig, axs = plt.subplots(1, 2) # 하나의 행과 두 개의 열을 지정함
2 axs[0].imshow(img_apple, cmap="gray_r")
3 axs[1].imshow(img_banana, cmap="gray_r")
4 plt.show()
```

### 실행결과



- ✓ 반환된 `axs`는 2개의 서브 그래프를 담고 있는 배열임
- ✓ `axs[0]`에는 사과 이미지를 그리고, `axs[1]`에는 바나나 이미지를 그렸음

# 01. 과일 사진 데이터 준비하기

## ❖ ④ Grayscale 이미지를 넘파이 배열로 저장하기 (1/4)

- 디렉토리 경로 정보를 쉽게 얻기 위해서 os 라이브러리를 импорт함

```
1 import os
2 cur_dir = os.getcwd()
3 fruit_list = ["Apple", "Banana", "Pineapple"]
4
5 fruit_npy = []
6 for fruit_name in fruit_list:
7     folder_name = cur_dir + "\\ " + fruit_name
8     file_list = os.listdir(folder_name)
9
10    for file_name in file_list:
11        img = mpimg.imread(folder_name + "\\ " + file_name)
12        R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
13        imgGray = 0.299 * R + 0.587 * G + 0.114 * B
14        imgGray = np.array(imgGray, dtype="int")
15        imgGray2 = 255 - imgGray
16        fruit_npy.append(imgGray2)
17
18 fruit_npy = np.array(fruit_npy)
19 print(fruit_npy.shape)
```

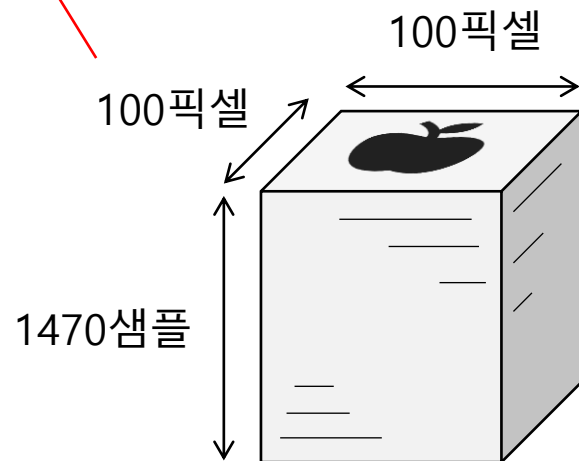
- ✓ getcwd( ) 함수:  
현재 작업 디렉토리(Current Working Directory) 문자열을 반환함
- ✓ listdir( ) 함수:  
지정한 디렉토리 내의 모든 파일과 디렉토리의 리스트를 반환함

# 01. 과일 사진 데이터 준비하기

## ❖ ④ Grayscale 이미지를 넘파이 배열로 저장하기 (2/4)

실행결과

(1470, 100, 100)



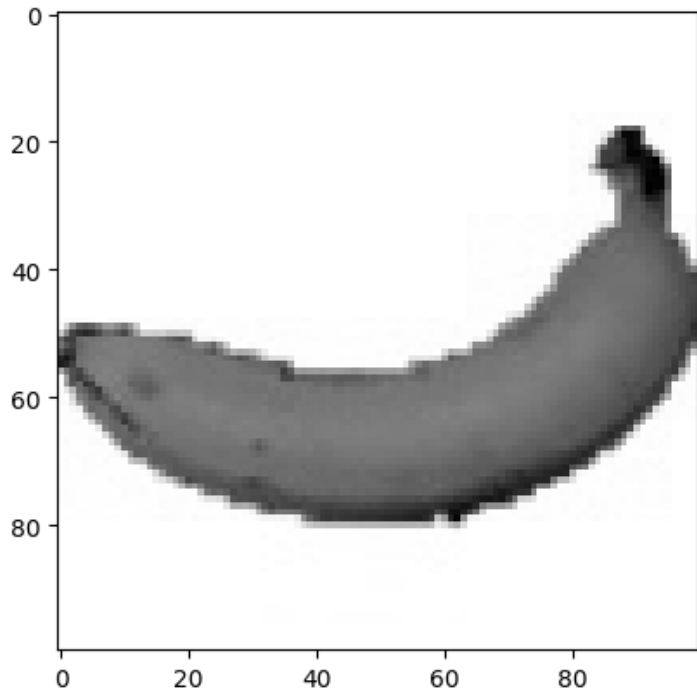
# 01. 과일 사진 데이터 준비하기

## ❖ ④ Grayscale 이미지를 넘파이 배열로 저장하기 (3/4)

- fruit\_npy 배열에 저장된 Grayscale 이미지를 출력해 보자

```
1 plt.figure()  
2 plt.imshow(fruit_npy[600], cmap="gray_r")  
3 plt.show()
```

### 실행결과



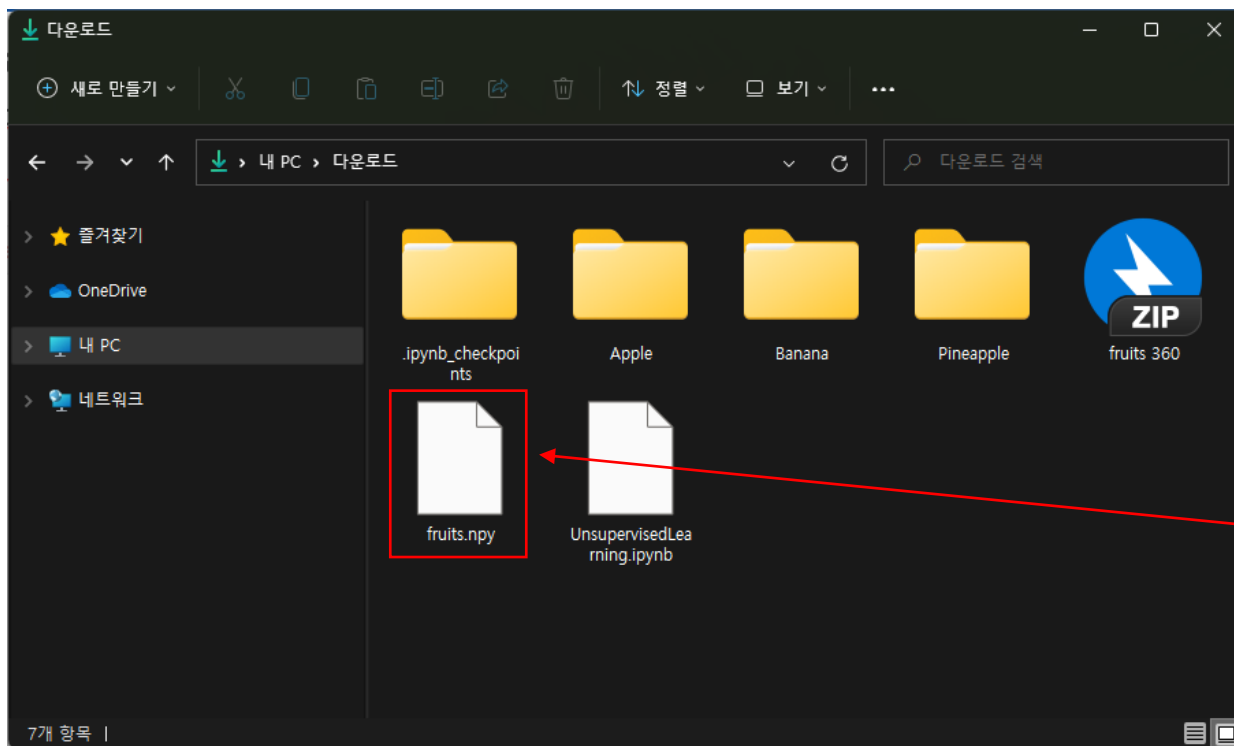
여기서 `fruit_npy[600]`과  
`fruit_npy[600, :, :]`는 동일한 표현임!

# 01. 과일 사진 데이터 준비하기

## ❖ ④ Grayscale 이미지를 넘파이 배열로 저장하기 (4/4)

- fruit\_numpy 배열을 저장해 보자
- 넘파이 배열의 기본 저장 포맷인 npy 파일로 저장하면 됨

