[데이터 사이언스 개론]
기말 프로젝트

컴퓨터공학과 201811259 배수빈

목차

1. 과제 요구사항

1. 과제 요구사항

1. Train 및 test데이터로 구분하기
2. Train/test 데이터 구분시에, 데이터가 클래스별로 잘 섞이도록
3. tensorflow는 v2.3 버전 이상
4. 모델 파일 이름은 ‘model-학번 ' 즉 ‘model-201811259’
5. 모델의 input, output shape 을 지정한 shape으로 고정

1. 과제 요구사항

1.Train 및 test데이터로 구분하기

(각각 X, Y 따로)

하나의 라벨에 해당하는

이미지만 저장하는 리스트

4 : 1

각각의 X, Y에 대한

리스트를 train_test_split

함수를 이용해서

4:1로 구분

X와 Y에 대해서

Train과 Test를 각각 합쳐

X_train_all, Y_train_all,

X_test_all, Y_test_all

생성

1. 과제 요구사항

2. Train/test 데이터 구분시에, 데이터가 클래스별로 잘 섞이도록

(각각 X, Y 따로)

하나의 라벨에 해당하는

이미지만 저장하는 리스트

4 : 1

각각의 X, Y에 대한

리스트를 train_test_split

함수를 이용해서

4:1로 구분

X와 Y에 대해서

Train과 Test를 각각 합쳐

X_train_all, Y_train_all,

X_test_all, Y_test_all

생성

4개의 리스트를 각각

np_shuffle 함수

(직접 코딩) 를 이용해

Shuffle함

1. 과제 요구사항

3. tensorflow는 v2.3 버전 이상

```
import tensorflow as tf
print(tf.__version__)
```

```
2.3.1
```

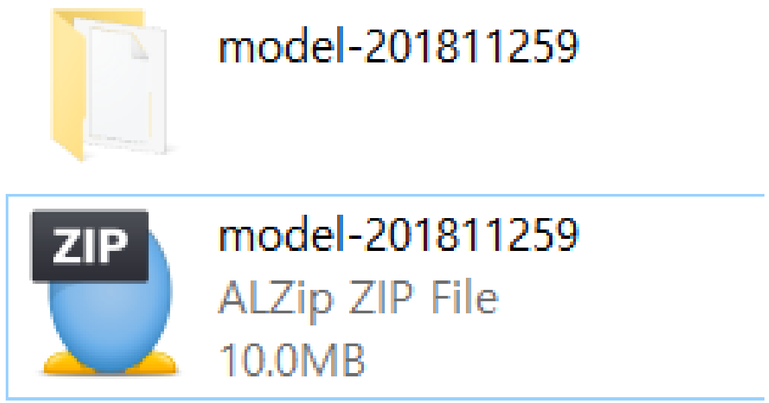
텐서플로우 버전 2.3으로

업데이트 완료하여

모델을 학습시켰음

1. 과제 요구사항

4. 모델 파일 이름은 ‘model-학번 ' 즉 ‘model-201811259’



학습을 완료하여 생성한 모델을

model-201811259

로 생성하여 zip파일로 압축하였음

1. 과제 요구사항

5. 모델의 input, output shape 을 지정한 shape으로 고정

```
model = Sequential([
        Input(shape=(300,300,3), name='input_layer'),

        ...

        Dense(3, activation='softmax', name='output_layer')
        ])
```

```
model = Sequential([
    Input(shape=(300,300,3), name='input_layer'),
    Conv2D(32, kernel_size=3, activation='relu', name='conv_layer1'),
    MaxPooling2D(pool_size=3),
    Conv2D(32, kernel_size=3, activation='relu', name='conv_layer2'),
    MaxPooling2D(pool_size=3),
    Conv2D(32, kernel_size=3, activation='softmax', name='conv_layer3'),
    Flatten(),
#     Dense(32, activation='softmax', name='Dense1'),
#     Dropout(0.2),
    Dense(32, activation='relu', name='Dense2'),
    Dense(3, activation='softmax', name='output_layer')
    ])
```

