

기계학습기초

Hongik Univ.

# 과일 분류 머신러닝

기말발표

# 목차

10월 19일

- 프로젝트 목표 정의

- 넘파이 연산만을 사용한 CNN 모델 설계

- 데이터 전처리과정

- Convolution Layer 구현

- Pooling Layer 구현

- Affine Layer 구현

- 1차 테스트 결과 확인 및 계획 수정

- KERAS 라이브러리를 활용한 CNN모델 설계

- 초기 레이어 구성

- 컨볼루션 레이어 중첩하여 레이어 층 증가

- 컨볼루션-풀링 사이클 2회에서 3회로 증가

- Overfitting 문제 해결

- 데이터셋 구성 변화

- Final Insight



## 프로젝트 목표 정의

10월 19일

### 프로젝트 목표 정의

### 넘파이 연산만을 사용한 CNN 모델 설계

### 1차 테스트 결과 확인 및 계획 수정

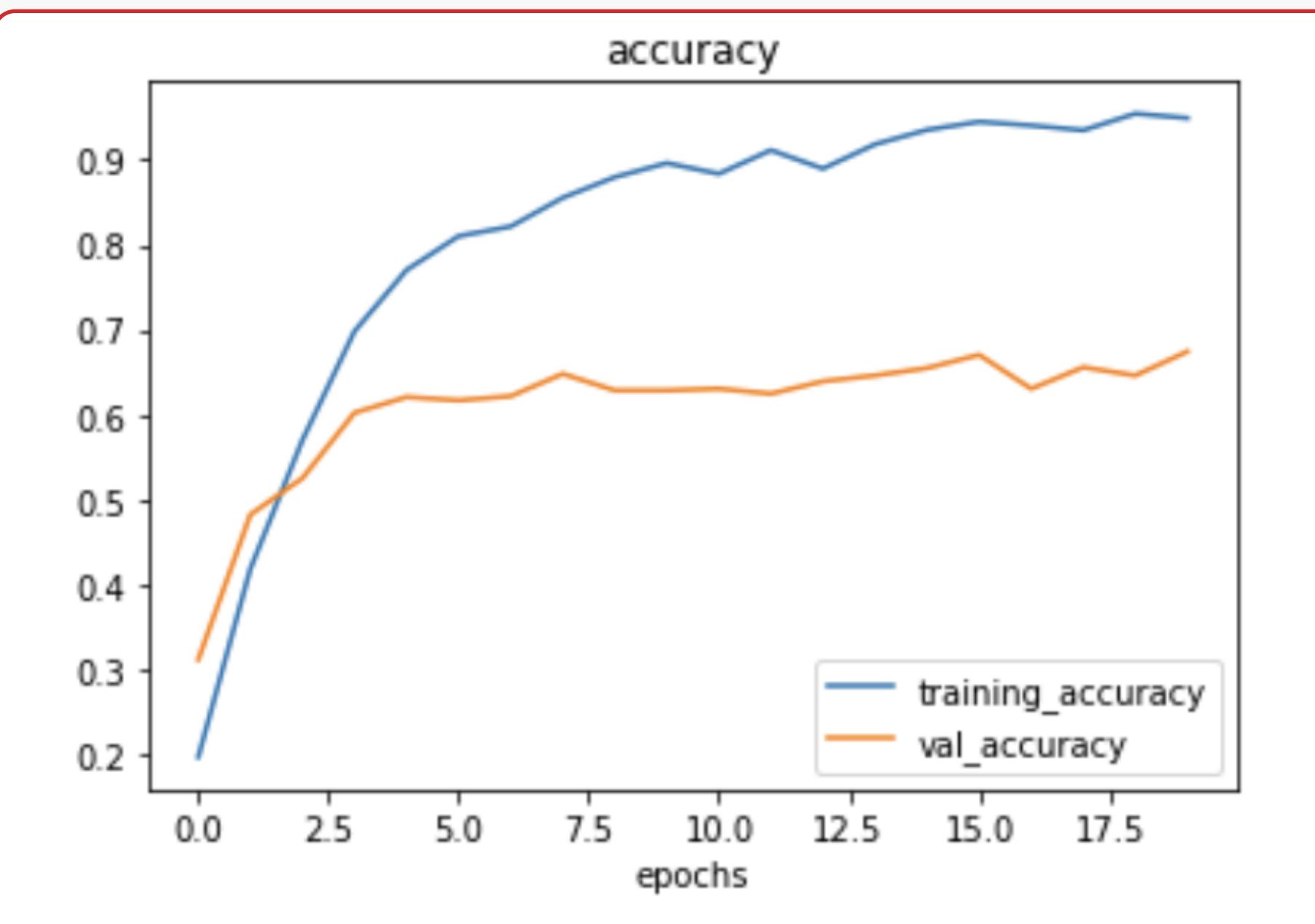
### KERAS 라이브러리를 활용한 CNN모델 설계

### Final Insight

```
72/72 [=====] - 42s 583ms/step - loss: 0.1519 - accuracy: 0.9492 - val_loss:  
2.9711 - val_accuracy: 0.6741
```

시작하기에 앞서, 데이터의 출처인 **Kaggle**에서 같은 데이터셋을 사용한 다른 과일분류 ML모델의 정확도들을 비교

KERAS 라이브러리를 활용한 예제들의 경우, 정확도가 67%에서 75% 사이에 분포함을 파악



KERAS 라이브러리를 사용하지 않고, 넘파이연산만을 사용하여 **기초적인 토대부터 재설계하여 KERAS를 사용하였을 때의 정확도를 구현**하는 것을 1차 목표로 설정

같은 과일 데이터셋을 사용하여 다른 사람이 설계한 머신러닝 모델의 플랏함수

Test Accuracy= 67.41%

```

from PIL import Image as img
from numpy import asarray
import numpy as np
import os
np.set_printoptions(threshold=np.inf)

def shuffle_dataset(x, t):
    """
    데이터셋을 뒤섞는다.

    Parameters
    -----
    x : 훈련 데이터
    t : 정답 레이블

    Returns
    -----
    x, t : 뒤섞은 훈련 데이터와 정답 레이블
    """
    permutation = np.random.permutation(x.shape[0])
    x = x[permutation,:,:] if x.ndim == 2 else x[permutation,:,:,:]
    t = t[permutation]

    return x, t

def load_fruit():
    x_train, t_train, x_test, t_test = [],[],[],[]
    base_dir = "./dataset/"
    fruit=['apple','banana','blueberry','cherry','coconut','grape','kiwi','lemon','pear','tomato']

    for i in range(len(fruit)):
        file_list = os.listdir(base_dir+fruit[i])
        for j in range(800):
            image=img.open(base_dir+fruit[i]+"/"+file_list[j])
            data=asarray(image)
            x_train.append(data)
            t_train.append(i)
        for j in range(800, 1000):
            image=img.open(base_dir+fruit[i]+"/"+file_list[j])
            data=asarray(image)
            x_test.append(data)
            t_test.append(i)

    x_train, t_train = np.array(x_train), np.array(t_train)
    x_train = x_train.transpose(0, 3, 2, 1)
    x_train, t_train = shuffle_dataset(x_train, t_train)

    x_test, t_test = np.array(x_test), np.array(t_test)
    x_test = x_test.transpose(0, 3, 2, 1)
    x_test, t_test = shuffle_dataset(x_test, t_test)

    return (x_train, t_train), (x_test, t_test)

```

## 10개의 과일을 선정하여 각 과일이 담긴 이미지들을 1000 개씩 인풋으로 삽입

os.listdir()함수를 통해 파일명의 리스트를 가져온 후, Image.open() 함수에 필요한 디렉토리 인자에 삽입하여 이미지를 가져옴

## 10000장의 jpg파일을 향후 Convolution처리가 용이한 차원의 배열로 형태변환을 진행

누적되어있던 datas array를 np.array()함수를 통해 넘파이배열로 변환한 후, reshape()함수를 사용

**[10(과일 종류)\*1000(사진개수)\*3(RGB)\*104(세로픽셀높이)\*128(가로픽셀너비)] shape을 띤 5차원 배열로 최종변환**

**Conclusion:** 성공적으로 0부터 255까지의 값으로 구성된 5차원 배열이 최종 반환된다는 사실을 확인하였음

```

class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        # 중간 데이터(backward 시 사용)
        self.x = None
        self.col = None
        self.col_W = None

        # 가중치와 편향 매개변수의 기울기
        self.dW = None
        self.db = None

    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h = 1 + int((H + 2 * self.pad - FH) / self.stride)
        out_w = 1 + int((W + 2 * self.pad - FW) / self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out

    def backward(self, dout):
        FN, C, FH, FW = self.W.shape
        dout = dout.transpose(0, 2, 3, 1).reshape(-1, FN)

        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

        return dx