```
1 np.save("fruits.npy", fruit_numpy)
```



현재 작업 중인 폴더에 저장된 것을  
확인할 수 있음

# 01. 과일 사진 데이터 준비하기

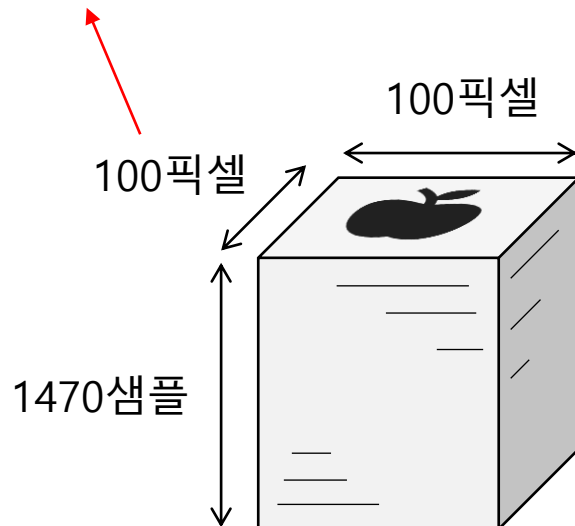
## ❖ ⑤ npy 파일 로드하기

- npy 파일을 로드(Load)하는 방법을 아주 간단함
- load( ) 메서드에 파일 이름을 전달하면 됨

```
1 fruits = np.load("fruits.npy")  
2  
3 print(fruits.shape)
```

### 실행결과

(1470, 100, 100)



- ✓ fruits는 3차원 넘파이 배열임
- ✓ 이 배열의 첫 번째 차원(1470)은 샘플의 개수를 나타냄
- ✓ 두 번째 차원(100)은 이미지의 높이를 나타냄
- ✓ 세 번째 차원(100)은 이미지의 너비를 나타냄

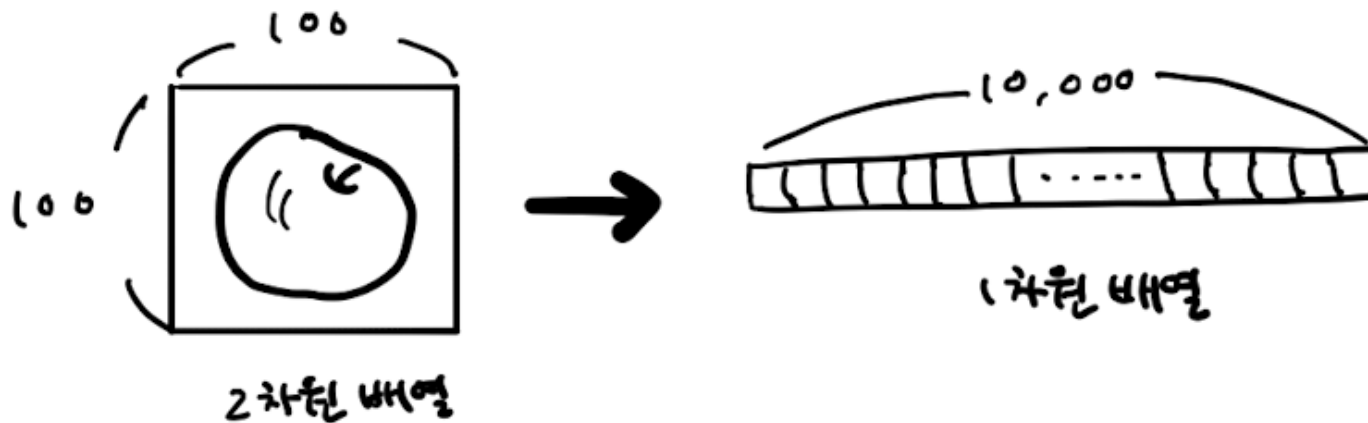
## 02. 픽셀값 분석하기

- 01. 과일 사진 데이터 준비하기
- 03. 평균값과 가까운 사진 고르기
- 04. 비슷한 샘플끼리 모으기
- 05. 마무리

## 02. 픽셀값 분석하기

### ❖ ① 2차원 배열을 1차원 배열로 만들기 (1/2)

- 사용하기 쉽게 fruits 데이터를 사과, 바나나, 파인애플로 각각 나누어 보자
- 넘파이 배열을 나눌 때  $100 \times 100$  이미지를 펼쳐서 길이가 10,000인 1차원 배열로 만들어 보자
- 이렇게 펼치면 이미지로 출력하긴 어렵지만 배열을 계산할 때 편리함





## 02. 픽셀값 분석하기

### ❖ ① 2차원 배열을 1차원 배열로 만들기 (2/2)

- fruits 배열에서 순서대로 490개씩 선택하기 위해 슬라이싱(Slicing) 연산자를 사용함
- 그 다음 reshape( ) 메서드를 사용해 두 번째 차원(100)과 세 번째 차원(100)을 10,000으로 합침
- 첫 번째 차원을 -1로 지정하면 자동으로 남은 차원을 할당함
- 여기에서는 첫 번째 차원이 샘플 개수임

```
1 apple = fruits[0:490].reshape(-1, 100 * 100)
2 banana = fruits[490:980].reshape(-1, 100 * 100)
3 pineapple = fruits[980:].reshape(-1, 100 * 100)
4
5 print("사과 배열의 크기:", apple.shape)
6 print("바나나 배열의 크기:", banana.shape)
7 print("파인애플 배열의 크기:", pineapple.shape)
```

#### 실행결과

사과 배열의 크기: (490, 10000)  
바나나 배열의 크기: (490, 10000)  
파인애플 배열의 크기: (490, 10000)

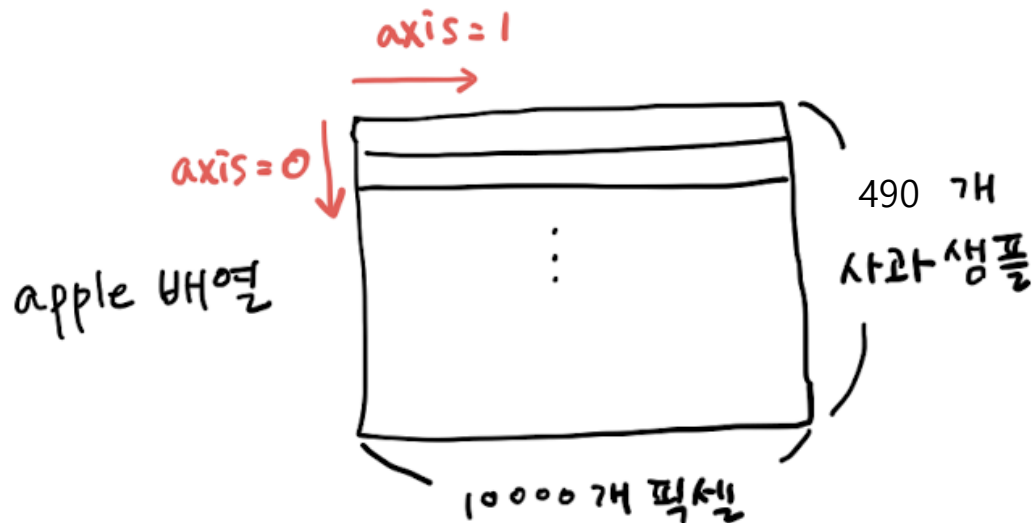
우리는 fruits 배열에 각 과일이 490개씩 있다고 알고 있지만, 실전에서는 어떤 과일이 몇 개씩 존재할지 알 수 없음

여기에서는 예를 위해서 만든 데이터임을 잊지 말자!

## 02. 픽셀값 분석하기

### ❖ ② 픽셀 평균값 계산하기 (1/2)

- 이제 apple, banana, pineapple 배열에 들어 있는 샘플의 픽셀 평균값을 계산해 보자
- 이를 위해서, 넘파이 `mean()` 메서드를 사용하겠음
- 샘플마다 픽셀의 평균값을 계산해야 하므로 `mean()` 메서드가 평균을 계산할 축을 지정해야 함
  - ✓ `axis=0`으로 하면 첫 번째 축인 행(Row)을 따라 계산함
  - ✓ `axis=1`로 하면 두 번째 축인 열(Column)을 따라 계산함



## 02. 픽셀값 분석하기

### ❖ ② 픽셀 평균값 계산하기 (2/2)

- 우리가 필요한 것은 샘플의 평균임
- 샘플은 모두 가로로 값을 나열했으니, axis=1로 지정하여 평균을 계산하겠음
- 평균을 계산하는 넘파이 np.mean( ) 함수를 사용해도 되지만  
넘파이 배열은 이런 함수들을 메서드로도 제공함
- apple 배열의 mean( ) 메서드로 각 샘플의 픽셀 평균값을 계산해 보자