좌측이 요구사항,

우측이 직접 생성한 모델의 input, output shape

2. 모델 학습 시행착오

2. 모델 학습 시행착오

1. 시도 A

Convolutional Layer — 활성화 함수 : relu / padding : same / 커널사이즈 : 3

Max Pooling Layer

Convolutional Layer — 활성화 함수 : relu / padding : same / 커널사이즈 : 3

Max Pooling Layer

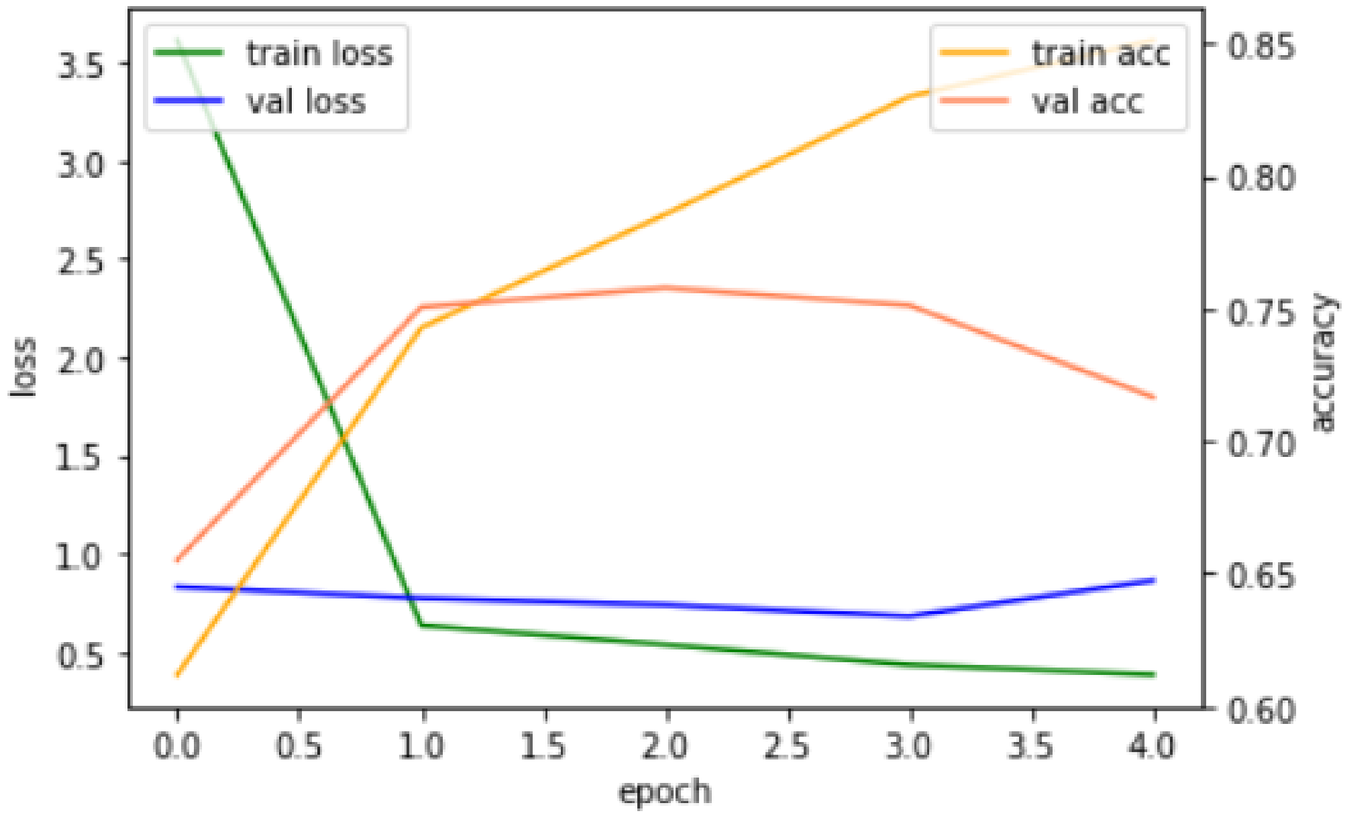
Flatten

Dense Layer (Output) — 활성화 함수 : softmax

Batch size : 50 / Epoch : 5

2. 모델 학습 시행착오

1. 시도 A



- Epoch 3부터 val_accuracy 감소
- Epoch 3부터 오버피팅 발생

=> accuracy를 높이기 위해

활성화 함수를 변경해보자 !

