

THE DEVIL WEARS RED RAG



1. 주제

- 주제 선정 이유

2. 서비스 제작 과정

- 팀원 역할
- 개발환경

3. 프로토타이핑

- 데이터 수집
- 이미지 전처리

4. 모델링 (개인)

- 모델 각 평가지표
- 모델링 결과

5. 시연

TOPIC

주제 및 주제 선정 이유





FASHION EVALUATION SERVICE

...FOR WOMEN

여성용 패션 진단 서비스

PROCESS

서비스 제작 과정





TEAM

Sowon Park

패션 스타일별 시대 예측 모델 구현 및 안내 PPT 제작

Seoyeong Na

패션 스타일별 소득수준 예측 모델 구현, 아이디어 기획

Taehyoung Lee

패션 스타일별 연령대 예측 모델 구현, HTML 제작, 메인 페이지 디자인

Soojin Lee

패션 스타일별 직업군 예측 모델 구현, PPT 제작

Seyoon Jung

패션 스타일 예측 모델 구현

기능설계 및 업무 분담

대분류	중분류	소분류	상세 내역	우선순위	담당자
데이터 수집	데이터 수집	활용 데이터 선정	AI허브 '연도별 패션 선호도 파악 및 추천 데이터' 선정	1	전원
		데이터 사용 범위 설정	가용 용량 문제 해결을 위해 test 데이터만을 가져와 다시 train, test 자료로 활용(총 22,496개 이미지 다운로드)	1	전원
데이터 전처리	이미지 사이즈 변경	이미지 파일 형식 변경	opencv를 활용하여 .jpg->.npy 변환	2	전원
		1차 데이터 축소	3000*4000px color jpg 파일을 (300,400,3) npy 파일로 변환	2	전원
		2차 데이터 축소	(300*400*3) npy 파일을 (150*200*3) npy 파일로 변환	2	전원
		3차 데이터 축소	(150*200*3) npy 파일을 (75*100*3) npy 파일로 변환	2	전원
	데이터 형식 변경	데이터 타입 지정	이미지 데이터를 float16 형식으로 고정 활용	2	전원
기능 설계	분류 타겟 설정	분류모형 구현을 위한 타겟 설정	1. 시대 2. 연령대 3. 스타일 4. 직업군 5. 소득수준	3	전원
	예측 모델 구축	분류 모형 구축	CNN 모델 구현 1. 패션 스타일별 시대 예측 모델 구현	3	박소원
			CNN 모델 구현 2. 패션 스타일별 연령대 예측 모델 구현	3	이태형
			CNN 모델 구현 3. 패션 스타일 예측 모델 구현	3	정세윤
			CNN 모델 구현 4. 패션 스타일별 직업군 예측 모델 구현	3	이수진
			CNN 모델 구현 5. 패션 스타일별 소득수준 예측 모델 구현	3	나서영
서비스 구현	WEB 구현	페이지 연동	Flask를 통한 메인-결과 페이지 연결 구축	4	이태형
		페이지 디자인, 구현	사진을 입력받는 메인 페이지 디자인	5	이태형
	서비스 안내서 제작	서비스 안내서 제작	서비스 제안 및 안내 PT를 위한 visual 기반 디자인	5	박소원, 이수진
			서비스 제안 및 안내 PT를 위한 visual 제작	5	전원

