4장. 딥러닝 신경망 적용

Wrap Up

1. 텐서플로우 2.x는 기본적으로 **즉시 실행 모드**를 사용하며, **텐서**라는 개념을 통해 연산을 수행합니다.
2. **@tf.function**은 파이썬 함수를 텐서플로우 그래프 모드로 변경해줍니다.
3. 신경망은 **퍼셉트론 알고리즘**에서부터 출발합니다. **다층 퍼셉트론**을 사용하면 XOR 게이트 문제를 해결할 수 있습니다.
4. **경사하강법과 역전파**는 학습을 위해 사용되는 주요 개념입니다. 경사하강법에서는 **학습률, 가중치 초기화**에 대해 알아보았으며, **역전파**에서는 **체인 룰**을 사용하는 것을 배웠습니다.
5. 케라스에서의 개발 과정은 **[데이터 정의 → 모델 정의 → 손실함수, 옵티마이저, 평가지표 선택 → 모델 학습]**으로 이루어집니다.
6. 케라스 모델의 **첫 번째 층은 항상 입력 데이터의 형태를 전달**해주어야 합니다.
7. 대표적으로 손실 함수에는 **['mse', 'binary_crossentropy', 'categorical_crossentropy']**, 옵티마이저에는 **['sgd', 'rmsprop', 'adam']**이 있으며, 문자열로 지정하여 사용할 수 있습니다.

4장에서 만나볼 내용?

- 4장에서는 3장에서 배운 내용을 활용해서 다음 4가지 문제를 다뤄봅니다.
  1. **다중 분류(Multi Classification)**: MNIST와 Fashion-MNIST
  2. **회귀 문제(Regression)**: 보스턴 주택 가격 예측
  3. **이진 분류(Binary Classification)**: 빙산과 선박 분류
  4. **다중 레이블 분류(Multi-label Classification)**: 색깔별로 옷 분류하기

- 이번 장에서는 여러 가지 데이터셋(dataset)을 다뤄봅니다.
  사용할 데이터셋은 다양한 구조의 신경망을 실험할때 자주 사용되니 적극적으로 활용하기를 바랍니다.

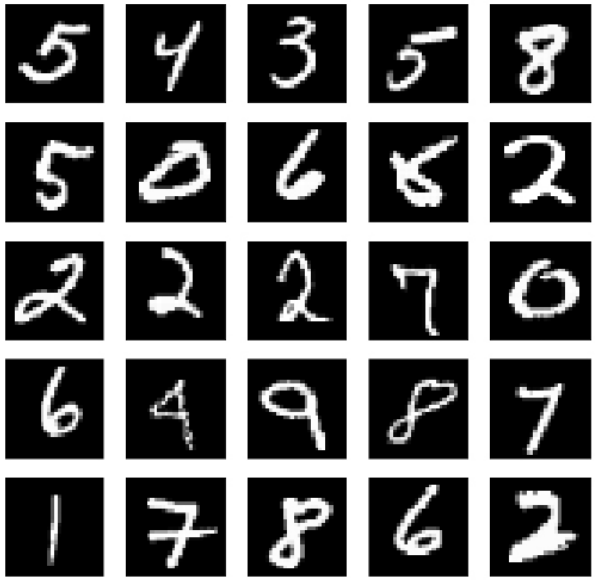
> 4장부터는 사용하는 데스크톱이나 노트북에 GPU가 내장되어 있지 않다면, 구글의 코랩이나 캐글 노트북 사용을 추천합니다.
>
> 참고로 신경망에서 적용되는 연산은 상대적으로 다른 연산에 비해 고비용의 연산에 속합니다. 이러한 고비용의 연산은 CPU의 수명을 급격히 단축시킬 수 있으니 주의해야 합니다.

첫 데이터셋

- 딥러닝계의 'Hello World'
  - MNIST와 Fashion-MNIST

- 가장 먼저 살펴볼 데이터셋: **MNIST**
  - 과거 NIST(미국 국립표준기술연구소)에서 수집한 손으로 직접 쓴 흑백 숫자
  - 0부터 9까지의 숫자를 예측하는 다중 분류 문제

- 데이터는 숫자 이미지(28, 28)와 숫자에 해당하는 레이블로 구성되어 있음
  - 60,000개 학습 데이터, 10,000개 테스트 데이터



[그림 4-1] MNIST 데이터셋 예시 샘플

MNIST 데이터셋

- 데이터 다운받기
  - 케라스에서 제공하는 데이터셋은 전부 tf.keras.datasets를 통해 접근 가능
  - load_data() 함수는 (x_train, y_train), (x_test, y_test) 형태로 분할해서 제공

[함께 해봐요] **MNIST 데이터셋 다운받기** mnist.ipynb

```
01 from tensorflow.keras.datasets.mnist import load_data
02
03 # 텐서플로우 저장소에서 데이터를 다운받습니다.
04 (x_train, y_train), (x_test, y_test) = load_data(path='mnist.npz')
```

- 데이터 형태 확인
  - **데이터, 레이블가 어떻게 구성되어 있는지 확인해보는 과정은 필수!**
  - 제공되는 코드를 통해 데이터를 그려보세요.