2. 모델 학습 시행착오

2. 시도 B : Convolutional Layer의 활성화함수를 elu로 변경

Convolutional Layer — 활성화 함수 : elu / padding : same / 커널사이즈 : 3

Max Pooling Layer

Convolutional Layer — 활성화 함수 : elu / padding : same / 커널사이즈 : 3

Max Pooling Layer

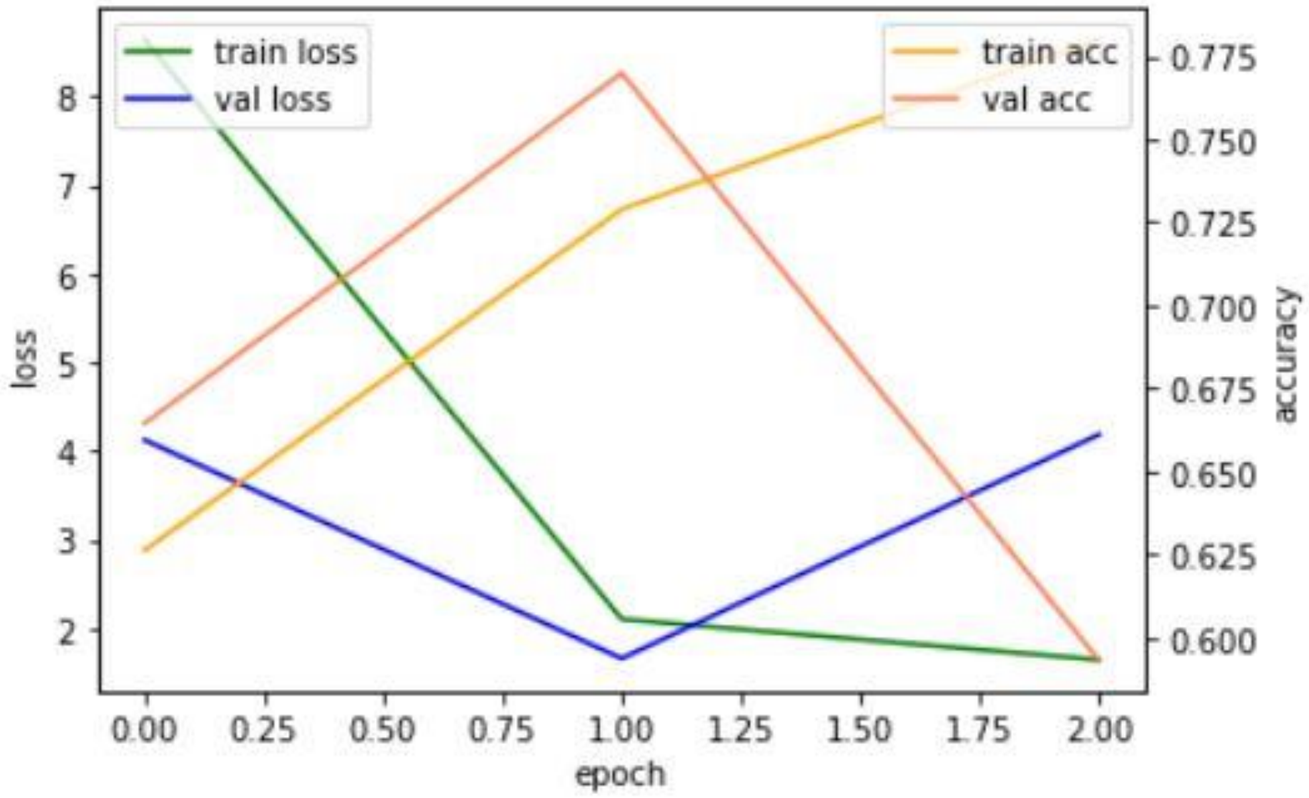
Flatten

Dense Layer (Output) — 활성화 함수 : softmax

Batch size : 50 / Epoch : 3

2. 모델 학습 시행착오

2. 시도 B : Convolutional Layer의 활성화함수를 elu로 변경



최대 val_accuracy = 0.77

- Epoch 3부터 val_accuracy 감소
- Epoch 3부터 val_loss 증가
- Epoch 3부터 오버피팅 발생

=> elu페기, 레이어 변경,

batch size 감소

2. 모델 학습 시행착오

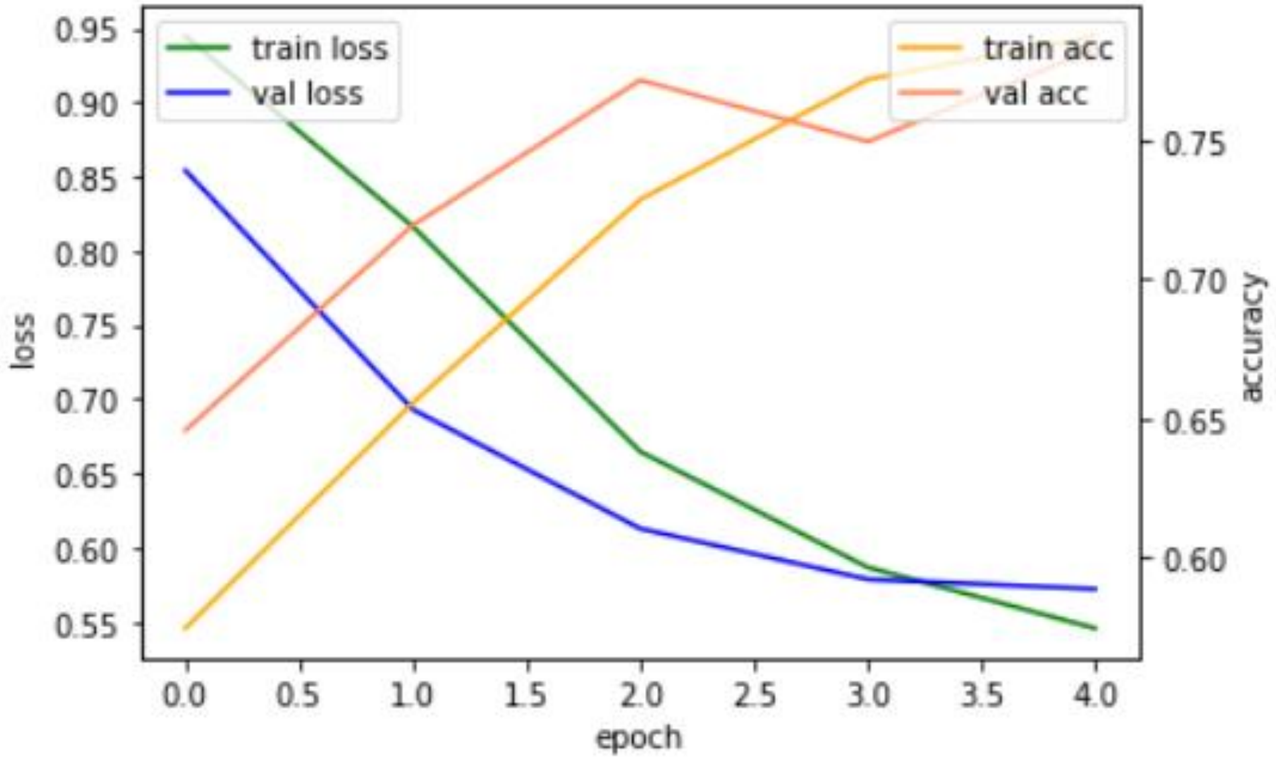
3. 시도 C

Convolutional Layer	활성화 함수 : relu / padding : same / 커널사이즈 : 3
Max Pooling Layer	
Convolutional Layer	활성화 함수 : relu / padding : same / 커널사이즈 : 3