SYSTEM

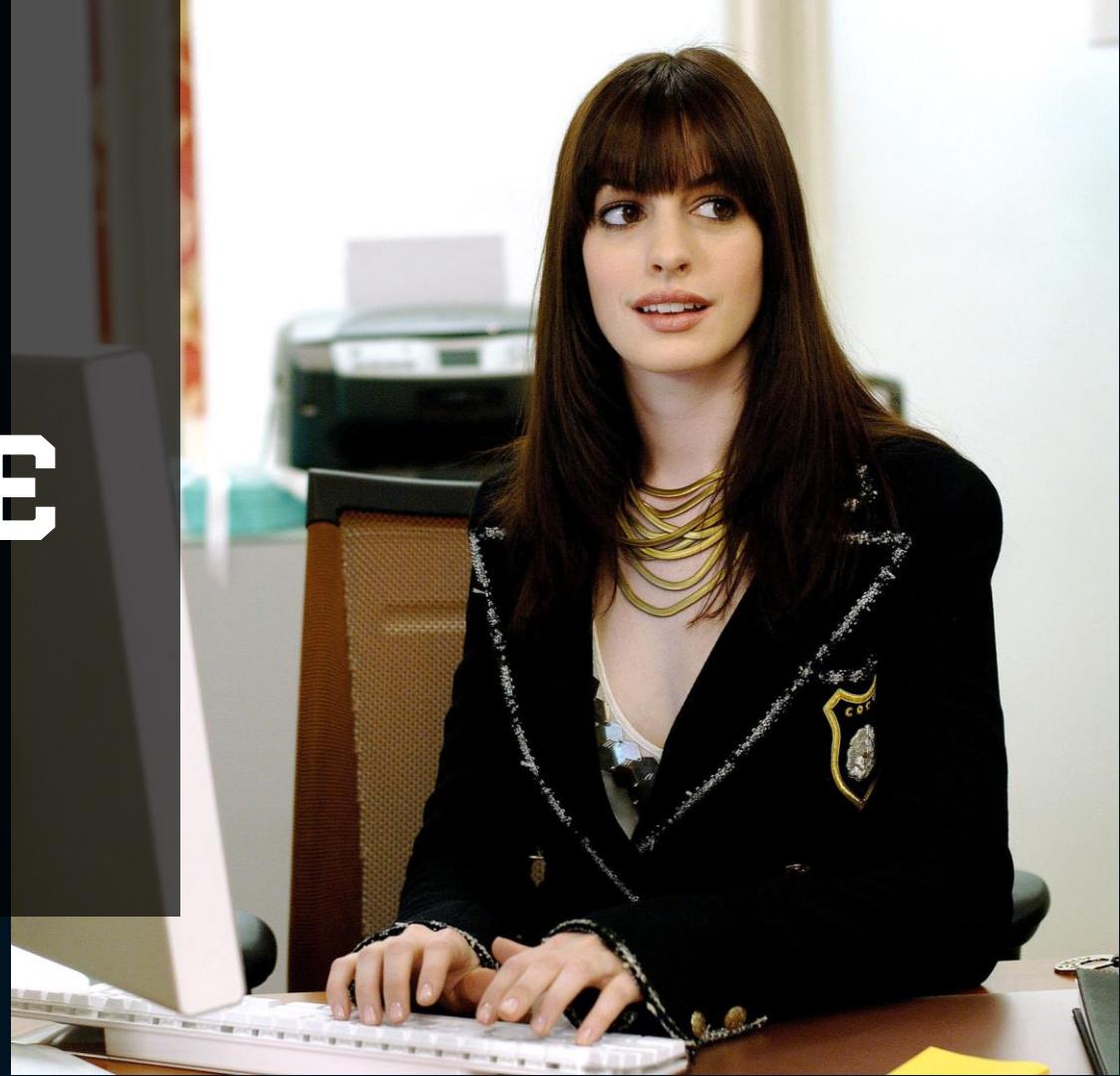
WINDOWS 11

LANGUAGE

PYTHON, HTML, CSS

LIBRARY

KERAS, NUMPY, CV2, JSON





PRE-

데이터 수집 및 전처리



DATA

데이터 출처: AI 허브

데이터 샘플 수: 26,000개의 컬러 이미지

데이터 라벨: 연도, 스타일, 직업, 월 수입, 나이

각 데이터 사이즈: 3,000px X 4,000px

DATA PRE-PROCESS

RESIZE

(4000,3000) -> (400,300) -> (100, 75)

NORMALIZATION

이미지 정규화

ONE-HOT-ENCODING

다중 분류 원-핫-인코딩





정세윤

모델 유형: 지도학습 - CNN 예측 모델

타겟 : 패션 스타일



```
data_1950_ = np.load('../data_1950.npz')
data_1950_ = data_1950_[‘images’].astype(np.float16)

data_1960_ = np.load('../data_1960.npz')
data_1960_ = data_1960_[‘images’].astype(np.float16)

data_1970_ = np.load('../data_1970.npz')
data_1970_ = data_1970_[‘images’].astype(np.float16)

data_1980_ = np.load('../data_1980.npz')
data_1980_ = data_1980_[‘images’].astype(np.float16)

data_2010_ = np.load('../data_2010.npz')
data_2010_ = data_2010_[‘images’].astype(np.float16)

data_2019_ = np.load('../data_2019.npz')
data_2019_ = data_2019_[‘images’].astype(np.float16)
```

```
combined_data = np.load('all_32.npy')
```

≡ all_32.npy
≡ all_resized.npy
≡ all1.csv
≡ data_1950.npz
≡ data_1960.npz
≡ data_1970.npz
≡ data_1980.npz
≡ data_2010.npz
≡ data_2019.npz
≡ test_x.npy
≡ test_y.npy
≡ train_x_2.npy
≡ train_x_resized2.npy
≡ train_x.npy
≡ train_y_2.npy
≡ train_y.npy
≡ val_x.npy
≡ val_y.npy

MemoryError: Unable to allocate 11.3 GiB for an array with shape (16872, 400, 300, 3) and data type float16



```
combined_data = np.load('all_resized.npy')

combined_data.shape

(22496, 100, 75, 3)
```

```
np.save('train_x.npy',train_x)
np.save('test_x.npy',test_x)
np.save('train_y.npy',train_y)
np.save('test_y.npy',test_y)
```

```
np.save('train_x_2.npy',train_x_2)
np.save('val_x.npy',val_x)
np.save('train_y_2.npy',train_y_2)
np.save('val_y.npy',val_y)
```

- ≡ all_32.npy
- ≡ all_resized.npy
- ≡ all1.csv
- ≡ data_1950.npz
- ≡ data_1960.npz
- ≡ data_1970.npz
- ≡ data_1980.npz
- ≡ data_2010.npz
- ≡ data_2019.npz
- ≡ test_x.npy
- ≡ test_y.npy
- ≡ train_x_2.npy
- ≡ train_x_resized2.npy
- ≡ train_x.npy
- ≡ train_y_2.npy
- ≡ train_y.npy
- ≡ val_x.npy
- ≡ val_y.npy



conv2d_4 (Conv2D)	(None, 100, 75, 16)	448
activation_8 (Activation)	(None, 100, 75, 16)	0
max_pooling2d_4 (MaxPooling2D)	(None, 50, 37, 16)	0
conv2d_5 (Conv2D)	(None, 50, 37, 16)	2320
activation_9 (Activation)	(None, 50, 37, 16)	0
max_pooling2d_5 (MaxPooling2D)	(None, 25, 18, 16)	0
flatten_2 (Flatten)	(None, 7200)	0
dense_6 (Dense)	(None, 16)	115216
activation_10 (Activation)	(None, 16)	0
dense_7 (Dense)	(None, 16)	272
activation_11 (Activation)	(None, 16)	0
dense_8 (Dense)	(None, 23)	391
=====		
Total params: 118647 (463.46 KB)		
Trainable params: 118647 (463.46 KB)		