```
1 print(apple.mean(axis=1).shape)
2 print(apple.mean(axis=1))
```

#### 실행결과

```
(490,)
[133.6748 151.9998 151.8302 151.7464 151.6402 151.4957 151.3911 154.8597
 ... (중략) ...
 153.126 152.8166 152.6294 152.356 152.209 155.3344 155.1957 151.6138
 151.489 143.1498]
```

사과 샘플 490개에 대한 픽셀 평균값을 계산하였음.  
히스토그램을 그려보면 평균값이 어떻게 분포되어 있는지 확인할 수 있음

## 02. 픽셀값 분석하기

### ❖ ③ 히스토그램 그리기 (1/2)

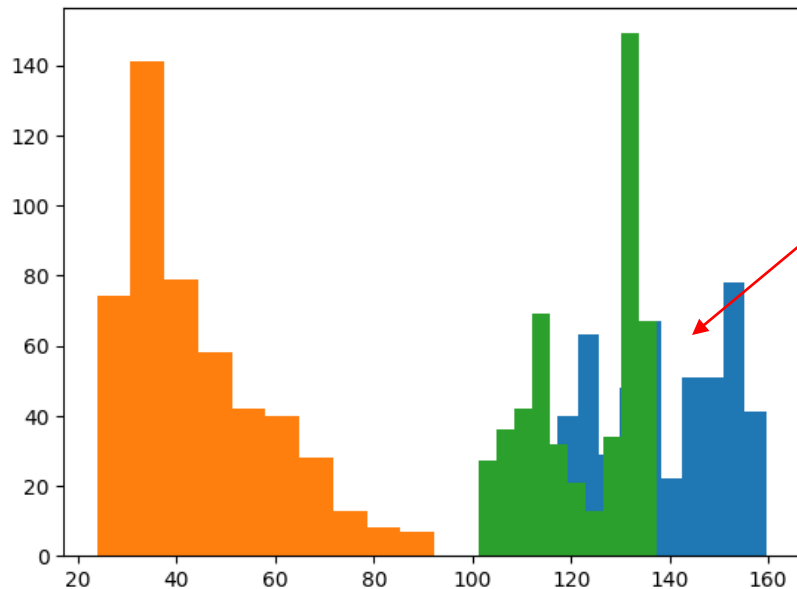
- matplotlib의 hist() 함수를 사용해 히스토그램을 그려보자

```
1 plt.figure()  
2 plt.hist(apple.mean(axis=1))  
3 plt.hist(banana.mean(axis=1))  
4 plt.hist(pineapple.mean(axis=1))  
5 plt.show()
```

#### 히스토그램(Histogram)

- ✓ 값이 발생한 빈도를 그래프로 표시한 것임
- ✓ 보통 x축이 값의 구간(계급)이고 y축은 발생 빈도 (도수)임

#### 실행결과



히스토그램을 겹쳐 그리다 보니 잘 안 보이는 부분이 생김.  
어떻게 하면 좋을까?

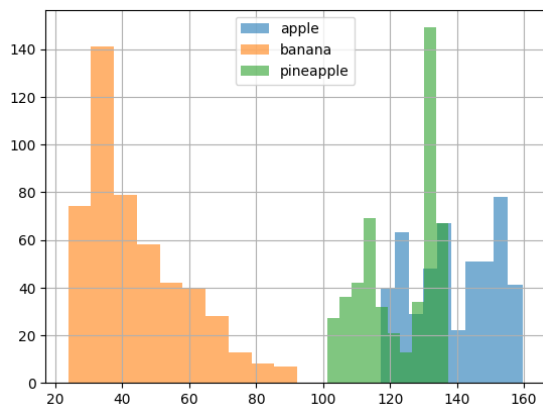
## 02. 픽셀값 분석하기

### ❖ ③ 히스토그램 그리기 (2/2)

- alpha 매개변수를 1보다 작게 하면 투명도를 줄 수 있음
- 또 matplotlib의 legend( ) 함수를 사용해 어떤 과일의 히스토그램인지 범례 정보를 추가하겠음