Max Pooling Layer	
Convolutional Layer	활성화 함수 : softmax / padding : same / 커널사이즈 : 3
Flatten	
Dense	활성화 함수 : softmax
Dense (Output)	활성화 함수 : softmax

Batch size : 32 / Epoch : 5

2. 모델 학습 시행착오

3. 시도 C



최대 val_accuracy = 0.7823

- Epoch 4에서 val_accuracy 감소
- Epoch 5에서 val_accuracy 재 증가
- val_loss 꾸준히 감소

=> Epoch3에서의 val_accuracy 감소가 우연일 수 있다고 판단

=> Epoch 수를 증가시켜서 테스트

2. 모델 학습 시행착오

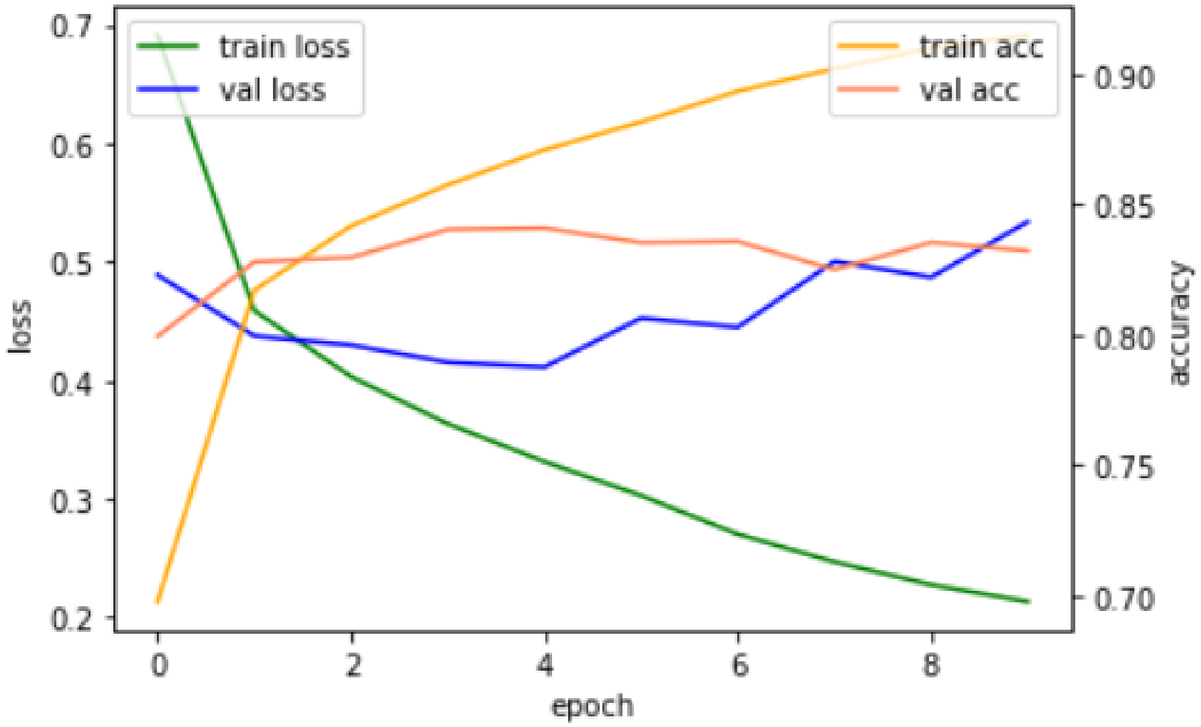
4. 시도 D : epoch를 10으로 증가 + 활성화함수 변경

Convolutional Layer	활성화 함수 : relu / padding : same / 커널사이즈 : 3
Max Pooling Layer	
Convolutional Layer	활성화 함수 : relu / padding : same / 커널사이즈 : 3
Max Pooling Layer	
Convolutional Layer	활성화 함수 : softmax / padding : same / 커널사이즈 : 3
Flatten	
Dense	활성화 함수 : relu
Dense (Output)	활성화 함수 : softmax