[함께 해봐요] **데이터의 형태 확인하기** mnist.ipynb

```
01 # 학습 데이터
02 print(x_train.shape, y_train.shape)
03 print(y_train)
04
05 # 테스트 데이터
06 print(x_test.shape, y_test.shape)
07 print(y_test)
```

```
(60000, 28, 28) (60000,)
[5 0 4 ... 5 6 8]
(10000, 28, 28) (10000,)
[7 2 1 ... 4 5 6]
```

MNIST 데이터셋

- 모델 검증을 위해 검증 데이터셋을 만듭니다
  - train_test_split() 함수 사용
  - test_size: 테스트 데이터셋 비율
  - random_state: 재생산성

[함께 해봐요] 검증 데이터 만들기 mnist.ipynb

```
01 from sklearn.model_selection import train_test_split
02
03 # 훈련/테스트 데이터를 0.7/0.3의 비율로 분리합니다.
04 x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
05                                                  test_size = 0.3,
06                                                  random_state = 777)
07 print(f'훈련 데이터 {x_train.shape} 레이블 {y_train.shape}')
08 print(f'검증 데이터 {x_val.shape} 레이블 {y_val.shape}')
```

- 학습을 위해 전처리(preprocessing)를 수행해야 함
  - 255로 나눠주어 0~1사이로 스케일 조정
  - **신경망은 스케일(scale)에 매우 민감!**
  - Dense 층 사용을 위해 **784차원의 1차원 배열**로 변환

[함께 해봐요] 모델 입력을 위한 데이터 전처리 mnist.ipynb

```
01 num_x_train = x_train.shape[0]
02 num_x_val = x_val.shape[0]
03 num_x_test = x_test.shape[0]
04
05 # 모델의 입력으로 사용하기 위한 전처리 과정입니다.
06 x_train = (x_train.reshape((num_x_train, 28 * 28))) / 255
07 x_val = (x_val.reshape((num_x_val, 28 * 28))) / 255
08 x_test = (x_test.reshape((num_x_test, 28 * 28))) / 255
09
10 print(x_train.shape)  # 모델 입력을 위해 데이터를 784차원으로 변경합니다.
```

여러 가지 전처리 방법 – 스케일링

Normalization(MinMax)

$$X = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Robust Normalization

$$X = \frac{x - x_{2/4}}{x_{3/4} - x_{1/4}}$$

Standardization

$$X = \frac{x - x_{,\,mean}}{x_{std}}$$

[그림 4-2] 여러 가지 전처리 방법 – 스케일링

MNIST 데이터셋

- 모델 마지막 층에서 **소프트맥스(softmax)** 함수를 사용하므로 범주형 레이블로 변환
  - to_categorical() 함수

```
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]]
```

- 모델을 구성합니다.
  - 모델은 **784차원의 데이터를 입력(input)**으로 받고, **열 개의 출력(output)**을 가짐

[함께 해봐요] **모델 구성하기** mnist.ipynb

```
01 from tensorflow.keras.models import Sequential
02 from tensorflow.keras.layers import Dense
03
04 model = Sequential()
05 # 입력 데이터의 형태를 꼭 명시해야 합니다.
06 # 784차원의 데이터를 입력으로 받고, 64개의 출력을 가지는 첫 번째 Dense층
07 model.add(Dense(64, activation = 'relu', input_shape = (784, )))
08 model.add(Dense(32, activation = 'relu'))      # 32개의 출력을 가지는 Dense층
09 model.add(Dense(10, activation = 'softmax'))  # 10개의 출력을 가지는 신경망
```