```

```

[Running] python -u
before convolution layer
(8000, 3, 104, 128)

after convolution layer
(8000, 10, 52, 64)
(filter_num: 10, filter size: 3X3, stride: 2, padding: 1)

[Done] exited with code=0 in 63.775 seconds

```

정상동작 여부 파악 위해 콘솔창을 통해 컨볼루션 레이어를 거치면서 생기는 shape의 변화를 파악

**Convolution 클래스를 선언할 때 초기화한 weight값과 forward함수의 매개변수인 x의 shape를 받아오기**

FN, FC, FH, FW에 filter의 shape를 받아오고, N, C, H, W에 input인 x의 shape를 받아옴

**Filter 적용으로 변하는 image의 크기 계산**

$\text{output\_height} = 1 + ((\text{height} + 2 * \text{pad} - \text{filter\_height}) / \text{stride})$

$\text{output\_width} = 1 + ((\text{width} + 2 * \text{pad} - \text{filter\_width}) / \text{stride})$

**input x와 weight(filter)의 행렬 곱 연산이 오래 걸리므로 im2col 함수를 사용하여 시간소요를 줄임**

input x를  $((N \times \text{output\_h} \times \text{output\_W}) \times (\text{FC} \times \text{FH} \times \text{FW}))$  형태로 바꾸고, weight(filter)를  $((\text{FC} \times \text{FH} \times \text{FW}) \times \text{FN})$  형태로 변형

변형된 x와 weight를 행렬곱

**np.transpose() 함수를 사용하여 (N, channel, height ,width) 형태로 맞춰준다.**

**Conclusion:** Convolution 클래스의 forward 함수 통과 결과, 예상했던 4차원 배열의 형태가 반환되었음

```

class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.x = None
        self.argmax = None

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h * self.pool_w)

        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        self.x = x
        self.argmax = arg_max

        return out

    def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
        dmax = np.zeros((dout.size, pool_size))
        dmax[np.arange(self.argmax.size), self.argmax.flatten()] = dout.flatten()
        dmax = dmax.reshape(dout.shape + (pool_size,))

        dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
        dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)

        return dx

```

before pooling layer  
(8000, 10, 52, 64)

after pooling layer  
(8000, 10, 26, 32)  
(filter\_h: 2, filter\_w: 2, stride: 1, padding: 0)

정상동작 여부 파악 위해 콘솔창을 통해 풀링 레이어를 거치  
면서 생기는 shape의 변화를 파악

### 오버피팅을 방지하고자, 모든 데이터들 중 일부만을 다음 레이어 로 가져감

Im2col를 이용해 4차원 numpy array를 2차원으로 변환하여 최대값 도출  
다시 원래의 차원으로 array shape를 변경

**Conclusion:** 반환하는 배열의 사이즈가 Input 배열의 절반  
인 것을 확인하였으며, 컨볼루션 레이어와의 연동또한 정상작  
동함을 확인하였음

# Affine Layer 구현완료

11월 16일

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 기중치와 편향 매개변수의 미분
        self.dW = None
        self.db = None

    def forward(self, x):
        # 텐서 차원 변환
        self.x = x.reshape(*self.original_x_shape) # 입력 데이터 모양 변경(텐서 대응)
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx = dx.reshape(*self.original_x_shape) # 입력 데이터 모양 변경(텐서 대응)
        return dx
```

```
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None # 손실함수
        self.y = None # softmax의 출력
        self.t = None # 정답 레이블(원-핫 인코딩 형태)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size: # 정답 레이블이 원-핫 인코딩 형태일 때
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

**Affine레이어와 Softmax 함수 구현을 통해 기초적인 머신러닝에 필요한 모든 레이어 및 함수를 구현완료하였음**

# 1차 테스트 결과 확인 및 전반적인 방향 설정

11월 16일

프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight

```
train loss:0.5045425442800055  
train loss:0.9403085050395106  
train loss:1.0991652887796508  
train loss:0.7531874960010319  
train loss:0.9959220360854784  
train loss:1.09814429688019  
train loss:1.0372499626969534  
train loss:0.952459620463891  
train loss:0.9423219616071234  
train loss:0.8110882712103289  
train loss:0.9353458332969738  
train loss:0.9791949972322902  
train loss:1.057941150723164  
train loss:0.9788707389113166  
train loss:0.9518831080864594  
train loss:0.8784503077734611  
train loss:0.9080532582256896  
train loss:1.2496200883593294  
train loss:0.8137032110087645  
train loss:0.9610811552920384  
train loss:0.9403992407482104  
train loss:1.3044999950216656  
train loss:1.120664984543576  
train loss:1.0792425659788503  
train loss:1.0227578546644842  
train loss:1.01544627829637  
train loss:0.8800101739774132  
train loss:1.011327900824992  
train loss:0.9710557233800767  
train loss:1.0869069825115885  
train loss:1.014379915456273  
train loss:0.8363056103048517  
train loss:0.94864867441545  
train loss:0.8463695055797453  
train loss:0.9955680096631333  
train loss:1.0401552575420123  
train loss:1.0507866488188817  
train loss:1.3056149747232138  
train loss:0.7991947058166734  
train loss:1.0061459663908126  
train loss:1.0039959101461653  
train loss:0.9053990671637773  
train loss:1.0407076885111393  
===== Final Test Accuracy =====  
test acc:0.3125
```

종료 코드 0(으)로 완료된 프로세스

**1차 테스트 결과 31.25%의 정확도로 목표치에 미치지 못함**

매 시도마다 3-4시간이 소요되어, 해당 모델을 통해 정확도 향상을 위한 세부수정이 힘들 것이라 판단



**KERAS 라이브러리로 최적의 정확도를 도출하는 레이어 구성을 먼저 파악하는 것으로 목표를 수정**

## KERAS를 활용한 결과 분석 1

11월 17일

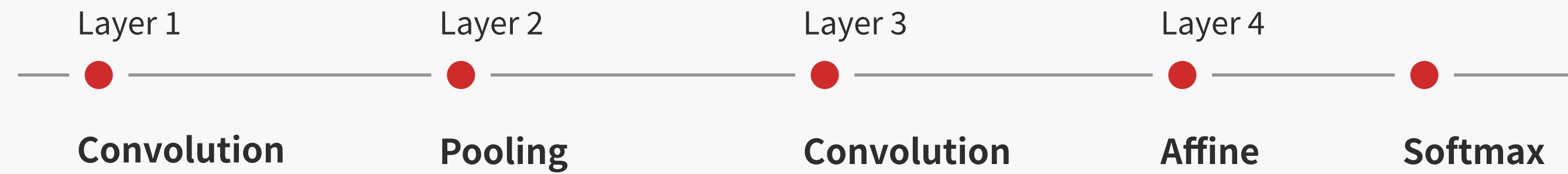
프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