Batch size : 32 / Epoch : 10

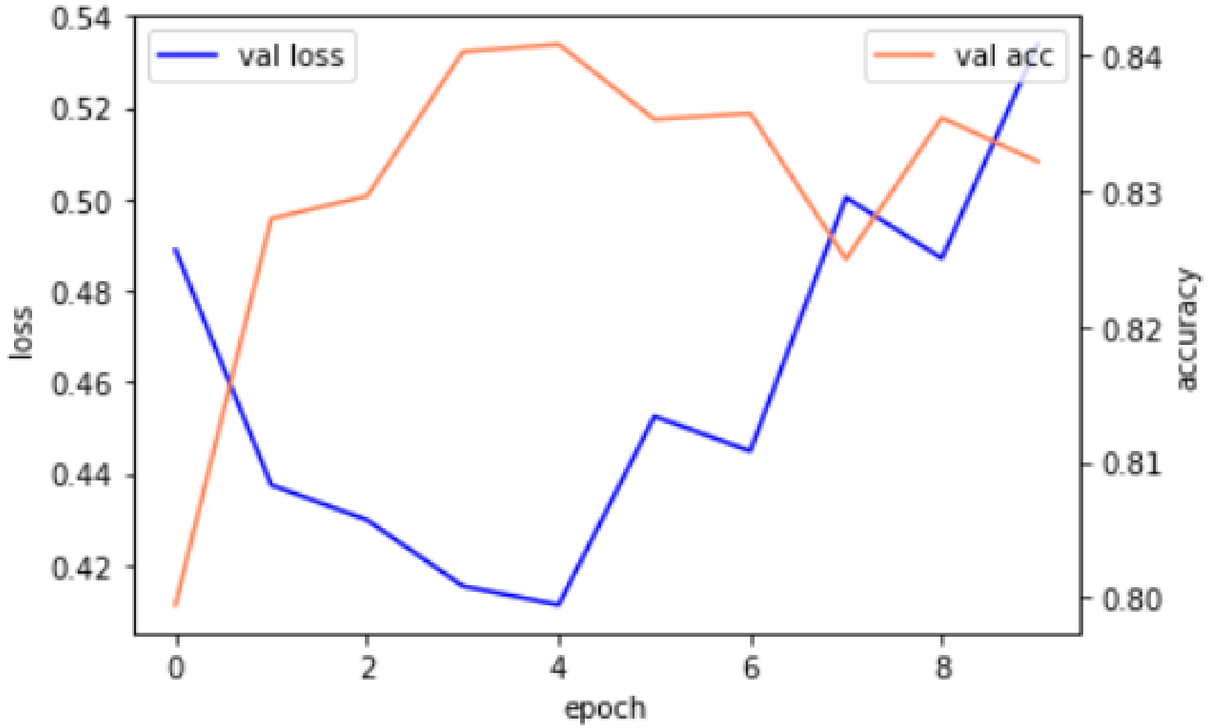
2. 모델 학습 시행착오

4. 시도 D : epoch를 10으로 증가 + 활성화함수 변경



2. 모델 학습 시행착오

4. 시도 D : epoch를 10으로 증가 + 활성화함수 변경



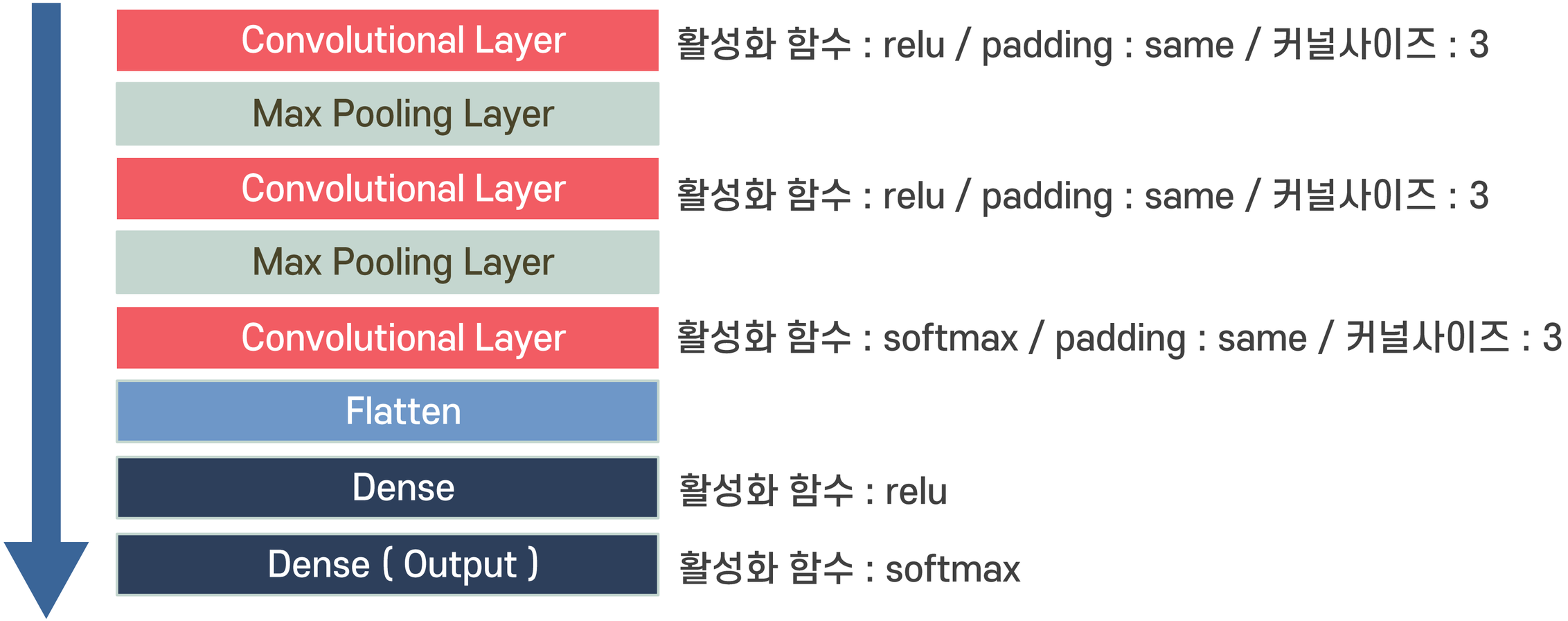
최대 val_accuracy = 0.8409

- Epoch 6부터 val_accuracy 감소
- Epoch 6부터 val_loss 증가
- Epoch 6부터 오버피팅 발생

=> Epoch5 까지만 재 실행

2. 모델 학습 시행착오 - 최종모델