```
1 plt.figure()
2 plt.hist(apple.mean(axis=1), alpha=0.6, label="apple")
3 plt.hist(banana.mean(axis=1), alpha=0.6, label="banana")
4 plt.hist(pineapple.mean(axis=1), alpha=0.6, label="pineapple")
5 plt.grid(True)
6 plt.legend()
7 plt.show()
```

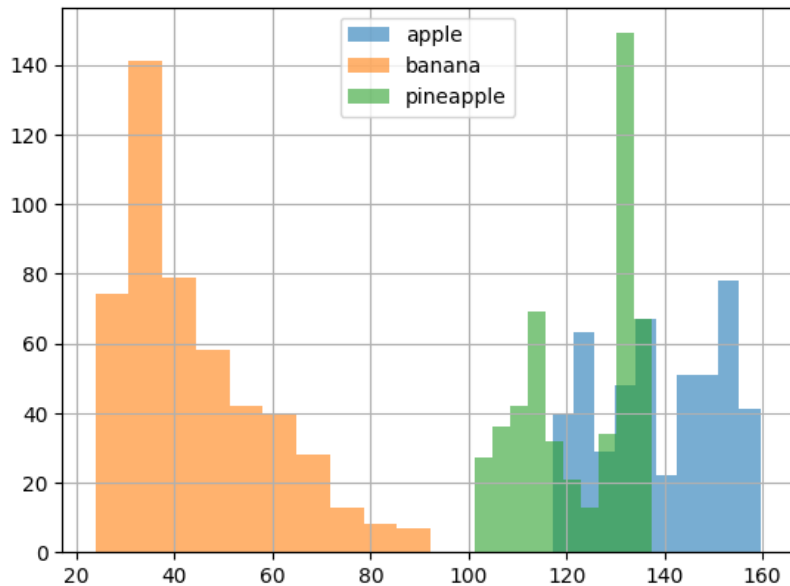
#### 실행결과



## 02. 픽셀값 분석하기

### ❖ ④ 히스토그램 분석하기

- 히스토그램을 보면 바나나 사진의 평균값은 40 아래에 집중되어 있음
- 사과나 파인애플은 100~160 사이에 많이 모여 있음
- 바나나는 평균값만으로 사과나 파인애플과 확실히 구분됨
- 바나나는 사진에서 차지하는 영역이 작기 때문에 평균값이 작음
- 반면 사과나 파인애플은 많이 겹쳐 있어서 픽셀 평균값으로 구분하기 쉽지 않음
- 사과나 파인애플은 대체로 형태가 동그랗고, 사진에서 차지하는 크기도 비슷하기 때문임



좀 더 나은 방법은 없을까?  
샘플의 평균값이 아니라 픽셀별 평균값을 비교해 보면 어떨까?

세 과일은 모양이 다르므로 픽셀값이 큰 위치가  
서로 조금 다를 것 같음

## 02. 픽셀값 분석하기

### ❖ ⑤ 픽셀별 평균값 계산하기 (1/2)

- 픽셀의 평균을 계산하는 것도 간단함
- axis=0으로 지정하면 됨
- matplotlib의 bar( ) 함수를 사용해 픽셀 10,000개에 대한 평균값을 막대그래프로 그려보자
- subplots( ) 함수로 3개의 서브 그래프를 만들어 사과, 바나나, 파인애플에 대해 그려보자

## 02. 픽셀값 분석하기

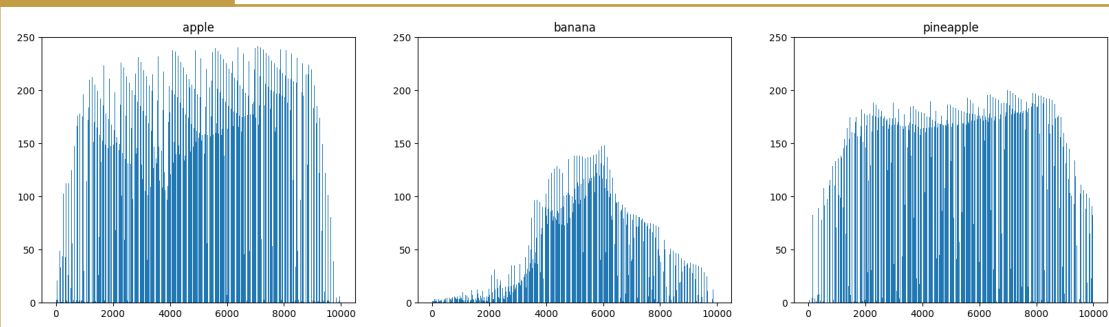
### ❖ ⑤ 픽셀별 평균값 계산하기 (2/2)

- `set_title()` 함수를 이용하면 서브 그래프의 제목(Title)을 입력할 수 있음

```
1 fig, axs = plt.subplots(1, 3, figsize=(20, 5))
2 axs[0].bar(range(10000), np.mean(apple, axis=0))
3 axs[1].bar(range(10000), np.mean(banana, axis=0))
4 axs[2].bar(range(10000), np.mean(pineapple, axis=0))
5 axs[0].set_title("apple")
6 axs[1].set_title("banana")
7 axs[2].set_title("pineapple")
8 axs[0].set_ylim([0, 250])
9 axs[1].set_ylim([0, 250])
10 axs[2].set_ylim([0, 250])
11 plt.show()
```

`set_ylim()` 함수를 이용하면 y축 범위 값을 설정할 수 있음

#### 실행결과



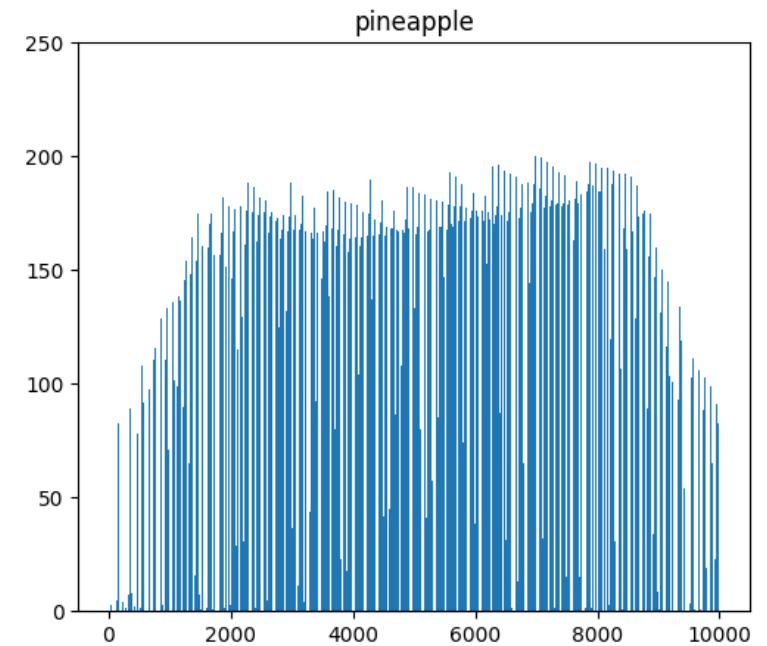
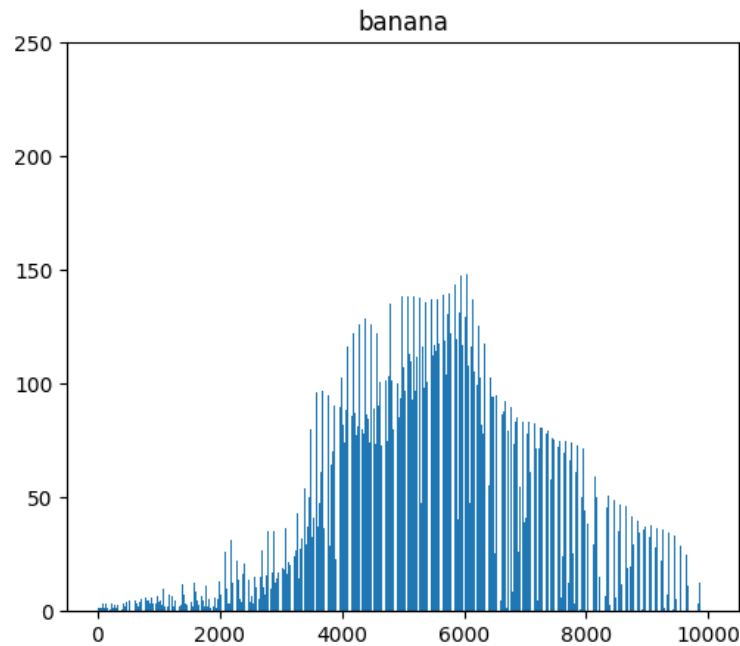
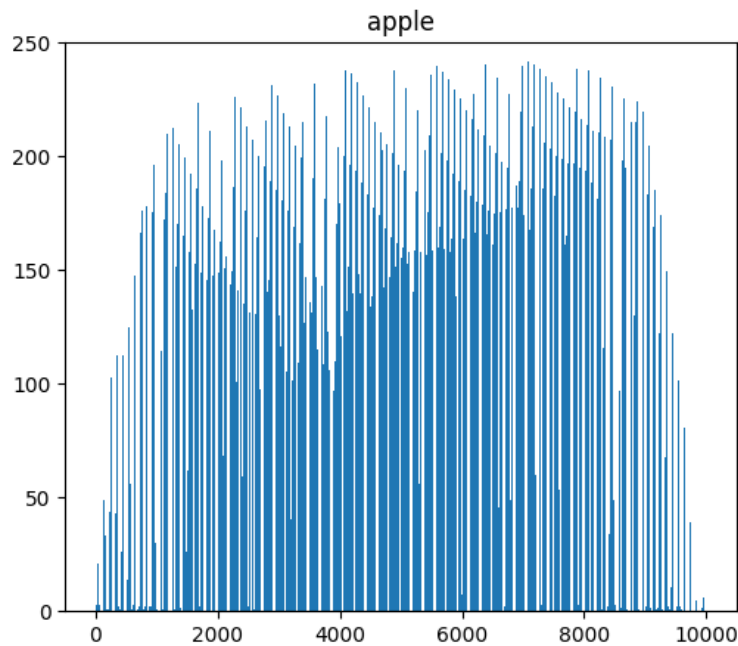


## 02. 픽셀값 분석하기

### ❖ ⑥ 막대그래프 분석하기 (1/3)

- 3개의 그래프를 보면 과일마다 높은 구간이 다름
- 사과는 오른쪽으로 갈수록 값이 높아짐
- 바나나는 중앙의 픽셀값이 높음
- 파인애플은 픽셀값이 비교적 고르면서 높음

픽셀 평균값을 100×100 크기로 바꿔서  
이미지처럼 출력하여 비교해 보면 어떨까?



## 02. 픽셀값 분석하기

### ❖ ⑥ 막대그래프 분석하기 (2/3)

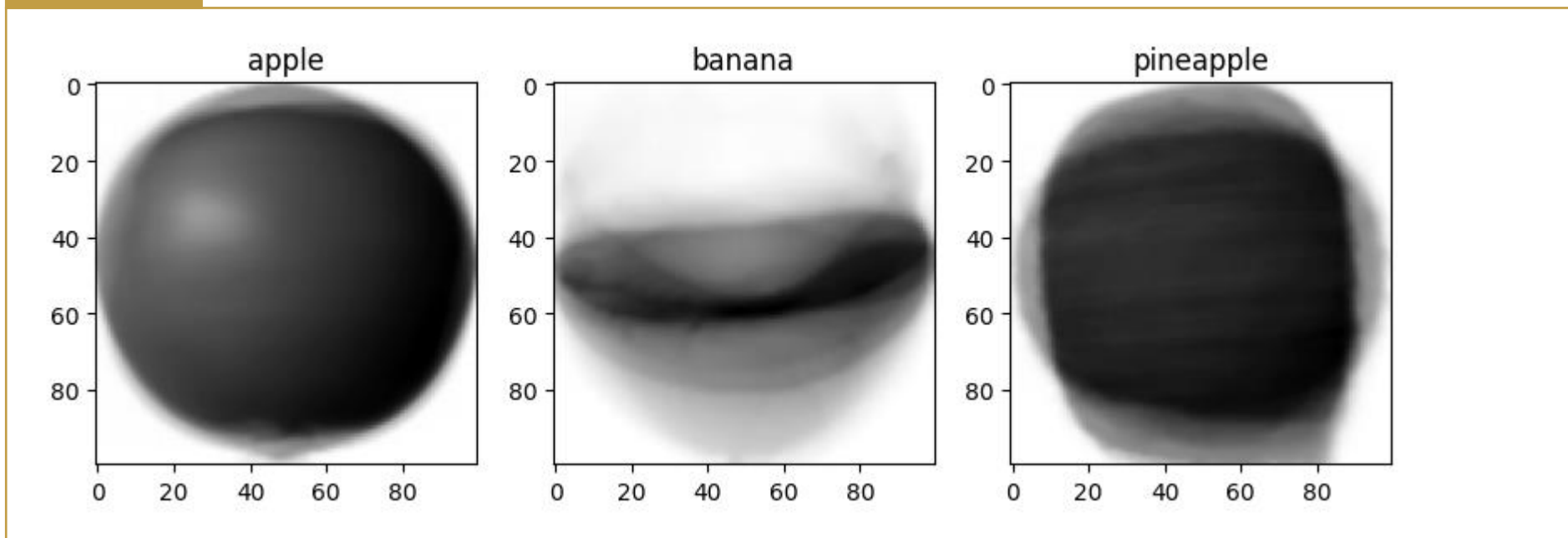
- 픽셀을 평균 낸 이미지를 모든 사진을 합쳐 놓은 대표 이미지로 생각할 수 있음