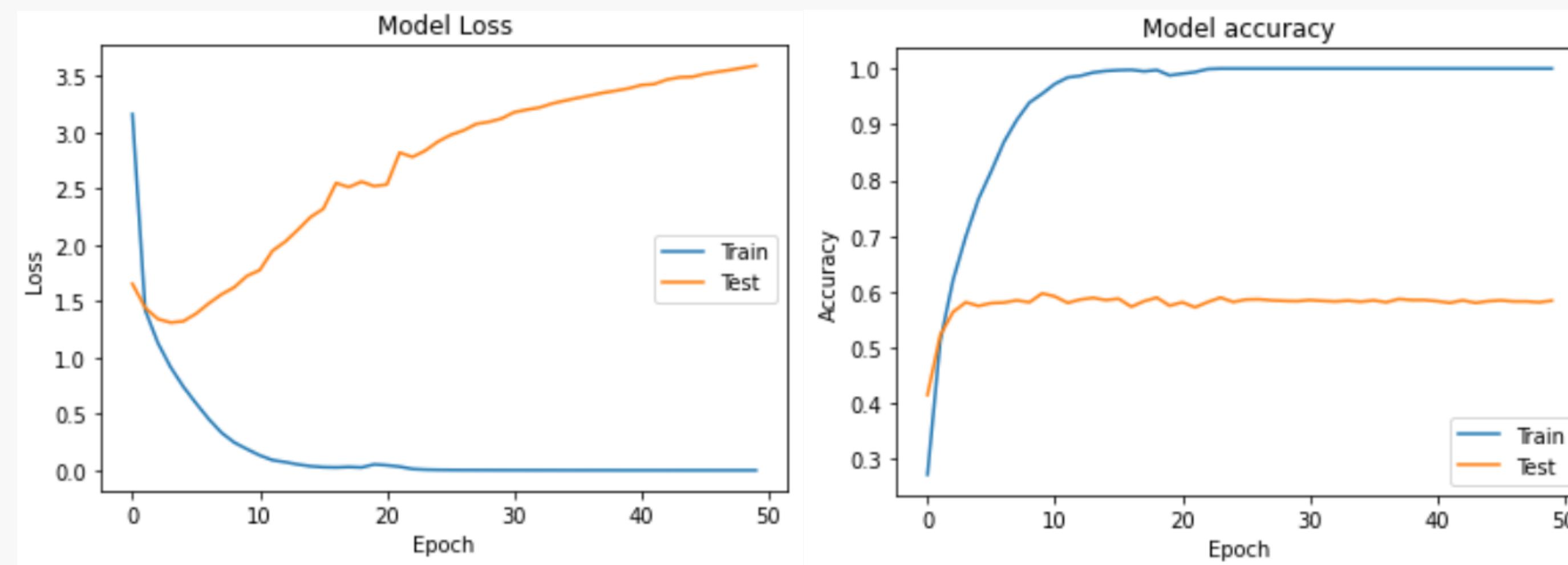
KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight



### Insight

- 정확도가 낮은 원인을 파악하고자 Kaggle, 등 다수의 CNN 사례에 대해 리서치를 해본 결과, 절대적인 레이어 층이 얕아 학습이 충분히 되지 않은 상태임을 파악



총 레이어: 4층

Loss: 3.590419292449951, Acc: 0.5839999914169312

### Conclusion

풀링을 진행하면서 이미지의 사이즈가 줄어들기 이전에 컨볼루션 단계를 중첩하여 레이어 층을 늘리기로 계획함

## KERAS를 활용한 결과 분석 2

11월 18일

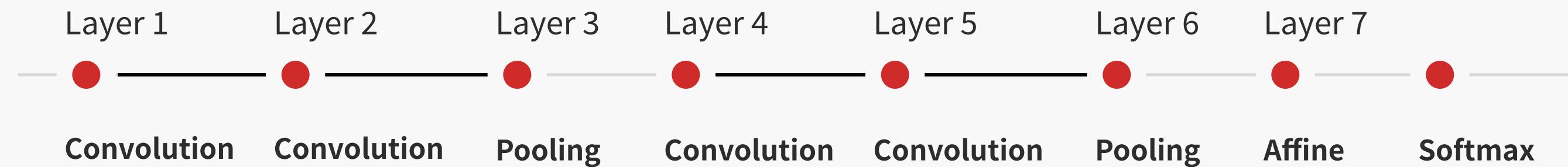
프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight



### Changes

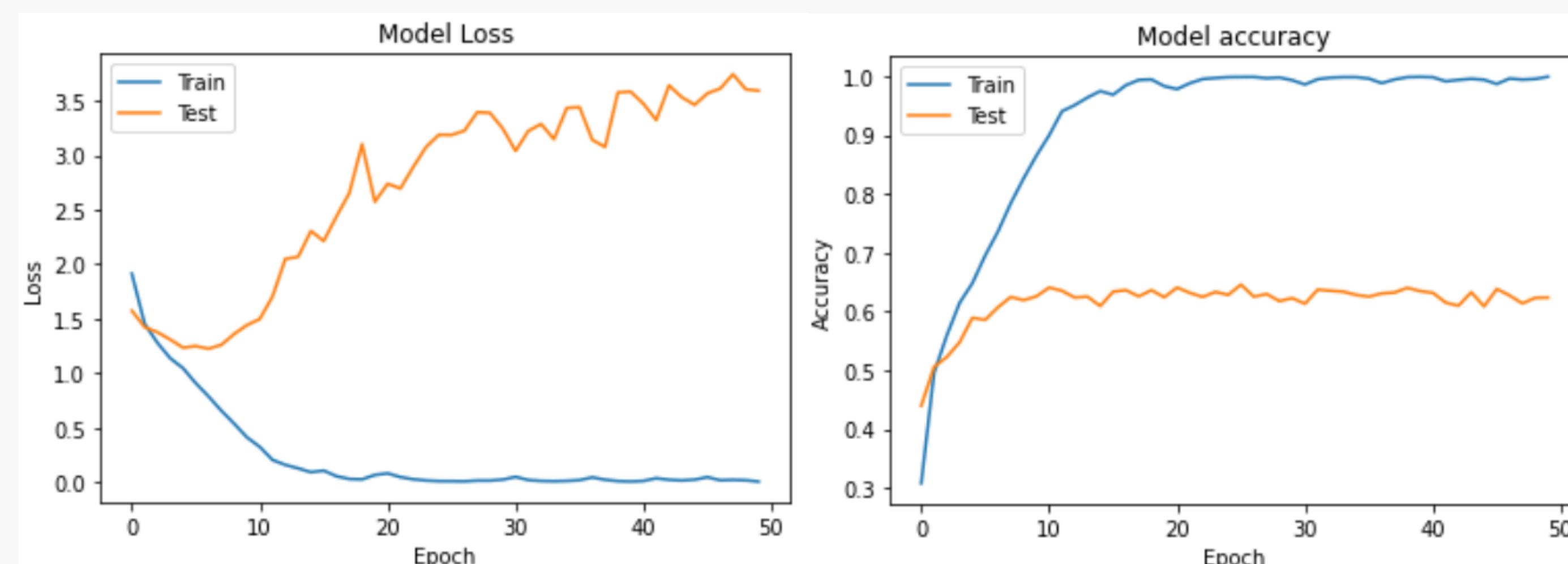
- Pooling을 거치기 전에 Convolution Layer를 한번만 거쳤던 기준과 달리 Convolution layer를 두번 중첩하여 총 7층을 구성하였음

### Insight

- 정확도 개선은 있었으나 우리가 목표로 했던 정확도에 미치지 못하였으며, 레이어 층 또한 리서치했던 알고리즘에 비해 아직 얕다고 분석하였고, 이에 대한 원인으로 데이터셋의 사이즈, 그리고 Convolution 레이어의 Stride 인자를 꼽았음

### Conclusion

데이터셋 이미지 사이즈 자체를  $104 \times 128$ 에서  $208 \times 256$ 으로 교체하였으나, 학습하는 데에 시간이 많이 소요되었음. 이에 기존 사이즈를 유지하되, Convolution에 전달되는 stride 인자 값을 1로 조정하여 컨볼루션을 거쳐도 원래 사이즈를 유지할 수 있도록 하였음