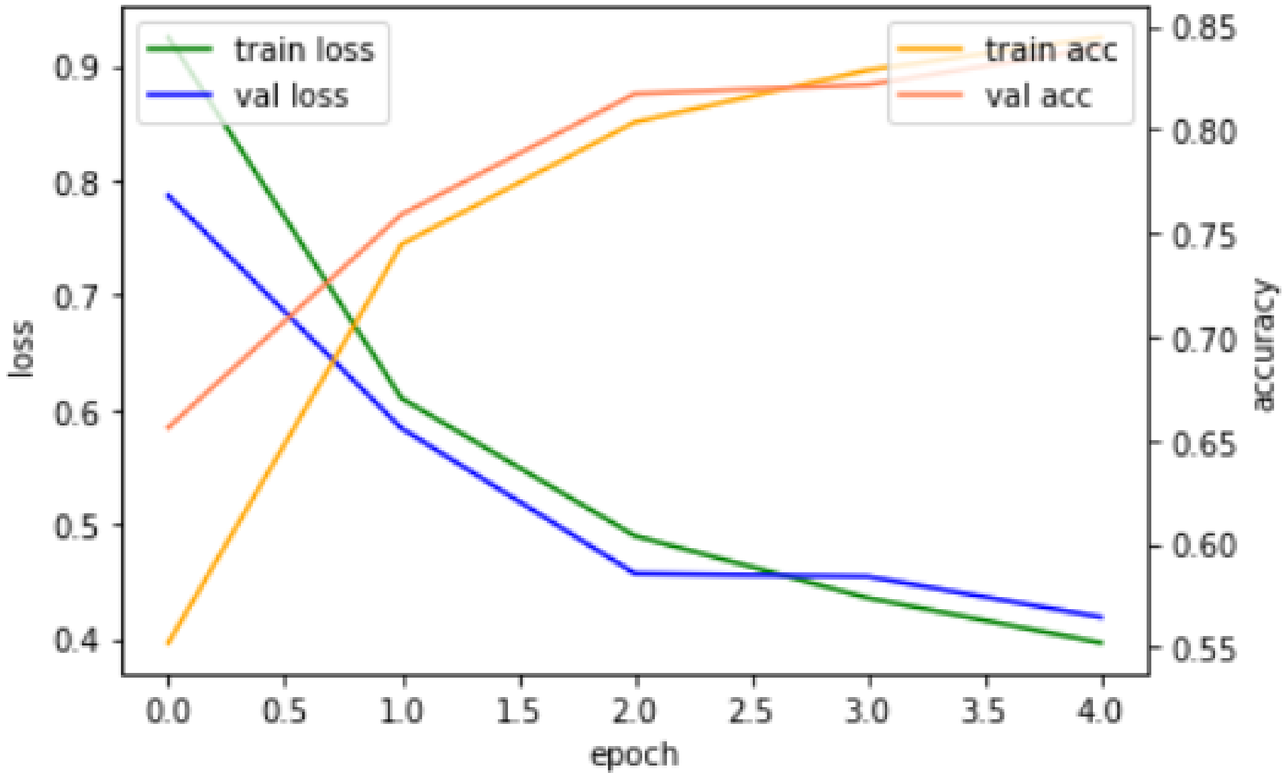
5. 시도 E : epoch를 5까지



Batch size : 32 / Epoch : 5

2. 모델 학습 시행착오 - 최종모델

5. 시도 E : epoch를 5까지



최대 val_accuracy = 0.8397

- 오버피팅이 발생하지 않고 Test와 Train의 정확도가 비례하게 증가

3. 최종 모델 성능 분석

3. 최종 모델 성능 분석

```
Epoch 5/5
1125/1125 [==============================] - 2655s 2s/step - loss: 0.3966 - accuracy:
0.8447 - val_loss: 0.4189 - val_accuracy: 0.8397
```