배치 정규화는 생략

클래스의 수가 23개인
다중 클래스 분류 문제를 다루기 위해
출력 뉴런 갯수는 23개로 설정

```
----- Epoch 6/500
422/422 [=====] - 35s 83ms/step - loss: 2.5167 - accuracy: 0.2312 - val_loss: 2.6504 - val_accuracy: 0.2055
Epoch 7/500
422/422 [=====] - 38s 90ms/step - loss: 2.4231 - accuracy: 0.2531 - val_loss: 2.5987 - val_accuracy: 0.2270
Epoch 8/500
422/422 [=====] - 29s 68ms/step - loss: 2.3319 - accuracy: 0.2770 - val_loss: 2.5588 - val_accuracy: 0.2435
Epoch 9/500
422/422 [=====] - 29s 70ms/step - loss: 2.2435 - accuracy: 0.3033 - val_loss: 2.5250 - val_accuracy: 0.2642
Epoch 10/500
422/422 [=====] - 39s 93ms/step - loss: 2.1586 - accuracy: 0.3330 - val_loss: 2.4642 - val_accuracy: 0.2840
Epoch 11/500
422/422 [=====] - 30s 71ms/step - loss: 2.0840 - accuracy: 0.3519 - val_loss: 2.4712 - val_accuracy: 0.3033
Epoch 12/500
422/422 [=====] - 27s 65ms 176/176 [=====] - 2s 13ms/step
Epoch 13/500
422/422 [=====] - 28s 67ms
[[1.4874345e-01 6.2042248e-02 4.0196557e-02 ... 7.6154345e-03
 3.1066483e-03 8.3692726e-03]
[8.8587839e-08 5.2802437e-03 1.2433491e-01 ... 1.8696494e-01
 1.6705485e-02 1.0990493e-05]
[2.8619360e-02 7.1489833e-02 7.8192107e-02 ... 1.4504197e-02
 4.8699188e-03 1.1859784e-02]
...
[5.7731491e-02 4.7709532e-03 3.9437260e-02 ... 2.8824931e-04
 1.4620998e-05 2.8260064e-01]
[4.6753259e-03 1.2780898e-04 7.6039257e-03 ... 4.5101051e-06
 4.5816439e-08 7.8816718e-01]
[4.0037581e-03 9.3169354e-02 6.5966696e-02 ... 8.8121966e-02
 1.0468859e-01 3.7851892e-04]]
```



이수진

모델 유형: 지도학습 - CNN 예측 모델

타겟 : 패션 스타일별 작업

직업 분류 모델링 학습 시작 시간

23:30

램 메모리 부족 상태

코랩 연결 끊김 상태

램 문제 해결:
Image Data Generator

메모리 효율성

이미지를 실시간으로 처리하고
학습 도중에 배치로 공급

실제로 에포크 1000 설정 후 870까지
무리 없이 코랩에서 학습 진행
(코랩 자동 연결 끊김 현상이 원인)

```
[ ] data_all = np.load('/content/drive/MyDrive/DL/all_resized.npy')

[ ] # 전체 데이터를 훈련 데이터와 테스트 데이터로 분리
train_x, test_x, train_y, test_y = train_test_split(data_all, label, stratify=label, test_size=0.2 ,random_state=7)

[ ] # 훈련 데이터를 다시 훈련 데이터와 검증 데이터로 분리
train_x_2, val_x, train_y_2, val_y = train_test_split(train_x, train_y, test_size=0.25, random_state=7)
```

```
..  
train_x = np.load('/content/drive/MyDrive/dl2/train_x.npy')  
train_y = np.load('/content/drive/MyDrive/dl2/train_y.npy')  
val_x = np.load('/content/drive/MyDrive/dl2/val_x.npy')  
val_y = np.load('/content/drive/MyDrive/dl2/val_y.npy')  
..
```

```
[ ] from keras.preprocessing.image import ImageDataGenerator  
  
[ ] train_datagen = ImageDataGenerator()  
  
[ ] augmented_train_data = train_datagen.flow(train_x, train_y)  
  
[ ] validation_data = train_datagen.flow(val_x, val_y)
```

Image Data Generator 객체의 메서드:
flow()

Flow() 메서드

실제로 이미지와 레이블 배열을 받아서
배치 사이즈만큼 데이터를 생성해주는
iterator(반복자)를 만듦

```
from keras.layers import LeakyReLU, BatchNormalization
from keras.optimizers import Adam
model_path = '/content/drive/MyDrive/dl2/model/'
m_File = model_path + 'cnn_fashion_model_2.h5'
mcCB = ModelCheckpoint(m_File,
                       save_best_only=True, monitor='val_loss')
esCB = EarlyStopping(monitor='val_loss', patience=10)

model = Sequential(name='Fashion_Job_Model')

model.add(Conv2D(32, 3, input_shape=(100,75, 3)))
model.add(LeakyReLU(alpha=0.01))
model.add(MaxPool2D())
model.add(Conv2D(32, 3))
model.add(LeakyReLU(alpha=0.01))
model.add(MaxPool2D())

model.add(Flatten())

model.add(Dense(units=256))
model.add(Dropout(rate=0.4))
model.add(Dense(units=6, activation='softmax'))
```

```
opt = Adam(learning_rate=1e-5)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(augmented_train_data,
                     epochs=1000,
                     validation_data=validation_data,
                     callbacks=[mcCB, esCB])
```

```
[34] test_x = np.load('/content/drive/MyDrive/dl2/test_x.npy')
    test_y = np.load('/content/drive/MyDrive/dl2/test_y.npy')

[35] from keras.models import load_model

[36] model_path = '/content/drive/MyDrive/dl2/model/cnn_fashion_model_2.h5'

[37] model = load_model(model_path)

[38] loss, accuracy = model.evaluate(test_x, test_y)
```

141/141 [=====] - 12s 85ms/step - loss: 2.9476 - accuracy: 0.2316



박소원

모델 유형: 지도학습 - CNN 예측 모델

타겟 : 패션 스타일별 시대 분류

DATA : AI허브 [연도별 패션 선호도 파악 및 추천 데이터]

TARGET : 시대 ⇒ 1950 ~ 2020년 까지 8개의 범주로 분류

**IMAGE
SIZE** : 400 * 300 px

MODEL : CNN (합성곱 신경망)

FASHION 으로 시대를 예측하자

FASHION - YEAR

Project_sowon

- ✓ 각 연도별 data 로딩 및 Under Sampling
(2000장 * 8 = 총 16000장)

Project_sowon_2

- ✓ target data 생성
- ✓ Train / test data 분리
- ✓ One-Hot Encoding

Project_sowon_3 ~ 7

- ✓ Modeling

Predict

- ✓ 평가 및 예측

■	📁 data
■	📁 models
■	📁 test_img
■	📝 predict-Copy1.ipynb
■	⭕ predict.ipynb
■	📝 project_sowon.ipynb
■	📝 project_sowon_2.ipynb
■	📝 project_sowon_3.ipynb
■	📝 project_sowon_4.ipynb
■	📝 project_sowon_5.ipynb
■	📝 project_sowon_6.ipynb
■	📝 project_sowon_7.ipynb
■	📅 data.npy
■	📅 test_X.npy
■	📅 test_X.zip
■	📅 test_y.npy
■	📅 train_X.npy
■	📅 train_X.zip
■	📅 train_y.npy