```
1 apple_mean = np.mean(apple, axis=0).reshape(100, 100)
2 banana_mean = np.mean(banana, axis=0).reshape(100, 100)
3 pineapple_mean = np.mean(pineapple, axis=0).reshape(100, 100)
4
5 fig, axs = plt.subplots(1, 3, figsize=(10, 5))
6 axs[0].imshow(apple_mean, cmap="gray_r")
7 axs[1].imshow(banana_mean, cmap="gray_r")
8 axs[2].imshow(pineapple_mean, cmap="gray_r")
9 axs[0].set_title("apple")
10 axs[1].set_title("banana")
11 axs[2].set_title("pineapple")
12 plt.show()
```

## 02. 픽셀값 분석하기

### ❖ ⑥ 막대그래프 분석하기 (3/3)

#### 실행결과



이 대표 이미지와 가까운 사진을 골라 낸다면  
사과, 바나나, 파인애플을 구분할 수 있지 않을까?

## 03. 평균값과 가까운 사진 고르기

- 01. 과일 사진 데이터 준비하기
- 02. 픽셀값 분석하기
- 04. 비슷한 샘플끼리 모으기
- 05. 마무리

## 03. 평균값과 가까운 사진 고르기

### ❖ ① 평균 절대 오차(Mean Absolute Error, MAE) 계산하기

- 사과 사진의 평균값인 `apple_mean`과 가장 가까운 사진을 골라 보자
- 절대 오차(Absolute Error)를 사용하겠음
- `fruits` 배열에 있는 모든 샘플에서 `apple_mean`을 뺀 절댓값의 평균을 계산하면 됨

```
1 abs_diff = np.abs(fruits - apple_mean)
2 abs_mean = np.mean(abs_diff, axis=(1, 2))
3 print(abs_mean.shape)
```

#### 넘파이 `abs()` 함수

- ✓ 절댓값을 계산하는 함수임
- ✓ 예를 들어, `np.abs(-1)`은 1을 반환함
- ✓ 배열을 입력하면 모든 원소의 절댓값을 계산하여 입력과 동일한 크기의 배열을 반환함
- ✓ 이 함수는 `np.absolute()` 함수의 다른 이름임

- ✓ `abs_diff`는 (1470, 100, 100) 크기의 배열임
- ✓ 따라서 각 샘플에 대한 평균을 구하기 위해 `axis`에 두 번째, 세 번째 차원을 모두 지정했음
- ✓ 이렇게 계산한 `abs_mean`은 각 샘플의 오차 평균이므로 크기가 (1470, )인 1차원 배열임

### 03. 평균값과 가까운 사진 고르기

#### ❖ ② 평균 절대 오차가 가장 작은 순서대로 100개의 이미지 출력하기 (1/2)

- abs\_mean 값이 가장 작은 순서대로 100개의 이미지를 골라 보자
- 즉, apple\_mean과 오차가 가장 작은 샘플 100개를 고르는 것임