학습하지 않은 데이터에 대한

Val_loss = 0.4189

Val_accuracy = 0.8397

3. 최종 모델 성능 분석

```
from sklearn.metrics import classification_report

Y_pred = model.predict(X_test, batch_size=32, verbose=1)
Y_pred_classNum = np.argmax(Y_pred, axis = 1)
Y_test_classNum = np.argmax(Y_test, axis = 1)

print(classification_report(Y_test_classNum, y_pred_bool))
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93      4000
           1       0.80      0.83      0.81      3000
           2       0.86      0.71      0.78      2000

    accuracy                           0.84      9000
   macro avg       0.86      0.83      0.84      9000
weighted avg       0.86      0.86      0.86      9000
```

3. 최종 모델 성능 분석

* precision

$$\frac{TP}{TP \times FP}$$

	음식	실내	실외
precision	0.91	0.80	0.86

	예측			
실제 클래스		음식	실내	실외
	음식	A		
	실내	B		
	실외	C		

[음식]이라고 판단한 것들 중 정말로 [음식]일 확률

(A / A+B+C)

: 음식 > 실외 > 실내

3. 최종 모델 성능 분석

* Recall

$$\frac{TP}{TP \times FN}$$

	음식	실내	실외
recall	0.96	0.83	0.71

	예측			
실제 클래스		음식	실내	실외
	음식	A	B	C
	실내			
	실외			

실제[음식]인 데이터에 대해서 [음식]이라고 판단할 확률

(A / A+B+C)

: 음식 > 실내 > 실외

3. 최종 모델 성능 분석

* f1-score

$$2 \times \frac{Precision * Recall}{Precision + Recall}$$

	음식	실내	실외
f1-score	0.93	0.82	0.78

Precision과 Recall의 조화평균 (두 수치 적절하게 고려)

: 음식 > 실내 > 실외

3. 최종 모델 성능 분석

* f1-score

$$2 \times \frac{Precision * Recall}{Precision + Recall}$$

	음식	실내	실외
f1-score	0.93	0.82	0.78

✓ 대체적으로 음식 > 실내 > 실외 로 맞게 예측함

✓ Precision, Recall, F1_score 모두 음식이 큰 차이로 가장 큰 수치를 보임

=> 음식 데이터의 양이 타 클래스의 데이터보다 많았기 때문에
학습이 잘 되었고, 그에 따라 좀더 정확히 분류함을 알 수 있음

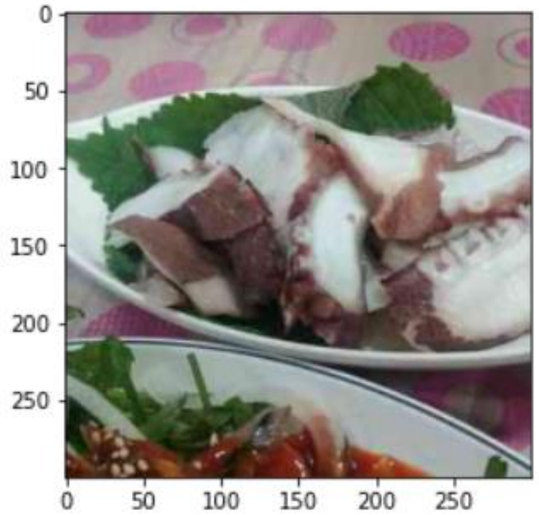
4. Prediction 시연

4. Prediction 시연

* 맞는 예측

Actual : 음식
Prediction : 음식

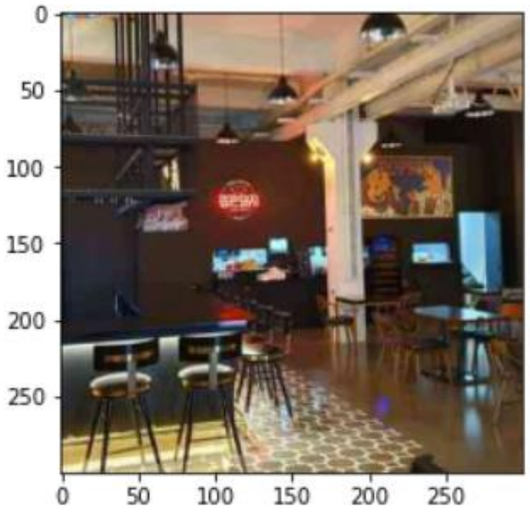
== Predictions ==
음식 확률 : 0.9866889
실내 확률 : 0.012887565
실외 확률 : 0.0004236092