FASHION - YEAR

Project_sowon

- ✓ 각 연도별 data 로딩 및 Under Sampling
(2000장 * 8 = 총 16000장)

```
1 label_path='./data/*csv'  
2 label_list=glob.glob(label_path)  
3 label_list  
  
['./data##data_1950.csv',  
 './data##data_1960.csv',  
 './data##data_1970.csv',  
 './data##data_1980.csv',  
 './data##data_1990.csv',  
 './data##data_2000.csv',  
 './data##data_2010.csv',  
 './data##data_2019.csv']
```

언더 샘플링을 위한 값 갯수 확인

```
1 for file in label_list:  
2     label=pd.read_csv(file)  
3     print(f"{file} => {label['era'].value_counts().iloc[0]}")
```

./data##data_1950.csv => 3358
./data##data_1960.csv => 3030
./data##data_1970.csv => 2910
./data##data_1980.csv => 3136
./data##data_1990.csv => 2210
./data##data_2000.csv => 2566
./data##data_2010.csv => 2848
./data##data_2019.csv => 2438

지속적인 **MemoryError** 발생

- 시대 예측 시 옷의 색상 / 패턴이 중요
- 다운 샘플링 진행

FASHION YEAR

Project_sowon_2

- ✓ target data 생성
- ✓ Train / test data 분리
- ✓ One-Hot Encoding

1.1 target data 만들기

```
1 target1=np.full((2000,), '1950')
2 target1
array(['1950', '1950', '1950', ..., '1950', '1950', '1950'], dtype='<...')

1 target2=np.full((2000,), '1960')
2 target3=np.full((2000,), '1970')
3 target4=np.full((2000,), '1980')
4 target5=np.full((2000,), '1990')
5 target6=np.full((2000,), '2000')
6 target7=np.full((2000,), '2010')
7 target8=np.full((2000,), '2019')

1 target=np.concatenate((target1,target2,target3,target4,target5,target6,target7,target8))
```

1.3 데이터 나누기

```
1 from sklearn.model_selection import train_test_split
2 train_X, test_X, train_y, test_y = train_test_split(data, target,
3                                                     stratify=target,
4                                                     random_state=7)
5
6 train_X.shape, test_X.shape, train_y.shape, test_y.shape
```

((12000, 400, 300, 3), (4000, 400, 300, 3), (12000,), (4000,))

```
1 np.save('train_X',train_X)
2 np.save('test_X',test_X)
3 np.save('train_y',train_y)
4 np.save('test_y',test_y)
```

또 지속적인 **MemoryError** 발생
→ **train / test data 저장**

FASHION - YEAR

Project_sowon_3

✓ Modeling

초기 확인을 위해
기본 코드 + callback 지정 후
모델링 진행

```
1 # 모델 설계 - CNN
2 from keras.models import Sequential
3 from keras.layers import Dense, Conv2D, Dropout, MaxPool2D, Flatten
4
5 model=Sequential()
6 model.add(Conv2D(128, 3, input_shape=(400,300,3), activation='relu'))
7 model.add(MaxPool2D())
8 model.add(Conv2D(64, 3, activation='relu'))
9 model.add(MaxPool2D())
10 model.add(Conv2D(32, 3, activation='relu'))
11 model.add(MaxPool2D())
12 model.add(Conv2D(16, 3, activation='relu'))
13 model.add(MaxPool2D())
14
15 model.add(Flatten())
16
17 model.add(Dense(100, activation='relu'))
18 model.add(Dense(8, activation='softmax'))
19
20 model.summary()
```

FASHION - YEAR

Callback 지정

```
1 # 객체 생성
2 model_file=model_path+'fashion_year_final.HDF5'
3
4 # callback
5 from keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler
6
7 # 가장 좋은 모델만 저장
8 mc_CB=ModelCheckpoint(monitor='val_accuracy', filepath=model_file, save_best_only=True)
9
10 # 조기종료 설정
11 es_CB=EarlyStopping(patience=5)
12
13 # epoch 마다 학습률 줄이기
14 def scheduler(epoch, lr):
15     if epoch>5:
16         return 0.0001
17     else:
18         return lr
19 lr_CB=LearningRateScheduler(scheduler)
```

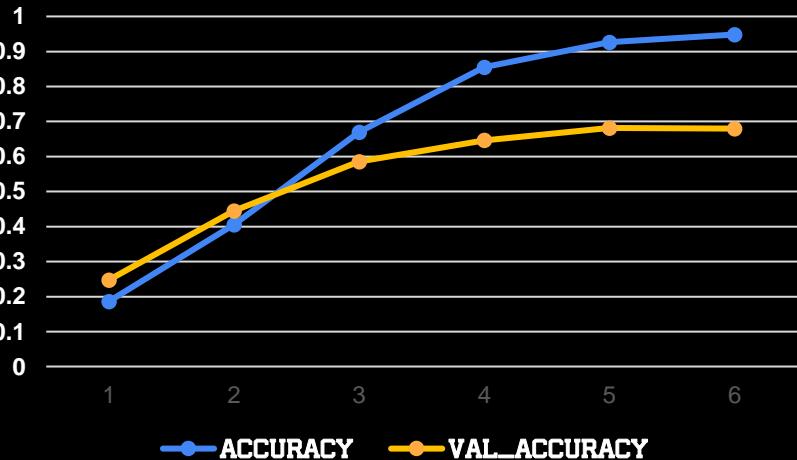
FASHION YEAR

LOSS / ACCURACY 확인

LOSS - VAL_LOSS



ACCURACY - VAL_ACCURACY



Train loss: 0.5661 - accuracy: 0.9038
Test loss: 2.0993 - accuracy: 0.6607

⇒ 과대적합

FASHION - YEAR

Project_sowon_4

- ✓ 과대적합 해결을 위한 Modeling

은닉층 추가

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, MaxPool2D, Flatten

model=Sequential()
model.add(Conv2D(128, 3, input_shape=(400,300,3), activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(64, 3, activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(32, 3, activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(16, 3, activation='relu'))
model.add(MaxPool2D())

model.add(Flatten())