총 레이어: 7층

Loss: 3.5923757553100586, Acc: 0.6234999895095825

## KERAS를 활용한 결과 분석 3

11월 20일

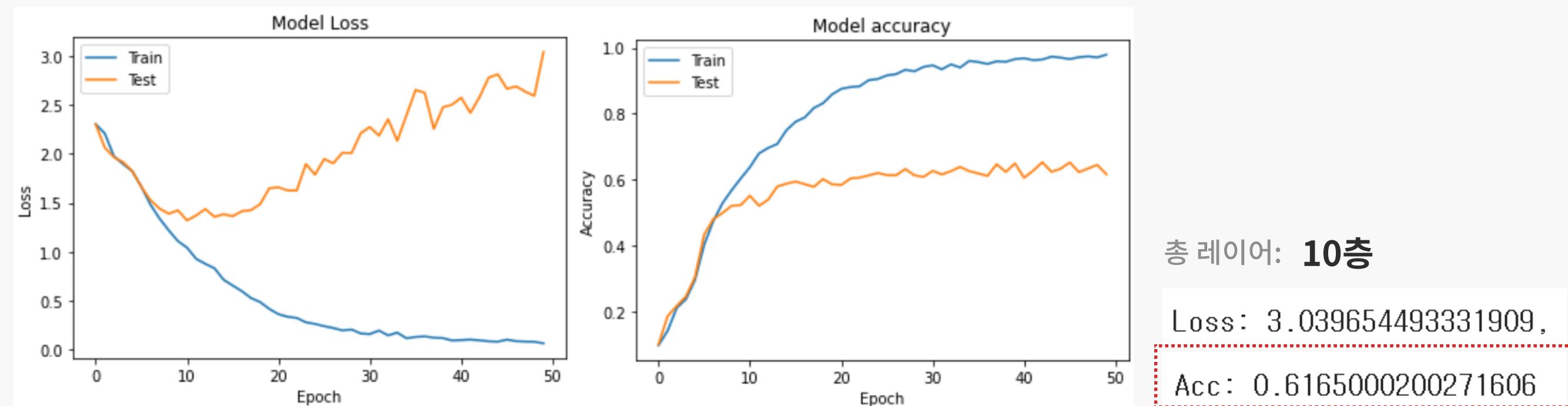
프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight

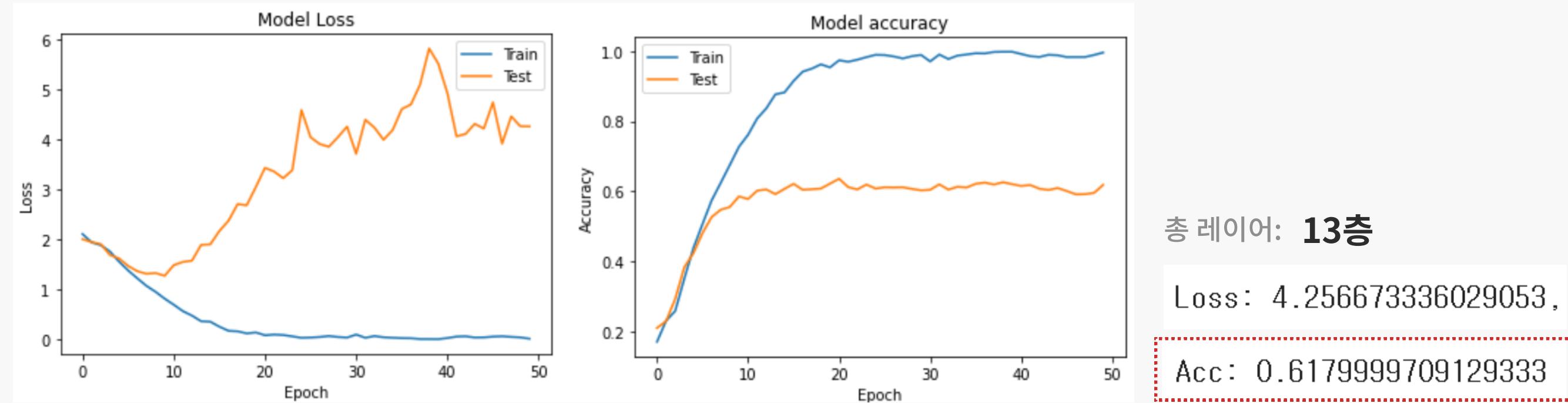
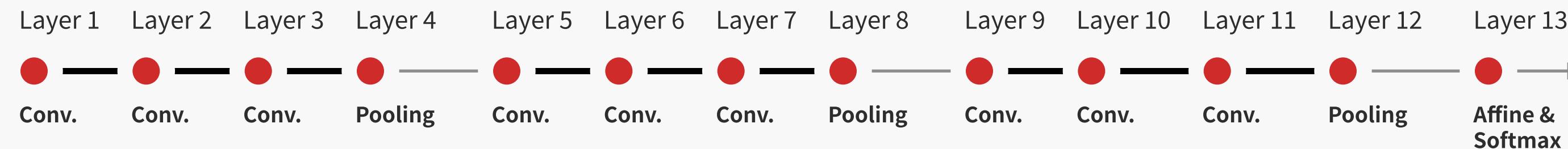


### Changes

1. Convolution - Pooling 사이클을 기존 2번에서 3번으로 늘림
2. 매 사이클에서의 Convolution 레이어 개수를 2에서 3으로 늘림

### Insight

- 층을 누적시켰음에도 정확도에서의 큰 차이를 찾아볼 수 없었음. 오히려, Epoch가 진행됨에 따라 test Accuracy가 떨어졌으며, 이에 대한 원인으로 Overfitting을 꼽음



### Conclusion

Regularization 기법 중 하나인 Dropout 레이어를 삽입하여 이를 해결하고자 하였음

## KERAS를 활용한 결과 분석 4

11월 21일

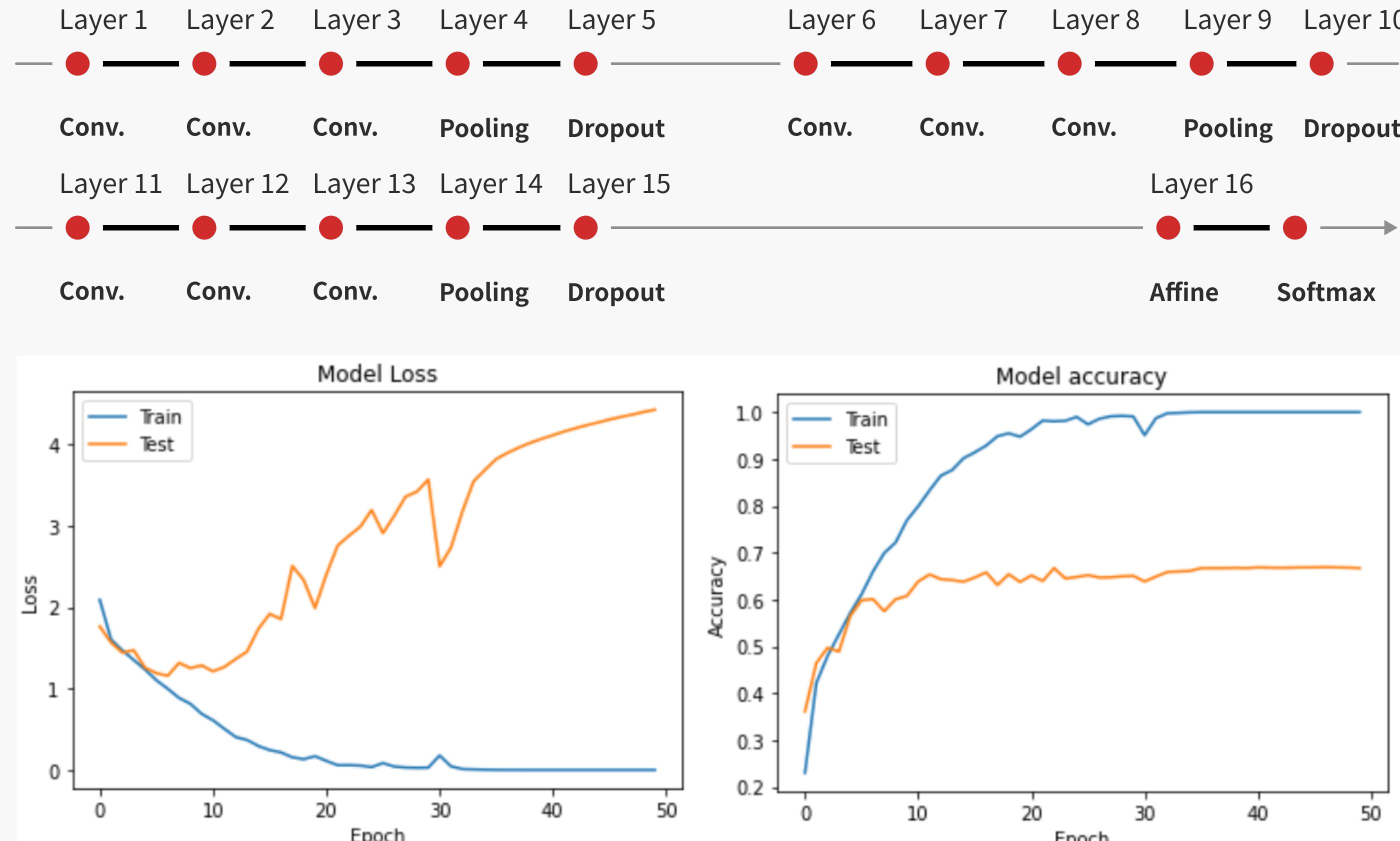
프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight



총 레이어: 16층

Loss: 4.42233419418335 , Acc: 0.6669999957084656

### Changes

- 매 사이클의 끝에 Dropout 레이어를 추가하였음

### Insight

- Regularization 기법을 적용 한 후 정확도가 유의미하게 상승하였음. 허나 여러번 시도하였음에도 목표했던 67.41%를 근소하게 넘기지 못했음.

### Conclusion

참고하였던 예제 코드와 다른 파일 데이터셋을 사용하였기에, 목표했던 정확도에 미치지 못하는 원인이 데이터셋일 것이라는 가설을 세움

## 현재 모델에 삽입되어 있는 데이터셋

정확도: 62.4%-66.6%

빨강색 Feature 공유



Apple

Cherry

Tomato

3개

노랑색 Feature 공유



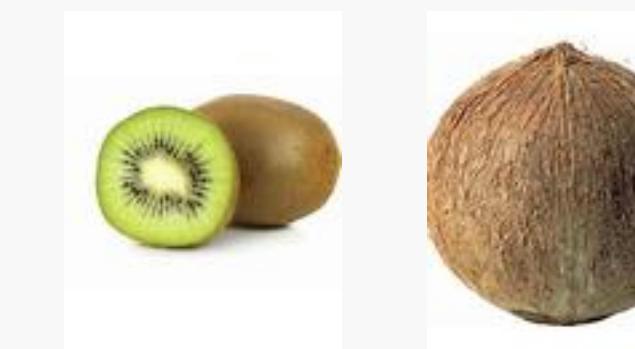
Lemon

Pear

Banana

3개

갈색 Feature 공유 2개



Kiwi

Coconut

2개

보라색 Feature 공유 2개



Grape

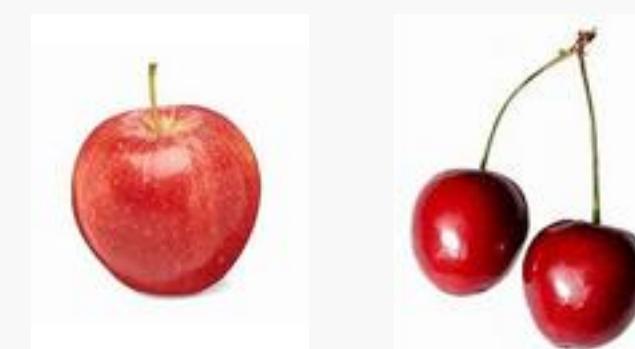
Blueberry



## 다른 예제코드에서 사용된 데이터셋

정확도: 67.41%

빨강색 Feature 공유 2개



Salak

Cherry

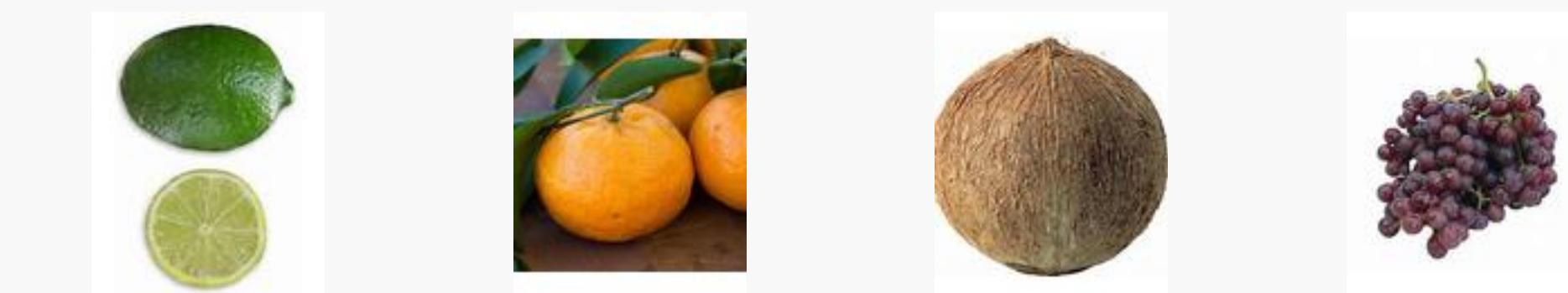
노랑색 Feature 공유 2개



Lemon

Banana

공유되는 색상 없음

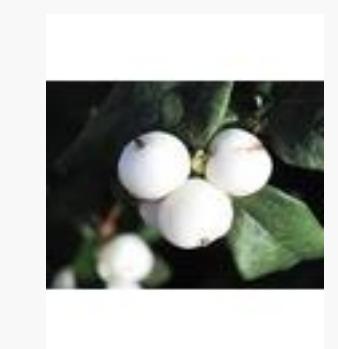


Lime

Mandarine

Coconut

Grape



Snow Berry

### Insight

- 현재 모델이 사용하는 데이터셋의 경우 같은 색상을 공유하는 과일이 적어도 2개씩 있는 데에 반해, 정확도가 더 높은 예제코드의 경우 5개의 과일이 모두 고유한 색상을 지니고 있음