```
1 apple_index = np.argsort(abs_mean)[:100]
2
3 fig, axs = plt.subplots(10, 10, figsize=(5, 5))
4 for j in range(10):
5     for k in range(10):
6         axs[j, k].imshow(fruits[apple_index[j * 10 + k]], cmap="gray_r")
7         axs[j, k].axis("off")
8
9 plt.show()
```

넘파이 `argsort()` 함수

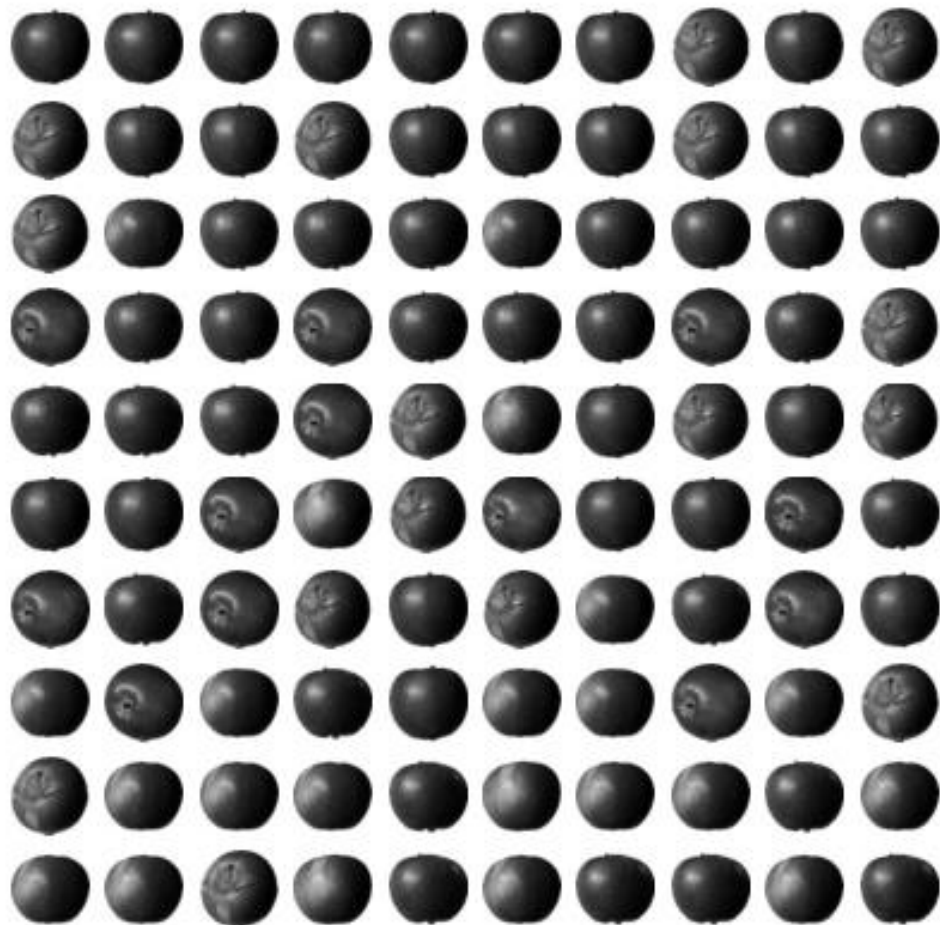
✓ 작은 것에서 큰 순서대로 나열한(오름차순)  
abs\_mean 배열의 인덱스(Index)를 반환함

- ✓ `argsort()` 함수로부터 반환된 인덱스 중에서 처음 100개를 선택해 10×10 격자로 이루어진 그래프를 그림
- ✓ 먼저 `subplot()` 함수로 10×10, 총 100개의 서브 그래프를 만들
- ✓ 그다음 이중 for 반복문을 순회하면서 10개의 행과 열에 이미지를 출력함
- ✓ `axs`는 (10, 10) 크기의 2차원 배열이므로 `j, k` 두 첨자를 사용하여 서브 그래프 위치를 지정함
- ✓ 깔끔하게 이미지만 그리기 위해 `axis("off")`를 사용하여 좌표축을 그리지 않았음
- ✓ 궁금하다면 "on"으로 값을 바꾸거나 해당 줄을 삭제하고 다시 그림을 그려보자

### 03. 평균값과 가까운 사진 고르기

❖ ② 평균 절대 오차가 가장 작은 순서대로 100개의 이미지 출력하기 (2/2)

#### 실행결과



apple\_mean과 가장 가까운 사진 100개를  
골랐더니 모두 사과임!

## 03. 평균값과 가까운 사진 고르기

### ❖ ③ 바나나 이미지에 대해서 수행하기 (1/2)

- banana\_mean과 가장 가까운 사진 100개를 골라서 출력해 보자

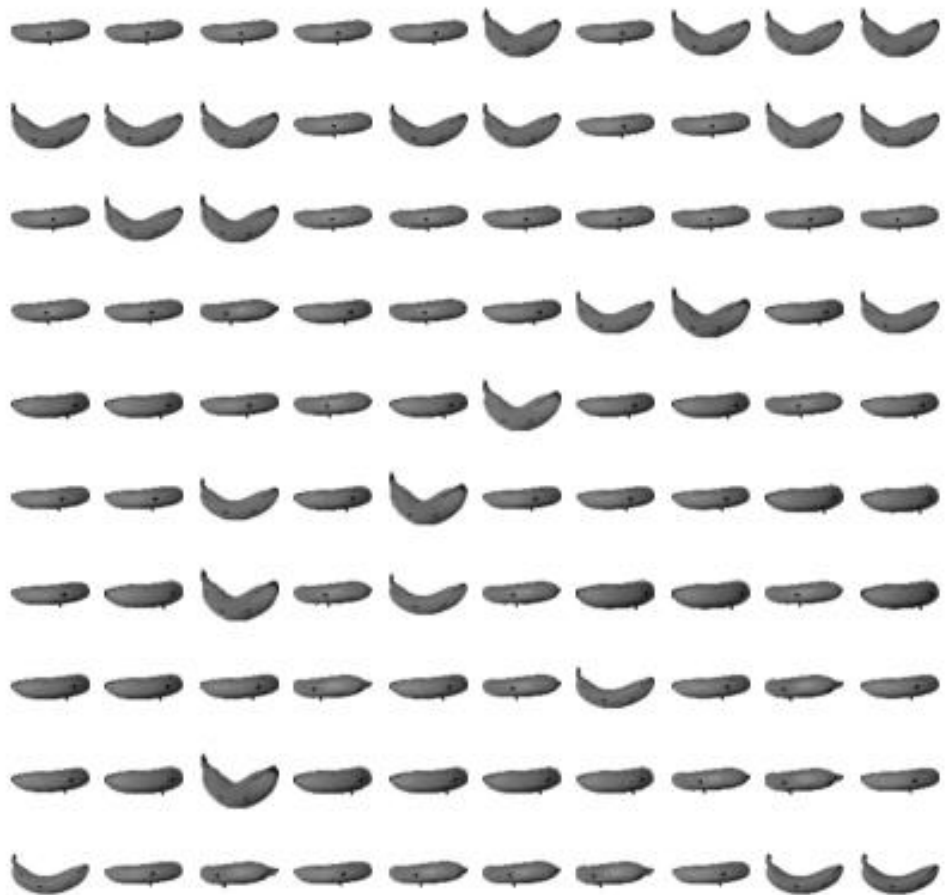
```
1 abs_diff = np.abs(fruits - banana_mean)
2 abs_mean = np.mean(abs_diff, axis=(1, 2))
3 banana_index = np.argsort(abs_mean)[:100]
4
5 fig, axs = plt.subplots(10, 10, figsize=(5, 5))
6 for j in range(10):
7     for k in range(10):
8         axs[j, k].imshow(fruits[banana_index[j * 10 + k]], cmap="gray_r")
9         axs[j, k].axis("off")
10
11 plt.show()
```



## 03. 평균값과 가까운 사진 고르기

### ❖ ③ 바나나 이미지에 대해서 수행하기 (2/2)

#### 실행결과



banana\_mean과 가장 가까운 사진 100개를  
골랐더니 모두 바나나임!

## 03. 평균값과 가까운 사진 고르기

### ❖ ④ 파인애플 이미지에 대해서 수행하기 (1/2)

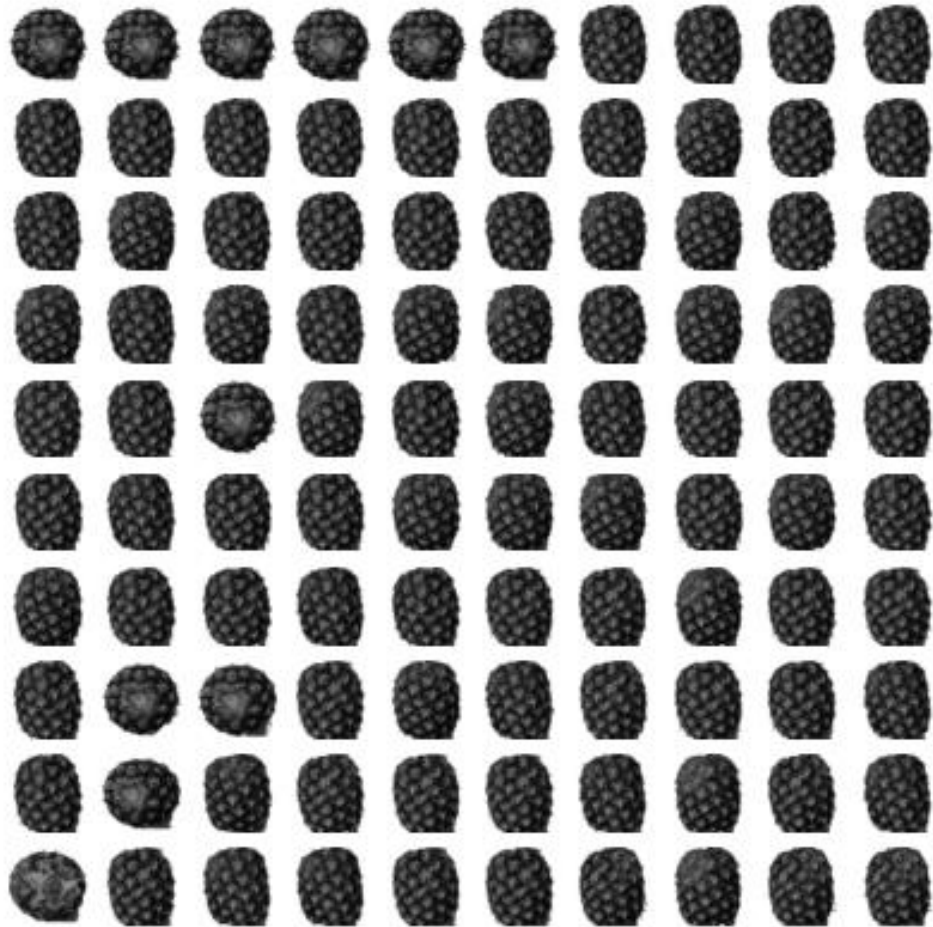
- pineapple\_mean과 가장 가까운 사진 100개를 골라서 출력해 보자