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2)) # Dense 사이에 위치
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2)) # Dense 사이에 위치
model.add(Dense(8, activation='softmax'))

model.summary()
```

FASHION - YEAR

Project_sowon_4

✓ 과대적합 해결을 위한 Modeling

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten
model=Sequential()
model.add(Conv2D(32, (3,3), input_shape=(28,28,1), activation='relu'))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2)) # Dense 사이에 위치
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2)) # Dense 사이에 위치
model.add(Dense(8, activation='softmax'))  
model.summary()
```

loss: 0.4578 - accuracy: 0.8479 - val_loss: 1.6113 - val_accuracy: 0.6371

은닉층 →

FASHION - YEAR

Project_sowon_5

- ✓ 과대적합 해결을 위한 Modeling

ResNet50 활용 시도

```
1 from tensorflow.keras.applications import ResNet50
2 from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D
3 from tensorflow.keras.models import Model
4 from tensorflow.keras.optimizers import Adam
5
6 # ResNet50
7 model_ = ResNet50(weights='imagenet', include_top=False, input_shape=(400, 300, 3))
8
9 # ResNet-50의 출력 레이어를 조정하여 새로운 분류 레이어를 추가
10 x = model_.output
11 x = GlobalAveragePooling2D()(x)
12 x = Dense(1024, activation='relu')(x) # 원하는 뉴런 수로 수정
13 output_ = Dense(8, activation='sigmoid')(x) # 분류 문제인 경우, 클래스 수에 맞게 조정
14
15 # 새로운 모델 정의
16 model = Model(inputs=model_.input, outputs=output_)
```

FASHION MALAR

Project_sowon_5

- ✓ 과대적합 해결을 위한 Modeling

300/300 [=

ResNet50 활용 시도

```
1 from tensorflow.keras.applications import ResNet50
2 from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D
3
4 # ResNet50
5 model_ = ResNet50(weights='imagenet', include_top=False)
6
7 # ResNet-50의 출력 레이어를 조정하여 새로운 분류 레이어를 추가
8
9 x = model_.output
10 x = GlobalAveragePooling2D()(x)
11 x = Dense(1024, activation='relu')(x) # 원하는 뉴런 수로 수정
12 output_ = Dense(8, activation='sigmoid')(x) # 분류 문제인 경우, 클래스 수에 맞게 조정
13
14
15 # 새로운 모델 정의
16 model = Model(inputs=model_.input, outputs=output_)
```

4429s

15s/step

FASHION - YEAR

Project_sowon_6

- ✓ 과대적합 해결을 위한 Modeling

규제 : L2

초기화 : He_Normal

배치정규화

적용

```
model=Sequential()
model.add(Conv2D(32, 3, input_shape=(400,300,3), kernel_regularizer='l2', kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D())

model.add(Conv2D(64, 3, kernel_regularizer='l2', kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D())

model.add(Conv2D(128, 3, kernel_regularizer='l2', kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D())

model.add(Conv2D(256, 3, kernel_regularizer='l2', kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D())

model.add(Flatten())

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5)) # Dense 사이에 위치
model.add(Dense(100, activation='relu'))|#
model.add(Dropout(0.5)) # Dense 사이에 위치
model.add(Dense(8, activation='softmax'))
```

FASHION - YEAR

Project_sowon_6

✓ 과대적합 해결을 위한 Modeling

```
model=Sequential()
model.add(Conv2D(32, 3, input_shape=(400,300,3), kernel_regularizer='l2', kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D())

model.add(Conv2D(64, 3, kernel_regularizer='l2', kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D())

model.add(Flatten())

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5)) # Dense 사이에 위치
model.add(Dense(100, activation='relu'))|
model.add(Dropout(0.5)) # Dense 사이에 위치
model.add(Dense(8, activation='softmax'))
```

규제 + Dropout 적용
loss: 2.1305 - accuracy: 0.1226

loss: 2.1086 - val_loss: 2.1086 - val_accuracy: 0.1138

FASHION - YEAR

Project_sowon_7

- ✓ 과대적합 해결을 위한 Modeling

가장 처음 모델로
돌아와서
은닉층 추가
및
dropout

```
model=Sequential()
model.add(Conv2D(128, 3, input_shape=(400,300,3), activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(64, 3, activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(32, 3, activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(16, 3, activation='relu'))
model.add(MaxPool2D())

model.add(Flatten())

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5)) # Dense 사이에 위치
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5)) # Dense 사이에 위치
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5)) # Dense 사이에 위치
model.add(Dense(8, activation='softmax'))

model.summary()
```

FASHION YEAR

Project_sowon_7

- ✓ 과대적합 해결을 위한 Modeling

loss: 2.0798 - accuracy: 0.1148
돌이도
은닉층 추가
및
dropout

```
model=Sequential()  
model.add(Conv2D(128, 3, input_shape=(400,300,3), activation='relu'))  
model.add(MaxPool2D())  
model.add(Conv2D(64, 3, activation='relu'))  
model.add(MaxPool2D())  
model.add(Conv2D(32, 3, activation='relu'))  
model.add(MaxPool2D())  
model.add(Conv2D(16, 3, activation='relu'))  
model.add(MaxPool2D())  
  
model.add(Flatten())  
model.add(Dense(100, activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(100, activation='relu'))  
model.add(Dropout(0.5)) # Dense 사이에 위치  
model.add(Dense(8, activation='softmax'))  
  
model.summary()
```

FASHION - YEAR

최종 모델 - CNN

```
1 # 모델 설계 - CNN
2 from keras.models import Sequential
3 from keras.layers import Dense, Conv2D, Dropout, MaxPool2D, Flatten
4
5 model=Sequential()
6 model.add(Conv2D(128, 3, input_shape=(400,300,3), activation='relu'))
7 model.add(MaxPool2D())
8 model.add(Conv2D(64, 3, activation='relu'))
9 model.add(MaxPool2D())
10 model.add(Conv2D(32, 3, activation='relu'))
11 model.add(MaxPool2D())
12 model.add(Conv2D(16, 3, activation='relu'))
13 model.add(MaxPool2D())
14
15 model.add(Flatten())
16
17 model.add(Dense(100, activation='relu'))
18 model.add(Dense(8, activation='softmax'))
19
20 model.summary()
```