### Conclusion

데이터셋의 차이가 두 모델 간의 정확도 차이가 발생하는 주 원인으로 판단하였음

## 데이터셋 분석\_2

11월 22일 • 데이터셋이 결과에 미치는 영향

프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight



Snow Berry



white Currant



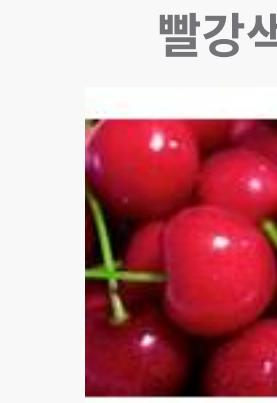
Sugar Apple



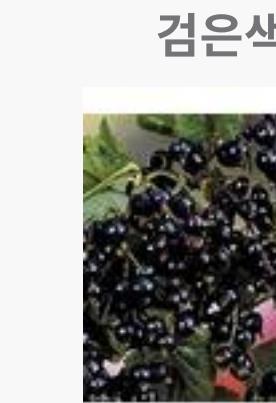
Mandarine



Salak



Cherry



Black currant



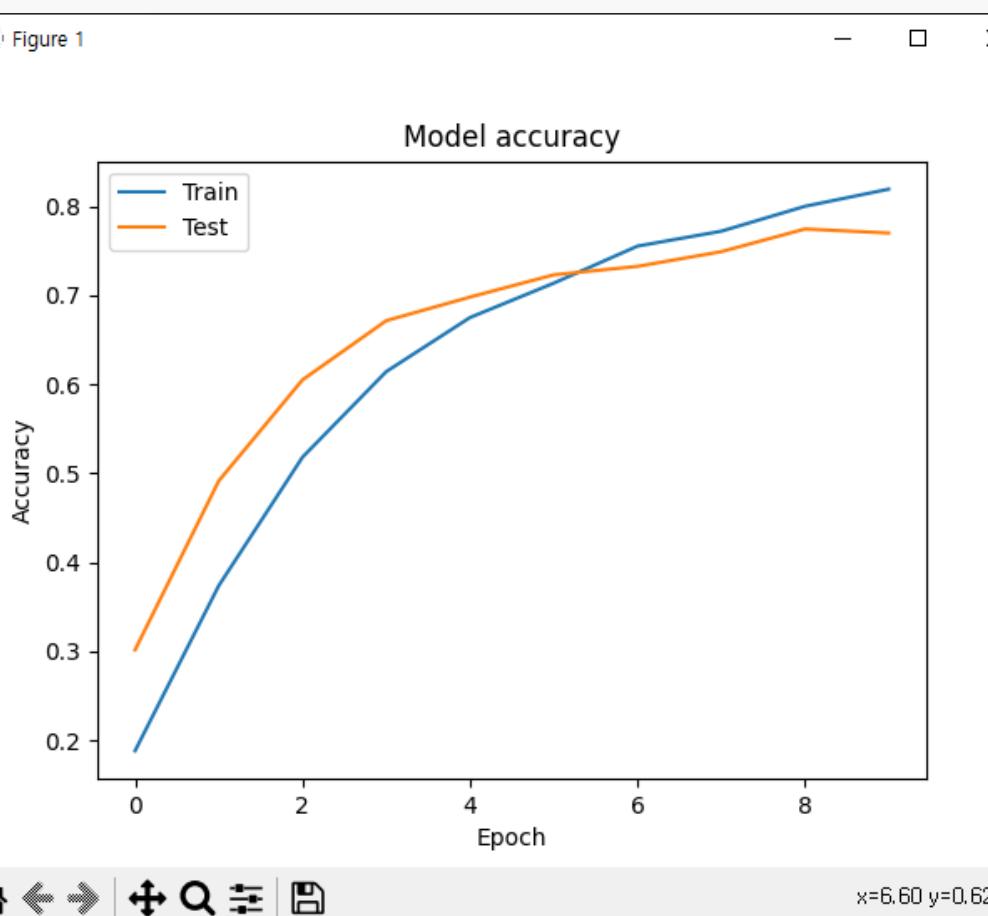
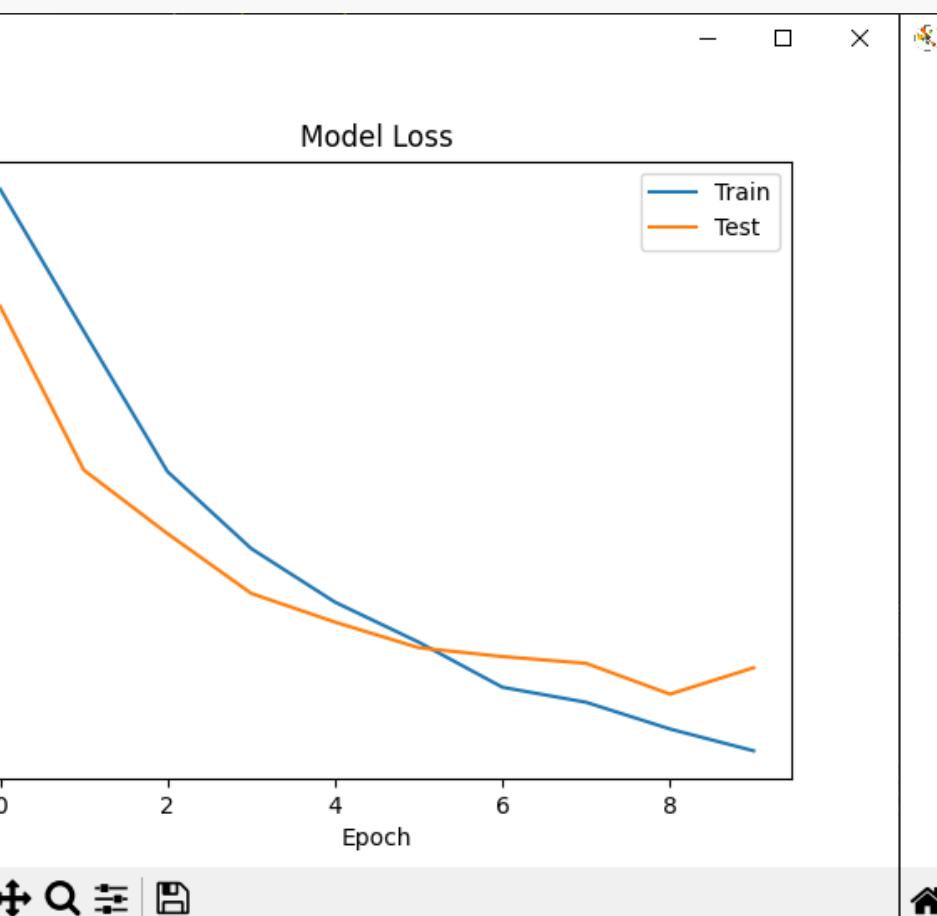
Banana



Lime



Blueberry



```

40/40 [=====] - 137s 3s/step - loss: 0.7328 - accuracy: 0.7552 - val_loss: 0.8212 - val_accuracy: 0.7325
Epoch 8/10
40/40 [=====] - 137s 3s/step - loss: 0.6895 - accuracy: 0.7720 - val_loss: 0.8016 - val_accuracy: 0.7490
Epoch 9/10
40/40 [=====] - 137s 3s/step - loss: 0.6126 - accuracy: 0.8000 - val_loss: 0.7130 - val_accuracy: 0.7745
Epoch 10/10
40/40 [=====] - 137s 3s/step - loss: 0.5499 - accuracy: 0.8192 - val_loss: 0.7891 - val_accuracy: 0.7700
63/63 - 7s - loss: 0.7891 - accuracy: 0.7700 - 7s/epoch - 105ms/step

Loss: 0.7890963554382324, Acc: 0.7699999809265137

```

### 의도:

데이터를 분류하는 과정에서 색상 Feature가 Convolution 결과에 미치는 영향을 파악하고자, 최대한 이미지 픽셀의 색상이 겹치지 않는 과일 위주로 데이터셋을 재구성하여 학습

### 정확도:

66% ➡ 76.99%

### Conclusion:

사용하는 색상이 다소 겹치는 저번 데이터셋 구성과 달리 정확도가 큰 폭으로 증가함을 확인할 수 있었으며, Test 데이터 간의 overfitting 문제 또한 해결되었음