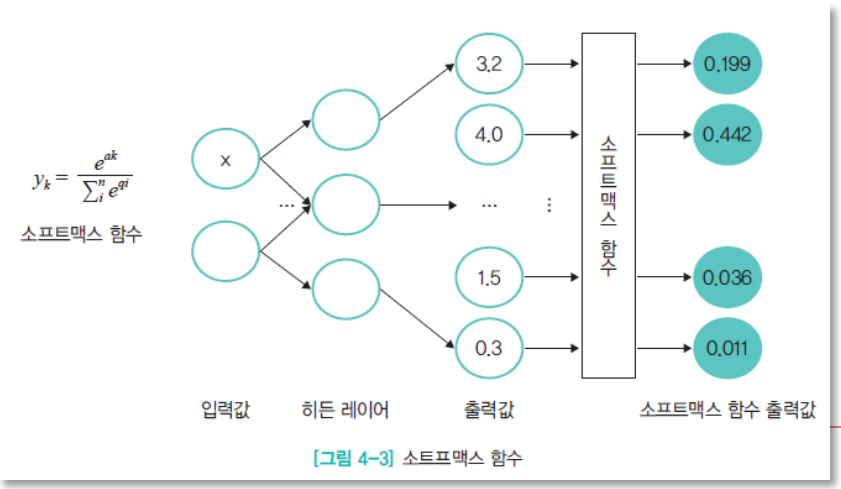
시그모이드, 소프트맥스 함수

- 소프트맥스 함수는 출력값의 범위 안에서 확률로써 해석할 수 있기에,
  결과의 해석이 더욱 용이함
  - 다른 표현: 일반적으로 확률을 구하는 방법과 비슷하므로 각 클래스에 해당하는 값들이 서로 영향을 줄 수 있어 비교에 용이

- 예시:
  - (불고기버거, 치즈버거, 치킨버거) = (3.1, 3.0, 2.9)
  - (불고기버거, 치즈버거, 치킨버거) = (2.0, 1.0, 0.7)

주의! 확률의 합은 항상 1입니다. 이 예시는 명확한 설명을 위해 1을 벗어나는 극단적인 수를 사용했습니다.

- 위 예시에서 위와 아래의 숫자 비교에서 어느 것이 더 명확한가요?
  - 제공되는 코드를 통해 값을 비교해보세요.



[그림 4-3] 소프트맥스 함수

MNIST 데이터셋

- 모델 구성의 마지막 단계는 **손실 함수, 옵티마이저, 평가지표를 설정**하는 것
  - 다중 분류 문제에서 손실 함수는 categorical_crossentropy 함수를 사용
  - **크로스 엔트로피(cross-entropy)**는 정보이론에서 파생된 개념
  - 서로의 결괏값이 틀린 경우 로그 함수의 특징대로 무한대로 발산하고, 동일한 경우 0으로 수렴
  - 옵티마이저는 Adam, 평가지표는 정확도(Acc, Accuracy)를 사용합니다.

[함께 해봐요] **학습과정 설정하기** mnist.pynb

```
01  model.compile(optimizer='adam',  # 옵티마이저: Adam
02                # 손실함수: categorical_crossentropy
03                loss = 'categorical_crossentropy',
04                # 모니터링 할 평가지표: acc
05                metrics=['acc'])
```

MNIST 데이터셋

- validation_data에 검증 데이터셋을 전달하고, 128 배치크기를 사용하며, 전체 데이터를 30회 반복

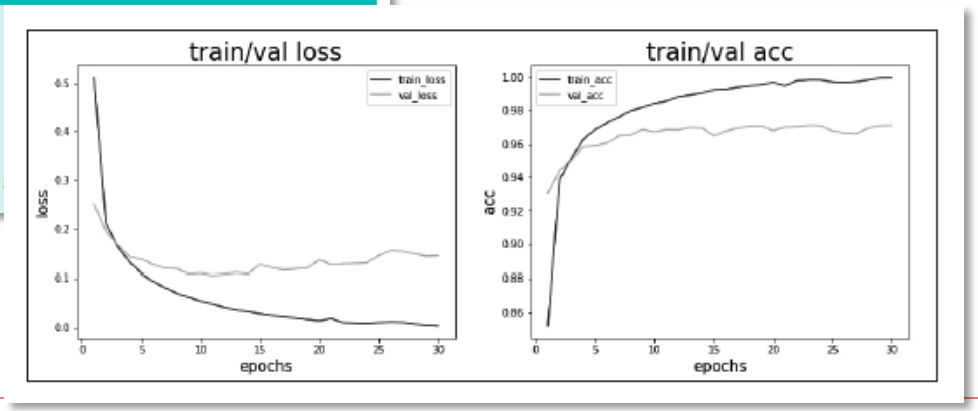
[함께 해봐요] **모델 학습하기** mnist.ipynb

```
01 history = model.fit(x_train, y_train,
02                     epochs = 30,
03                     batch_size = 128,
04                     validation_data = (x_val, y_val))
```