나서영

모델 유형: 지도학습 - CNN 예측 모델

타겟 : 패션 스타일별 월 수입

모델1. 가장 기본적인 CNN 모형

```
# 모델 구현
model = Sequential()

# 컨볼루션 레이어 추가
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(100,75,3)))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Dropout(0.25))
model.add(Flatten())

model.add(Dense(32, kernel_initializer='he_normal', activation='relu'))
model.add(Dense(6, kernel_initializer='he_normal', activation='softmax'))

# 애포크 별 최고 성능 모델 저장
checkpoint = ModelCheckpoint('./best_model4.hdf5', monitor='val_loss', verbose=True, save_best_only=True)
# 얼리스팅
early_stop = EarlyStopping(monitor='val_loss', patience=30)
# 러닝레이트
def scheduler(epoch, lr):
    if epoch > 20:
        return 0.000001
    else:
        return lr

learning_rate = LearningRateScheduler(scheduler)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(feature, label, epochs=1000, batch_size=32, validation_split=0.2, callbacks=[checkpoint, early_stop, learning_rate], verbose=1)
```

컨볼루션 층 2개

층별 뉴런 32개

배치 정규화 2회

차원 축소 2회

Train_accuracy = 0.33

Val_accuracy = 0.29

모델 2. 욕심을 가득 담은 CNN 모형

```
# 모델 구현
model = Sequential()

# 컨볼루션 레이어 추가
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(100,75,3)))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(32, kernel_initializer='he_normal', activation='relu'))
model.add(Dense(6, kernel_initializer='he_normal', activation='softmax'))
```

컨볼루션 층 4개

층별 뉴런 32개

배치 정규화 4회

차원 축소 4회

Train_accuracy = 0.29

Val_accuracy = 0.27

모델 3. 4번은 너무 과해요. 2번으로 합시다. CNN 모형

```
# 모델 구현
model = Sequential()

# 컨볼루션 레이어 추가
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(100,75,3)))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(32, kernel_initializer='he_normal', activation='relu'))
model.add(Dense(6, kernel_initializer='he_normal', activation='softmax'))
```

컨볼루션 층 4개

층별 뉴런 32개

배치 정규화 2회

차원 축소 2회

Train_accuracy = 0.31

Val_accuracy = 0.29

모델 4. 배치 정규화/드롭아웃 그게 뭔데.. CNN 모형

```
# 모델 구현
model = Sequential()

# 컨볼루션 레이어 추가
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(100,75,3)))
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))

model.add(Flatten())

model.add(Dense(32, kernel_initializer='he_normal', activation='relu'))
model.add(Dense(6, kernel_initializer='he_normal', activation='softmax'))
```

컨볼루션 층 4개

층별 뉴런 32개

배치 정규화, 드롭아웃 삭제

차원 축소 2회

Train_accuracy = 0.33

Val_accuracy = 0.27

모델 5. 컬럼에 규제를 추가한 CNN 모형

```
# 모델 구현
model = Sequential()

# 컨볼루션 레이어 추가
model.add(Conv2D(32, kernel_size=(3,3), kernel_regularizer='L2', input_shape=(100,75,3)))
model.add(Activation('ReLU'))

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), kernel_regularizer='L2', input_shape=(100,75,3)))
model.add(Activation('ReLU'))

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(32, kernel_initializer='he_normal', activation='relu'))
model.add(Dense(6, kernel_initializer='he_normal', activation='softmax'))
```

컨볼루션 층 4개

층별 뉴런 32개

배치 정규화, 드롭아웃 삭제

차원 축소 2회

L1, L2, L1L2 규제 추가

Train_accuracy = 0.28

Val_accuracy = 0.27

모델 6. 튜닝의 끝은 순정! 단순한 CNN 모형

```
# 모델 구현
model = Sequential()

# 컨볼루션 레이어 추가
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(100,75,3)))
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(32, kernel_initializer='he_normal', activation='relu'))
model.add(Dense(6, kernel_initializer='he_normal', activation='softmax'))
```

컨볼루션 층 2개

층별 뉴런 32개

배치 정규화, 드롭아웃 삭제

차원 축소 2회

Train_accuracy = 0.36

Val_accuracy = 0.33

모델 6. 튜닝의 끝은 순정! 단순한 CNN 모형

다 좋은데, EarlyStopping에 학습이 중지된다.

그렇다면 중지 기준을 모두 풀어준다면?!

모델 6. 실컷 학습해봐라 궁극의 CNN 모형

```
# 모델 구현
model = Sequential()

# 컨볼루션 레이어 추가
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(100,75,3)))
model.add(Activation('ReLU'))

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Activation('ReLU'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(32, kernel_initializer='he_normal', activation='relu'))
model.add(Dense(6, kernel_initializer='he_normal', activation='softmax'))

# 에포크 별 최고 성능 모델 저장
checkpoint = ModelCheckpoint('./best_model4.hdf5', monitor='val_loss', verbose=True, save_best_only=True)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(feature, label, epochs=1000, batch_size=32, validation_split=0.2, callbacks=[checkpoint], verbose=1)
```

컨볼루션 층 2개

층별 뉴런 32개

배치 정규화, 드롭아웃 삭제

차원 축소 1회

학습 제한 삭제

Train_accuracy = 0.66

Val_accuracy = 0.20

과소적합과 과대적합을 동시에 달성하는 쾌거!