## 데이터셋 분석\_3

11월 22일 • 데이터셋이 결과에 미치는 영향

프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight



Acerla



Apple



Mabolo



Apricot



Jamaica Cherry



Cherry



Camu camu



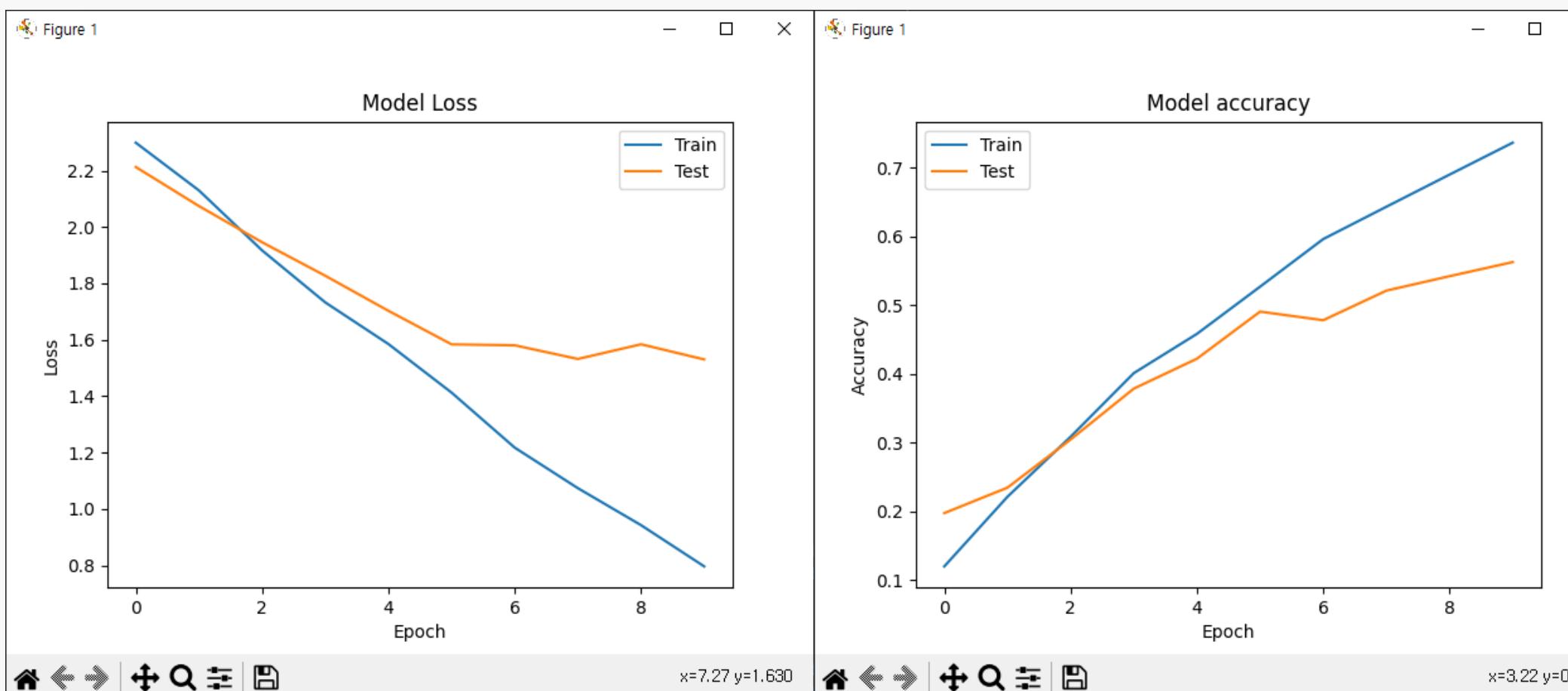
Malay Apple



Goumi



Jocote



의도:

빨강색을 띠는 과일들 만으로 데이터셋을 재구성하여, 색상 Feature가 Convolution 과정 간 분류에 핵심적인 기준이 된다는 것을 증명하고자 하였음

정확도:

**56.25%**

Conclusion:

예상했던 대로, 다른 색상들로 구성한 데이터셋 대비 훨씬 낮은 정확도를 보여, Convolution 과정 간 색상 Feature의 중요성이 증명되었음. 이와 같은 맥락으로 저번 구성에서 보이지 않았던 Overfitting 현상도 loss값의 변화를 통해 확인해볼 수 있었음

```

40/40 [=====] - 136s 3s/step - loss: 1.2177 - accuracy: 0.5900 - val_loss: 1.5800 - val_accuracy: 0.4780
Epoch 8/10
40/40 [=====] - 136s 3s/step - loss: 1.0739 - accuracy: 0.6430 - val_loss: 1.5321 - val_accuracy: 0.5210
Epoch 9/10
40/40 [=====] - 138s 3s/step - loss: 0.9428 - accuracy: 0.6898 - val_loss: 1.5842 - val_accuracy: 0.5420
Epoch 10/10
40/40 [=====] - 136s 3s/step - loss: 0.7966 - accuracy: 0.7361 - val_loss: 1.5305 - val_accuracy: 0.5625
63/63 - 7s - loss: 1.5305 - accuracy: 0.5625 - 7s/epoch - 108ms/step

Loss: 1.5304561853408813, Acc: 0.5625

```

## 데이터셋 분석\_4

11월 24일 • 데이터셋이 결과에 미치는 영향

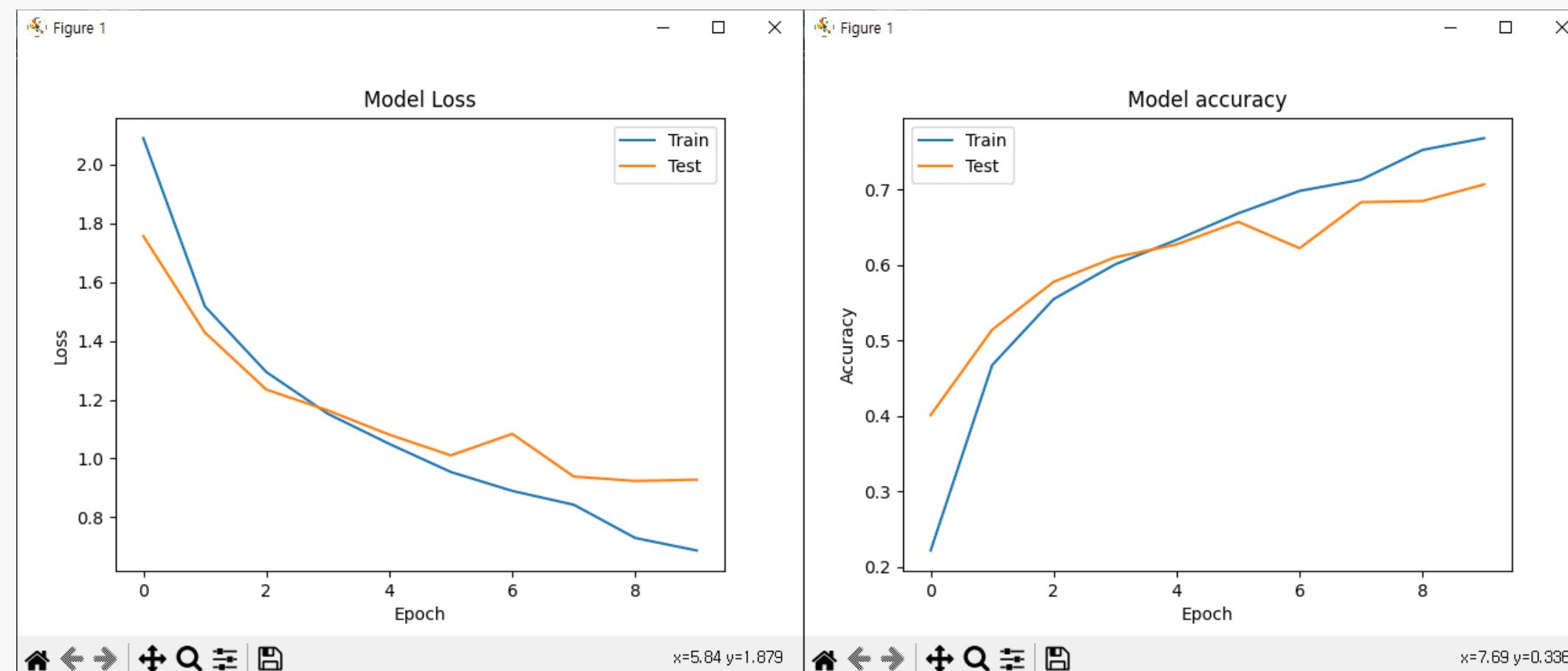
프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight



의도:

아까 전 구성이 색상 Feature이 미치는 영향을 파악하기 위한 데이터셋이였다면, 이번에는 과일의 실루엣과 형태 Feature가 그룹핑에 주는 영향을 파악하고자 개성적인 형태를 띤 과일들로 데이터셋들을 재구성하였음

정확도:

66% ► **70.64%**

Conclusion:

정확도는 색상 Feature을 중심으로 데이터셋을 재구성하였을 때에 비해 낮았으나, 70대의 정확도를 보임을 통해 형태 또한 색상과 마찬가지로 과일분류에 있어 중요한 Feature라는 것을 확인할 수 있었음

```

48/40 [=====] - 141s 4s/step - loss: 0.8901 - accuracy: 0.6979 - val_loss: 1.0842 - val_accuracy: 0.6220
Epoch 8/10
40/40 [=====] - 141s 4s/step - loss: 0.8430 - accuracy: 0.7128 - val_loss: 0.9389 - val_accuracy: 0.6830
Epoch 9/10
40/40 [=====] - 141s 4s/step - loss: 0.7299 - accuracy: 0.7523 - val_loss: 0.9239 - val_accuracy: 0.6845
Epoch 10/10
40/40 [=====] - 140s 4s/step - loss: 0.6873 - accuracy: 0.7678 - val_loss: 0.9283 - val_accuracy: 0.7065
63/63 - 7s - loss: 0.9283 - accuracy: 0.7065 - 7s/epoch - 110ms/step

Loss: 0.9282699823379517, Acc: 0.7064999938011169