```
1 abs_diff = np.abs(fruits - pineapple_mean)
2 abs_mean = np.mean(abs_diff, axis=(1, 2))
3 pineapple_index = np.argsort(abs_mean)[:100]
4
5 fig, axs = plt.subplots(10, 10, figsize=(5, 5))
6 for j in range(10):
7     for k in range(10):
8         axs[j, k].imshow(fruits[pineapple_index[j * 10 + k]], cmap="gray_r")
9         axs[j, k].axis("off")
10
11 plt.show()
```

### 03. 평균값과 가까운 사진 고르기

#### ❖ ④ 파인애플 이미지에 대해서 수행하기 (2/2)

##### 실행결과



pineapple\_mean과 가장 가까운 사진 100개를 골랐더니 모두 파인애플임!

## 03. 평균값과 가까운 사진 고르기

### ❖ 군집(Clustering) (1/2)

- 이번 수업 시간에는 Grayscale 이미지에 있는 픽셀값을 사용해서 과일 사진을 모으는 작업을 실습해 보았음
  - ✓ 이렇게 비슷한 샘플끼리 그룹으로 모으는 작업을 군집(Clustering)이라고 함
  - ✓ 군집은 대표적인 비지도 학습 작업 중 하나임
  - ✓ 군집 알고리즘으로 만든 그룹을 클러스터(Cluster)라고 부름

## 03. 평균값과 가까운 사진 고르기

### ❖ 군집(Clustering) (2/2)

- 하지만 우리는 이미 사과, 바나나, 파인애플이 있다는 것을 알고 있었음
- 즉, 타겟(레이블)값을 알고 있었기 때문에 사과, 바나나, 파인애플의 사진 평균값을 계산해서 가장 가까운 과일을 찾을 수 있었음
- 실제 비지도 학습에서는 타겟값을 모르기 때문에, 이처럼 샘플의 평균값을 미리 구할 수 없음

타겟값을 모르면서 어떻게 세 과일의 평균값을 찾을 수 있을까?

## 04. 비슷한 샘플끼리 모으기

- 01. 과일 사진 데이터 준비하기
- 02. 픽셀값 분석하기
- 03. 평균값과 가까운 사진 고르기
- 05. 마무리

## 04. 비슷한 샘플끼리 모으기

### ❖ 전체 과정 요약 (1/2)

- 동덕 마켓의 새로운 이벤트를 위해 고객들이 보낸 과일 사진을 자동으로 모아야 함
- 고객으로부터 어떤 과일 사진을 받게 될지 미리 예상할 수 없기 때문에  
타겟(레이블)값을 준비하여 분류 모델을 학습시키기는 어려움
- 타겟값이 없을 때 데이터에 있는 패턴을 찾거나 데이터 구조를 파악하는 머신러닝 방식을  
비지도 학습(Unsupervised Learning)이라고 함
- 타겟이 없기 때문에 알고리즘을 직접적으로 가르칠 수 없음
- 대신 알고리즘은 스스로 데이터가 어떻게 구성되어 있는지 분석함

## 04. 비슷한 샘플끼리 모으기

### ❖ 전체 과정 요약 (2/2)

- 대표적인 비지도 학습 문제는 '군집(Clustering)'임
- 군집은 비슷한 샘플끼리 그룹으로 모으는 작업임
- 이번 수업 시간에는 사진의 픽셀을 사용해서, 군집과 비슷한 작업을 수행해 보았음
- 하지만 샘플이 어떤 과일인지 미리 알고 있었기 때문에, 각 과일 사진의 평균값을 알 수 있었음

실제 비지도 학습에서는 타겟이 없는 사진을 사용해야 함.  
다음 수업 시간에는 이런 경우 어떻게 샘플 그룹의 평균값을 찾는지 알아보겠음



## 04. 비슷한 샘플끼리 모으기

### ❖ 전체 소스 코드 (1/5)

```
1  # 01. 과일 사진 데이터 준비하기
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.image as mpimg
5
6  img = mpimg.imread("Pineapple\\0_100.jpg")
7
8  print(type(img))
9  print(img.shape)
10
11  R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
12  imgGray = 0.299 * R + 0.587 * G + 0.114 * B
13  imgGray = np.array(imgGray, dtype="int")
14
15  print(imgGray.shape)
16  print(imgGray[50,:])
17
18  plt.figure()
19  plt.imshow(imgGray, cmap="gray")
20  plt.show()
```

```
21  imgGray2 = 255 - imgGray
22
23  plt.figure()
24  plt.imshow(imgGray2, cmap="gray")
25  plt.show()
26
27  plt.figure()
28  plt.imshow(imgGray2, cmap="gray_r")
29  plt.show()
30
31  img = mpimg.imread("Apple\\0_100.jpg")
32
33  R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
34  imgGray = 0.299 * R + 0.587 * G + 0.114 * B
35  imgGray = np.array(imgGray, dtype="int")
36
37  img_apple = 255 - imgGray
38
39
40
```

## 04. 비슷한 샘플끼리 모으기

### ❖ 전체 소스 코드 (2/5)

```
41  img = mpimg.imread("Banana\\0_100.jpg")
42
43  R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
44  imgGray = 0.299 * R + 0.587 * G + 0.114 * B
45  imgGray = np.array(imgGray, dtype="int")
46
47  img_banana = 255 - imgGray
48
49  # 하나의 행과 두 개의 열을 지정함
50  fig, axs = plt.subplots(1, 2)
51  axs[0].imshow(img_apple, cmap="gray_r")
52  axs[1].imshow(img_banana, cmap="gray_r")
53  plt.show()
54
55  import os
56
57  cur_dir = os.getcwd()
58  fruit_list = ["Apple", "Banana", "Pineapple"]
59
60  fruit_npy = []
```