- model.fit() 함수는 History 객체를 전달합니다. 이를 활용하여 학습 과정을 손쉽게 모니터링 할 수 있음

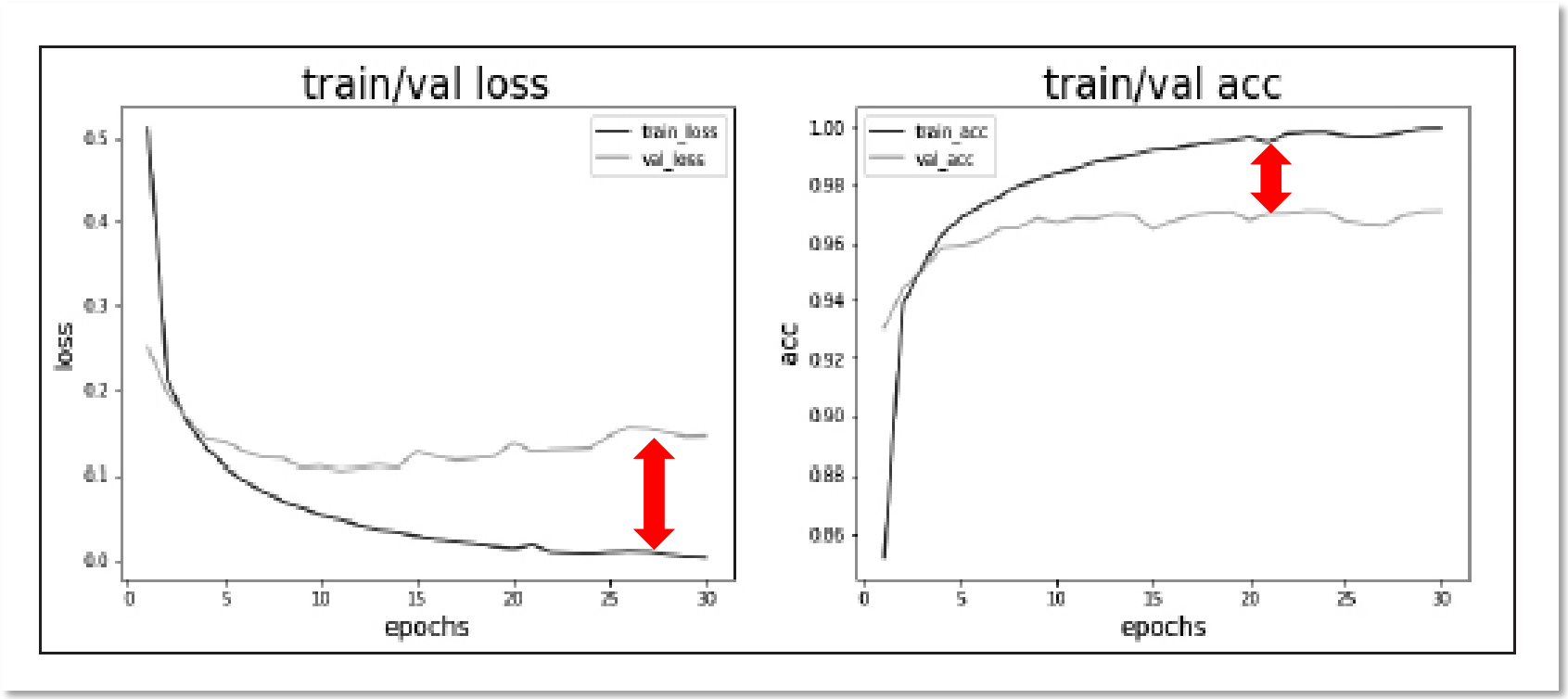
[함께 해봐요] **학습 결과 그려보기** mnist.ipynb

```
01 import matplotlib.pyplot as plt
02
03 his_dict = history.history
04 loss = his_dict['loss']
05 val_loss = his_dict['val_loss']  # 검증 데이터가 있는
```



MNIST 데이터셋

- 두 그래프가 어디서부터 벌어지나요?
  - **과대적합 문제**가 나타난 것
  - 데이터 특성, 모델 구조 등을 수정해보고 재학습
  - 벌어지기 전까지의 모델을 사용하여 결과를 확인하고 이를 저장 및 기록



MNIST 데이터셋

- 평가해보고, 결과를 확인

[함께 해봐요] **모델 평가하기** mnist.ipynb

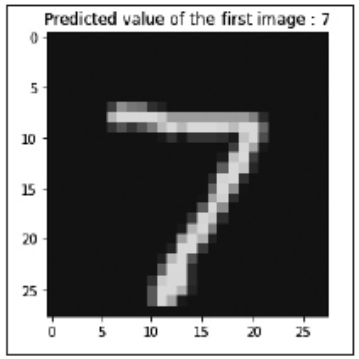
```
01  model.evaluate(x_test, y_test)
```

```
[0.1319049060290734, 0.9735]
```