```

# 최종 KERAS 결과

11월 24일

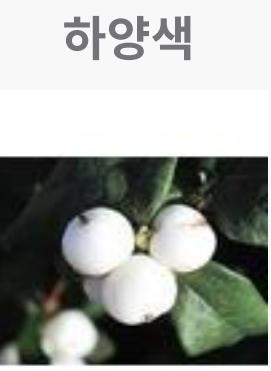
프로젝트 목표 정의

넘파이 연산만을 사용한 CNN 모델 설계

1차 테스트 결과 확인 및 계획 수정

KERAS 라이브러리를 활용한 CNN모델 설계

Final Insight



Snow Berry



white Currant



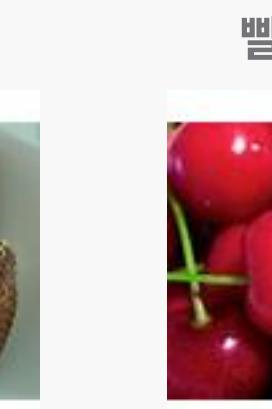
Sugar Apple



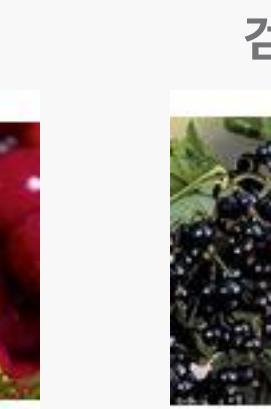
Mandarine



Salak



Cherry



Black currant



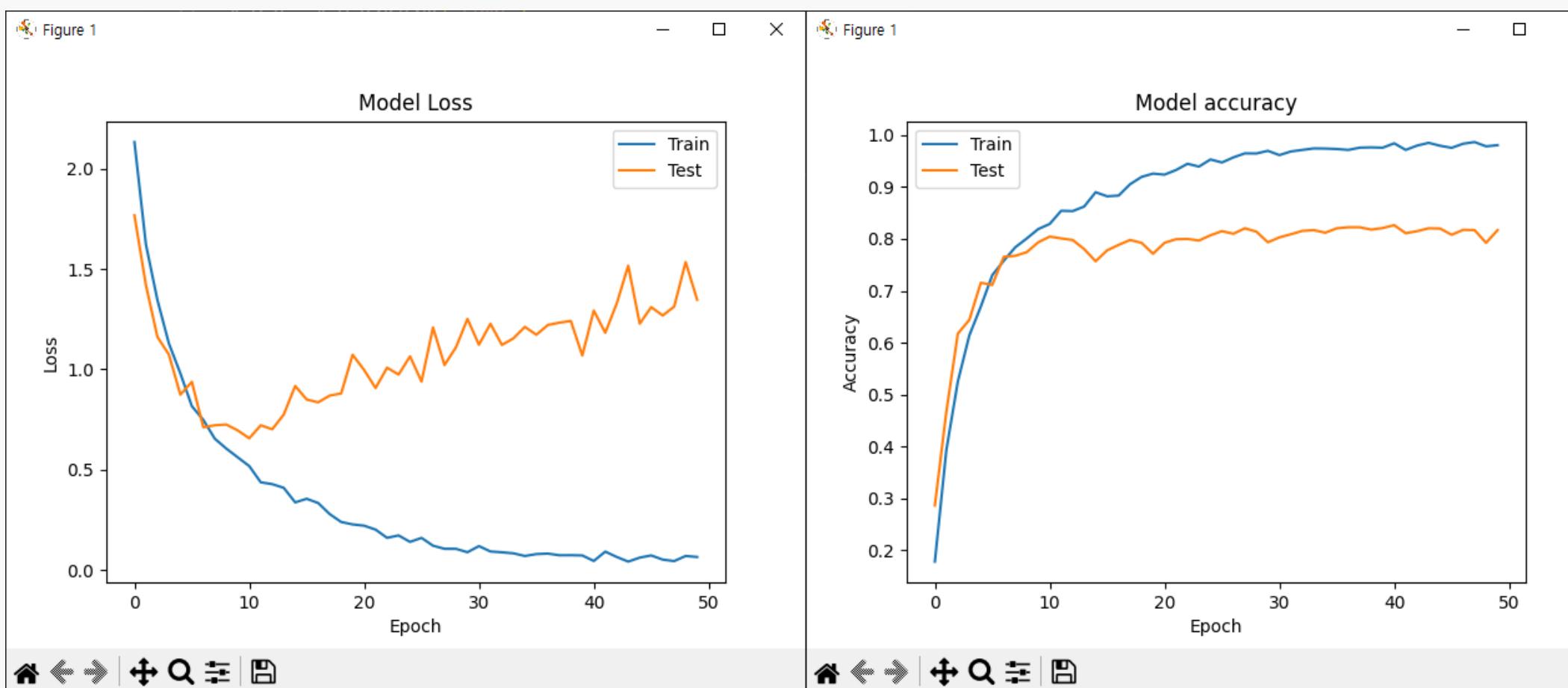
Banana



Lime



Blueberry



정확도: 66% → 81.69%

각자 다른 색상의 과일을 데이터셋으로 설정 한 후, Epoch 값을 50회로 상향 조정한 결과, 81.69%를 최종으로 얻을 수 있었음

```
40/40 [=====] - 141s 4s/step - loss: 0.0524 - accuracy: 0.9834 - val_loss: 1.2689 - val_accuracy: 0.8175
Epoch 48/50
40/40 [=====] - 141s 4s/step - loss: 0.0449 - accuracy: 0.9866 - val_loss: 1.3130 - val_accuracy: 0.8170
Epoch 49/50
40/40 [=====] - 142s 4s/step - loss: 0.0704 - accuracy: 0.9784 - val_loss: 1.5351 - val_accuracy: 0.7925
Epoch 50/50
40/40 [=====] - 141s 4s/step - loss: 0.0653 - accuracy: 0.9806 - val_loss: 1.3474 - val_accuracy: 0.8170
63/63 - 7s - loss: 1.3474 - accuracy: 0.8170 - 7s/epoch - 115ms/step

Loss: 1.347393274307251, Acc: 0.8169999718666077
```

## 데이터셋 구성의 중요성

머신러닝 모델을 설계하는 데에 있어 레이어의 구성 및 파라미터를 튜닝하는 과정 또한 매우 중요하지만, 초기 데이터셋을 정의하는 과정 역시 이 못지 않게 중요하다

## CNN model

컨볼루션 뉴럴 네트워크 모델의 정확도는 컨볼루션 레이어 층 개수에 따라 큰 폭으로 달라진다

풀링 레이어 한층에 컨볼루션 레이어 층을 3번 누적시킬 때 최적의 결과를 반환한다. (VGG16 모델 참고하였음)

컨볼루션 레이어 내에서 학습 진행률을 가장 크게 좌우하는 파라미터는 필터 개수이다. 해당 파라미터를 크게 설정할 경우, 학습 속도가 빨라지지만 다른 모델과 마찬가지로 overfitting이 발생할 가능성이 높아진다.

## Overfitting

Overfitting은 모델의 정확도를 높이기 위해선 피할 수 없는 문제이며, 이의 대표적인 해결방법으로 Dropout이 존재한다.

## Final Insight

비록 순수 넘파이만을 이용하여 정확도 70%를 넘기는 초기 목표는 달성하지 못하였으나, CNN 구현을 위해 여러 넘파이 연산들을 활용해보며, 머신러닝 모델이 학습하는 매커니즘을 자세히 탐구할 수 있었음. 또한 KERAS 라이브러리를 활용하여 최적의 CNN 모델 레이어 구성, Overfitting 문제 해결방안 등 여러 지식들을 얻을 수 있었음.

**감사합니다.**