```
61  for fruit_name in fruit_list:
62      folder_name = cur_dir + "\\ " + fruit_name
63      file_list = os.listdir(folder_name)
64
65      for file_name in file_list:
66          img = mpimg.imread(folder_name + "\\ " + file_name)
67          R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
68          imgGray = 0.299 * R + 0.587 * G + 0.114 * B
69          imgGray = np.array(imgGray, dtype="int")
70          imgGray2 = 255 - imgGray
71          fruit_npy.append(imgGray2)
72
73  fruit_npy = np.array(fruit_npy)
74  print(fruit_npy.shape)
75
76  plt.figure()
77  plt.imshow(fruit_npy[600], cmap="gray_r")
78  plt.show()
79
80  np.save("fruits.npy", fruit_npy)
```

## 04. 비슷한 샘플끼리 모으기

### ❖ 전체 소스 코드 (3/5)

```
81  fruits = np.load("fruits.npy")
82  print(fruits.shape)
83
84  # 02. 픽셀값 분석하기
85  apple = fruits[0:490].reshape(-1, 100 * 100)
86  banana = fruits[490:980].reshape(-1, 100 * 100)
87  pineapple = fruits[980:].reshape(-1, 100 * 100)
88
89  print("사과 배열의 크기:", apple.shape)
90  print("바나나 배열의 크기:", banana.shape)
91  print("파인애플 배열의 크기:", pineapple.shape)
92
93  print(apple.mean(axis=1).shape)
94  print(apple.mean(axis=1))
95
96  plt.figure()
97  plt.hist(apple.mean(axis=1))
98  plt.hist(banana.mean(axis=1))
99  plt.hist(pineapple.mean(axis=1))
100 plt.show()
```

```
101 plt.figure()
102 plt.hist(apple.mean(axis=1), alpha=0.6, label="apple")
103 plt.hist(banana.mean(axis=1), alpha=0.6, label="banana")
104 plt.hist(pineapple.mean(axis=1), alpha=0.6,
105          label="pineapple")
106 plt.grid(True)
107 plt.legend()
108 plt.show()
109
110 fig, axs = plt.subplots(1, 3, figsize=(20, 5))
111 axs[0].bar(range(10000), np.mean(apple, axis=0))
112 axs[1].bar(range(10000), np.mean(banana, axis=0))
113 axs[2].bar(range(10000), np.mean(pineapple, axis=0))
114 axs[0].set_title("apple")
115 axs[1].set_title("banana")
116 axs[2].set_title("pineapple")
117 axs[0].set_ylim([0, 250])
118 axs[1].set_ylim([0, 250])
119 axs[2].set_ylim([0, 250])
120 plt.show()
```

## 04. 비슷한 샘플끼리 모으기

### ❖ 전체 소스 코드 (4/5)

```
121 apple_mean = np.mean(apple, axis=0).reshape(100, 100)
122 banana_mean = np.mean(banana, axis=0).reshape(100, 100)
123 pineapple_mean = np.mean(pineapple,
124                          axis=0).reshape(100, 100)
125
126 fig, axs = plt.subplots(1, 3, figsize=(10, 5))
127 axs[0].imshow(apple_mean, cmap="gray_r")
128 axs[1].imshow(banana_mean, cmap="gray_r")
129 axs[2].imshow(pineapple_mean, cmap="gray_r")
130 axs[0].set_title("apple")
131 axs[1].set_title("banana")
132 axs[2].set_title("pineapple")
133 plt.show()
134
135 # 03. 평균값과 가까운 사진 고르기
136 abs_diff = np.abs(fruits - apple_mean)
137 abs_mean = np.mean(abs_diff, axis=(1,2))
138 print(abs_mean.shape)
139
140
```

```
141 apple_index = np.argsort(abs_mean)[:100]
142
143 fig, axs = plt.subplots(10, 10, figsize=(5, 5))
144 for j in range(10):
145     for k in range(10):
146         axs[j, k].imshow(fruits[apple_index[j * 10 + k]],
147                          cmap="gray_r")
148         axs[j, k].axis("off")
149 plt.show()
150
151 abs_diff = np.abs(fruits - banana_mean)
152 abs_mean = np.mean(abs_diff, axis=(1, 2))
153 banana_index = np.argsort(abs_mean)[:100]
154 fig, axs = plt.subplots(10, 10, figsize=(5, 5))
155 for j in range(10):
156     for k in range(10):
157         axs[j, k].imshow(fruits[banana_index[j * 10 + k]],
158                          cmap="gray_r")
159         axs[j, k].axis("off")
160 plt.show()
```

## 04. 비슷한 샘플끼리 모으기

### ❖ 전체 소스 코드 (5/5)

```
161 abs_diff = np.abs(fruits - pineapple_mean)
162 abs_mean = np.mean(abs_diff, axis=(1, 2))
163 pineapple_index = np.argsort(abs_mean)[:100]
164
165 fig, axs = plt.subplots(10, 10, figsize=(5, 5))
166 for j in range(10):
167     for k in range(10):
168         axs[j, k].imshow(fruits[pineapple_index[j*10+k]],
169                           cmap="gray_r")
170         axs[j, k].axis("off")
171 plt.show()
```

## 05. 마무리

- 01. 과일 사진 데이터 준비하기
- 02. 픽셀값 분석하기
- 03. 평균값과 가까운 사진 고르기
- 04. 비슷한 샘플끼리 모으기

## 05. 마무리

### ❖ 키워드로 끝내는 핵심 포인트 (1/2)

#### 비지도 학습(Unsupervised Learning)

- ✓ 머신러닝의 한 종류로 학습 데이터에 타겟(레이블)이 없음
- ✓ 타겟이 없기 때문에 외부의 도움 없이 스스로 유용한 무언가를 학습해야 함
- ✓ 대표적인 비지도 학습 작업으로는 군집, 차원 축소 등이 있음

#### 히스토그램(Histogram)

- ✓ 구간별로 값이 발생한 빈도를 그래프로 표시한 것임
- ✓ 보통 x축이 값의 구간(계급)이고, y축은 발생 빈도(도수)임

## 05. 마무리

### ❖ 키워드로 끝내는 핵심 포인트 (2/2)

#### 군집 알고리즘(Clustering Algorithm)

- ✓ 비슷한 샘플끼리 하나의 그룹으로 모으는 대표적인 비지도 학습 작업임
- ✓ 군집 알고리즘으로 모은 샘플 그룹을 클러스터(Cluster)라고 부름



## 05. 마무리

### ❖ 확인 문제 1.

히스토그램을 그릴 수 있는 맷플롯립 함수는 무엇인가요?

- ① hist( )
- ② scatter( )
- ③ plot( )
- ④ bar( )

## 05. 마무리

### ❖ 확인 문제 2.

수업 중에 했던 것처럼 바나나 사진의 평균 `banna_mean`과 비슷한 사진 100장을 찾아 출력해 보세요. 바나나 사진을 모두 찾을 수 있나요?

- ❖ 01. 과일 사진 데이터 준비하기
- ❖ 02. 픽셀값 분석하기
- ❖ 03. 평균값과 가까운 사진 고르기
- ❖ 04. 비슷한 샘플끼리 모으기
- ❖ 05. 마무리

# THANK YOU!

## Q & A

- Name: 권범
- Office: 동덕여자대학교 인문관 B821호
- Phone: 02-940-4752
- E-mail: [bkwon@dongduk.ac.kr](mailto:bkwon@dongduk.ac.kr)