< 음식 >

Actual : 실내
Prediction : 실내

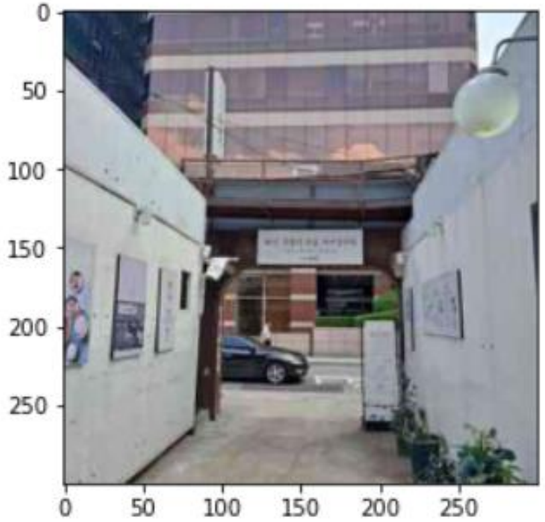
== Predictions ==
음식 확률 : 0.019992538
실내 확률 : 0.87856233
실외 확률 : 0.10144507



< 실내 >

Actual : 실외
Prediction : 실외

== Predictions ==
음식 확률 : 0.0092320945
실내 확률 : 0.3860613
실외 확률 : 0.6047066



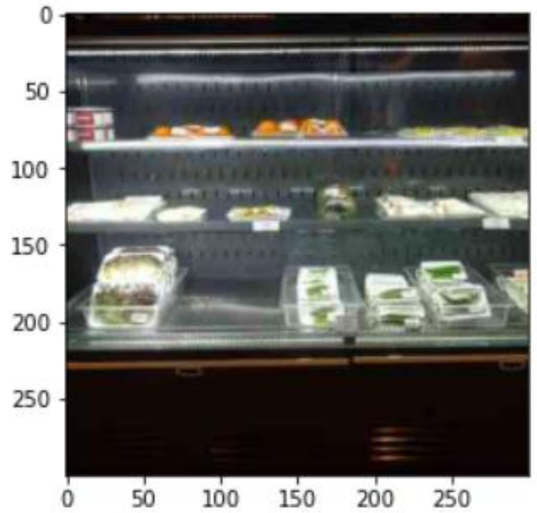
< 실외 >

4. Prediction 시연

* 틀린 예측

Actual : 음식
Prediction : 실내

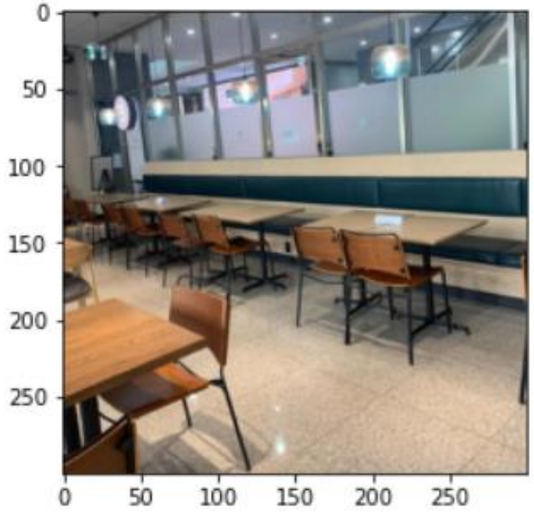
== Predictions ==
음식 확률 : 0.063880265
실내 확률 : 0.7300443
실외 확률 : 0.20607536



< 음식 → 실내 >

Actual : 실내
Prediction : 실외

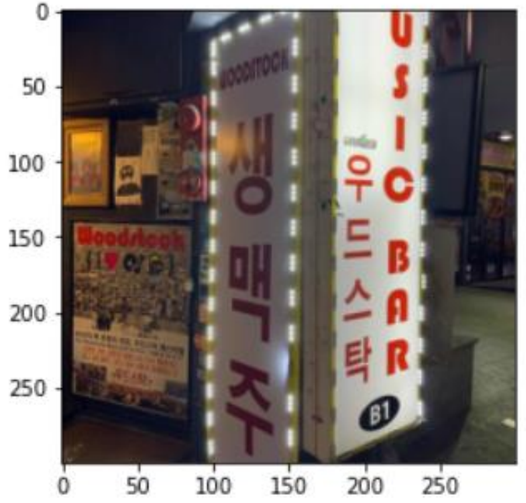
== Predictions ==
음식 확률 : 0.03162623
실내 확률 : 0.34537774
실외 확률 : 0.62299603



< 실내 → 실외 >

Actual : 실외
Prediction : 실내

== Predictions ==
음식 확률 : 0.009607278
실내 확률 : 0.78620976
실외 확률 : 0.20418294



< 실외 → 실내 >

감사합니다 ☺