모델 7. 믿을 것은 AutoKeras 뿐

```
# 이미지 분류기 생성
clf = ak.ImageClassifier(max_trials=10) # max_trials는 시도할 모델의 수

# 콜백함수 지정
model_path = "/content/drive/MyDrive/ml/model/my_autokeras_model"
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
    model_path, monitor='val_loss', save_best_only=True)

# 모델 학습
clf.fit(feature, target, epochs=1000, callbacks=[model_checkpoint])

# 모델을 원하는 경로에 저장
clf.export_model(model_path)
```

모델 7. 믿을 것은 AutoKeras 뿐

```
Trial 2 Complete [02h 15m 41s]
val_loss: 1.610053300857544

Best val_loss So Far: 1.610053300857544
Total elapsed time: 02h 15m 41s

Search: Running Trial #3

Value          | Best Value So Far | Hyperparameter
efficient      | resnet             | image_block_1/block_type
True           | True               | image_block_1/normalize
True           | True               | image_block_1/augment
True           | True               | image_block_1/image_augmentation_1/horizontal_flip
False          | True               | image_block_1/image_augmentation_1/vertical_flip
0              | 0                  | image_block_1/image_augmentation_1/contrast_factor
0              | 0                  | image_block_1/image_augmentation_1/rotation_factor
0.1            | 0.1                | image_block_1/image_augmentation_1/translation_factor
0              | 0                  | image_block_1/image_augmentation_1/zoom_factor
True           | None               | image_block_1/efficient_net_block_1/pretrained
b7             | None               | image_block_1/efficient_net_block_1/version
True           | None               | image_block_1/efficient_net_block_1/trainable
True           | None               | image_block_1/efficient_net_block_1/imagenet_size
global_avg     | global_avg         | classification_head_1/spatial_reduction_1/reduction_type
0              | 0                  | classification_head_1/dropout
adam           | adam               | optimizer
2e-05          | 0.001              | learning_rate
```

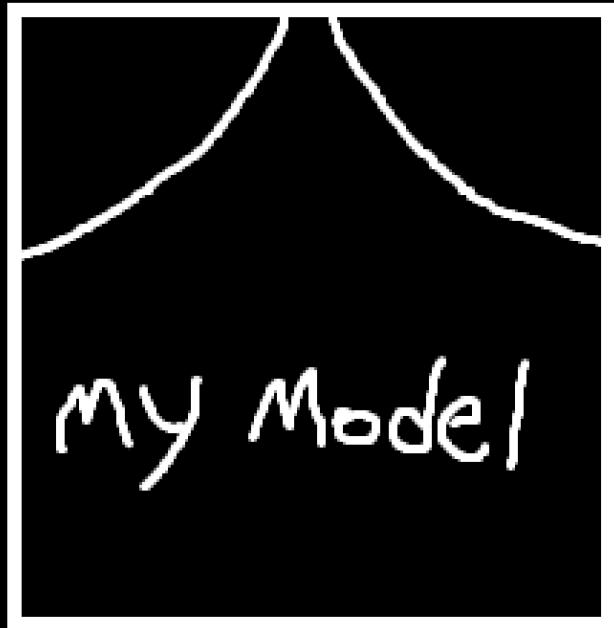
오차율 줄지 않음

점수 상승하지 않음

모델 6. 튜닝의 끝은 순정! 단순한 CNN 모형

‘최고’ 모델이 아닌 ‘차악’ 모델로 선정되었다.

나의 모델, 왜 실패했을까?



원인1. 이미지 데이터 배경의 다양성

원인2. 이미지 크기의 과도한 축소

원인3. 소득과 옷의 상관관계?

원인4. 라벨별 데이터 사이즈 차이 ?!

차선책. 가중치를 직접 기입한 보정된 CNN 모형

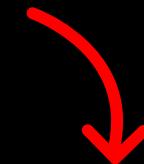
아이디어:

비뚤어진 모델 predict 점수에 가중치를 곱해

제대로 된 수치를 추측할 수 있지 않을까?

3~6번 값이 당선될 확률을 높여주자!

income
2 7113
1 6744
3 3332
4 2039
6 1828
5 1440



```
arr = np.array([[1.5,1,2,2.5,3,3]])  
result = model.predict(img) * arr
```

차선책. 가중치를 직접 기입한 보정된 CNN 모형

```
score.tolist()
```

✓ 0.0s

```
[[0.22683244943618774,  
 0.23634912073612213,  
 0.16275054216384888,  
 0.12842223048210144,  
 0.11580201238393784,  
 0.12984365224838257]]
```

```
result.tolist()
```

✓ 0.0s

```
[[0.3402486741542816,  
 0.23634912073612213,  
 0.32550108432769775,  
 0.3210555762052536,  
 0.3474060371518135,  
 0.3895309567451477]]
```

1유형 -> 5유형으로 상승



단, 임의의 가중치 조정에 대하여

1. 컬럼별 가중치가 적절한 수준으로 가해졌는가?

2. 가중치를 조정하는 것 자체가 올바른가?

두 가지 의문이 남았다.



이태형

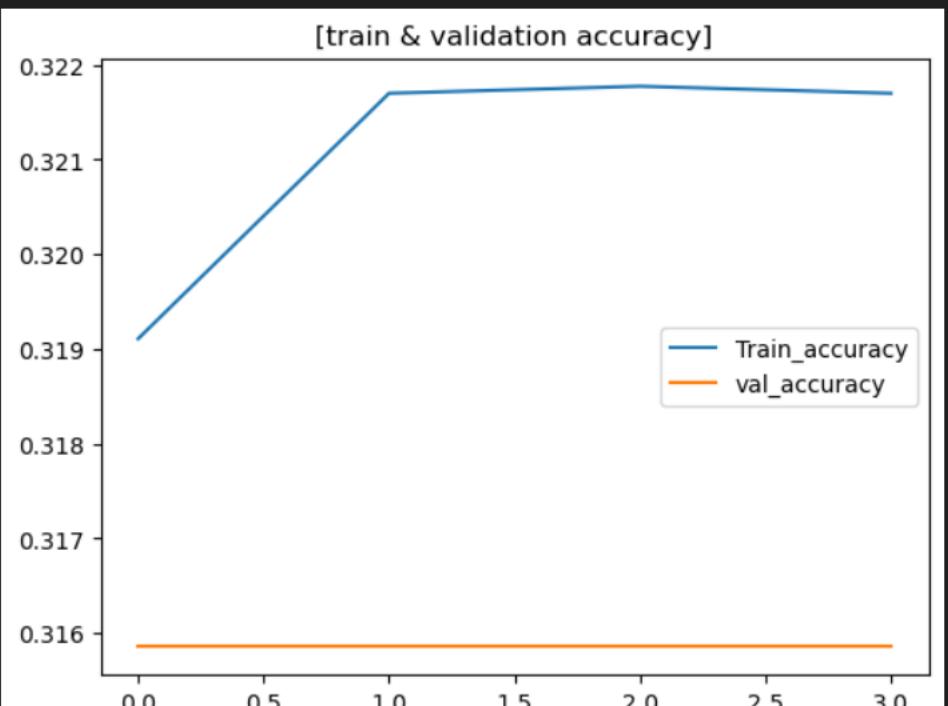
모델 유형: 지도학습 - CNN 예측 모델

타겟 : 패션 스타일별 연령대

데이터 사이즈

<input type="checkbox"/>	NAME ▲	SIZE ▲
<input type="checkbox"/>	VS_woman_1990.zip	3GB
<input type="checkbox"/>	VS_woman_1980.zip	3GB
<input type="checkbox"/>	VS_woman_1950.zip	3GB
<input type="checkbox"/>	VS_woman_2000.zip	3GB
<input type="checkbox"/>	VS_woman_1970.zip	4GB
<input type="checkbox"/>	VS_man_2019.zip	5GB
<input type="checkbox"/>	VS_woman_2019.zip	6GB

CNN과 성능



Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 100, 75, 16)	448
activation_8 (Activation)	(None, 100, 75, 16)	0
max_pooling2d_4 (MaxPooling2D)	(None, 50, 37, 16)	0
conv2d_5 (Conv2D)	(None, 50, 37, 16)	2320
activation_9 (Activation)	(None, 50, 37, 16)	0
max_pooling2d_5 (MaxPooling2D)	(None, 25, 18, 16)	0
flatten_2 (Flatten)	(None, 7200)	0
dense_6 (Dense)	(None, 16)	115216
activation_10 (Activation)	(None, 16)	0
dense_7 (Dense)	(None, 16)	272
activation_11 (Activation)	(None, 16)	0
dense_8 (Dense)	(None, 4)	68

Total params: 118324 (462.20 KB)
Trainable params: 118324 (462.20 KB)
Non-trainable params: 0 (0.00 Byte)

Colab 사용

▼ 훈련용과 테스트용 데이터 준비

AutoKeras 사용

```
Trial 1 Complete [00h 31m 05s]
val_loss: 1.3527686595916748
```

```
Best val_loss So Far: 1.3527686595916748
Total elapsed time: 00h 31m 05s
```

```
Search: Running Trial #2
```

Value	Best Value So Far	Hyperparameter
resnet	'vanilla'	'image_block_1/block_type'
True	'True'	'image_block_1/normalize'
True	'False'	'image_block_1/augment'
True	'None'	'image_block_1/image_augmentation_1/horizontal_flip'
True	'None'	'image_block_1/image_augmentation_1/vertical_flip'
0	'None'	'image_block_1/image_augmentation_1/contrast_factor'
0	'None'	'image_block_1/image_augmentation_1/rotation_factor'
0.1	'None'	'image_block_1/image_augmentation_1/translation_factor'
0	'None'	'image_block_1/image_augmentation_1/zoom_factor'
False	'None'	'image_block_1/res_net_block_1/pretrained'
resnet50	'None'	'image_block_1/res_net_block_1/version'
True	'None'	'image_block_1/res_net_block_1/imagenet_size'
global_avg	'flatten'	'classification_head_1/spatial_reduction_1/reduction_type'
0	'0.5'	'classification_head_1/dropout'
adam	'adam'	'optimizer'
0.001	'0.001'	'learning_rate'

```
Epoch 1/1000
423/423 [=====] - 3894s 9s/step - loss: 1.5195 - accuracy: 0.2990 - val_loss: 4.5751 - val_accuracy: 0.2065
Epoch 2/1000
423/423 [=====] - 3727s 9s/step - loss: 1.3808 - accuracy: 0.3030 - val_loss: 1.3471 - val_accuracy: 0.2947
Epoch 3/1000
423/423 [=====] - 3713s 9s/step - loss: 1.3753 - accuracy: 0.3029 - val_loss: 1.5536 - val_accuracy: 0.1433
Epoch 4/1000
423/423 [=====] - 3827s 9s/step - loss: 1.3697 - accuracy: 0.3051 - val_loss: 1.8637 - val_accuracy: 0.2842
```

```
import autokeras as ak

# 이미지 분류기 생성
clf = ak.ImageClassifier(max_trials=10) # max_trials는 시도할 모델의 수를 정의합니다.

# 데이터 로드 및 전처리

# 모델 학습
clf.fit(train_x, train_y, epochs=1000)

# 성능 평가
print('Accuracy: {}'.format(accuracy=clf.evaluate(test_x, test_y)))
```

사건0. 첫 프로젝트 모임 전, 박** 팀원이

심각한 허리 부상으로 불참

사건1. 이** 팀원, 다운로드 버튼 보지 못해

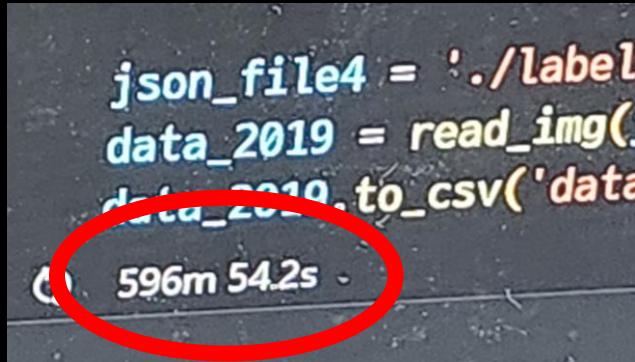
데이터 다운로드에 실패

사건2. 0]*** 팀원, 이미지 가공을 위한

코드 파일 “shift+delete”로 삭제

사건3. 힘들게 복구한 코드를 돌려놓고 귀가,

다음날 아침 터져 있는 코드를 박** 팀원이 발견



사건4. 모델 학습까지 30시간,

코랩에 코드를 돌려놓고 귀가했으나

나** 팀원의 구글 폴더가 전부 사라짐

감사합니다.