손실값 정확도

[함께 해봐요] **학습된 모델을 통해 값 예측하기**

```
01  import numpy as np
02
03  results = model.predict(x_test)
04  print(results.shape)
05  np.set_printoptions(precision=7)  # numpy 소수점 제한
06  print(f'각 클래스에 속할 확률 : \n{results [0]}')
```




```
(10000, 10)
각 클래스에 속할 확률 :
[5.9363152e-12 7.4104497e-18 9.0246495e-08 3.2391474e-06 2.5945717e-12
 1.6788119e-08 1.2139338e-26 9.9999356e-01 1.2242263e-09 3.0679507e-06]
```

숫자 7

MNIST 데이터셋 – End

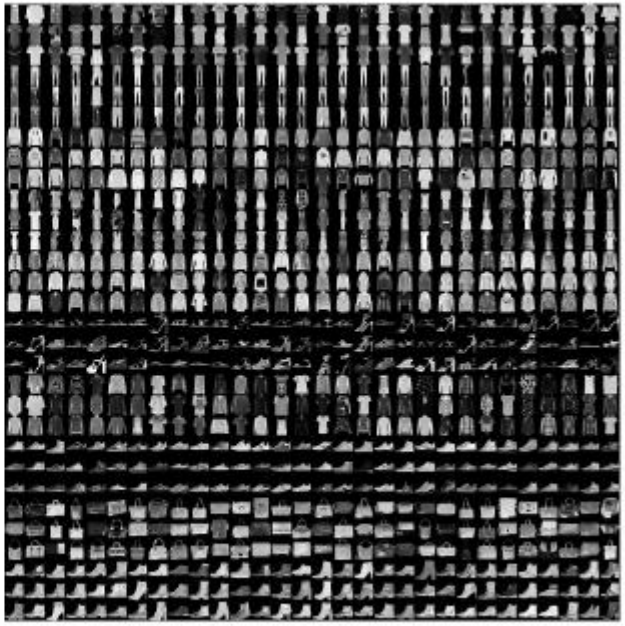
- 모델을 해석해보자
  - 혼동 행렬(Confusion Matrix)
  - 분류 보고서(Classification Report)

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.97	0.98	1135
2	0.96	0.98	0.97	1032
3	0.97	0.97	0.97	1010
4	0.98	0.95	0.97	982
5	0.98	0.96	0.97	892
6	0.97	0.98	0.98	958
7	0.97	0.96	0.97	1028
8	0.95	0.96	0.96	974
9	0.94	0.98	0.96	1009
micro avg	0.97	0.97	0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

Fashion-MNIST 데이터셋

- MNIST와 매우 비슷한 데이터셋
  - 딥러닝계의 Hello World
  - 60,000개 학습 데이터, 10,000개 테스트 데이터

레이블	의류 품목
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sendal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



[그림 4-4] Fashion-MNIST 데이터셋(Zalando, MIT License)

Fashion-MNIST 데이터셋

- 데이터 다운로드

[함께 해봐요] Fashion-MNIST 데이터셋 다운받기 fashion-mnist.ipynb

```
01 from tensorflow.keras.datasets.fashion_mnist import load_data
02
03 # Fashion-MNIST 데이터를 다운받습니다.
04 (x_train, y_train), (x_test, y_test) = load_data()
05 print(x_train.shape, x_test.shape)
```

- 전처리 및 검증 데이터셋

```
01 # 0~1 범위로 만듭니다.
02 x_train = x_train / 255
03 x_test = x_test / 255
```

스케일링

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

범주형 레이블

검증 데이터셋

```
x_train, x_val, y_train, y_val =
train_test_split(x_train, y_train,
                 test_size = 0.3,
                 random_state = 777)
```

Fashion-MNIST 데이터셋

- 비교를 위한 두 가지 모델 구성
  - (28, 28) → 784 차원으로 만드는 것 대신,
  - Flatten() 층을 사용
  - Flatten(): (128, 6, 2, 2) → (128, 24)

더 깊게 구성!

[함께 해봐요] **첫 번째 모델 구성하기** fashion-mnist.ipynb

```
first_model = Sequential()
first_model.add(Flatten(input_shape = (28, 28)))
first_model.add(Dense(64, activation = 'relu'))
first_model.add(Dense(32, activation = 'relu'))
first_model.add(Dense(10, activation = 'softmax'))
```

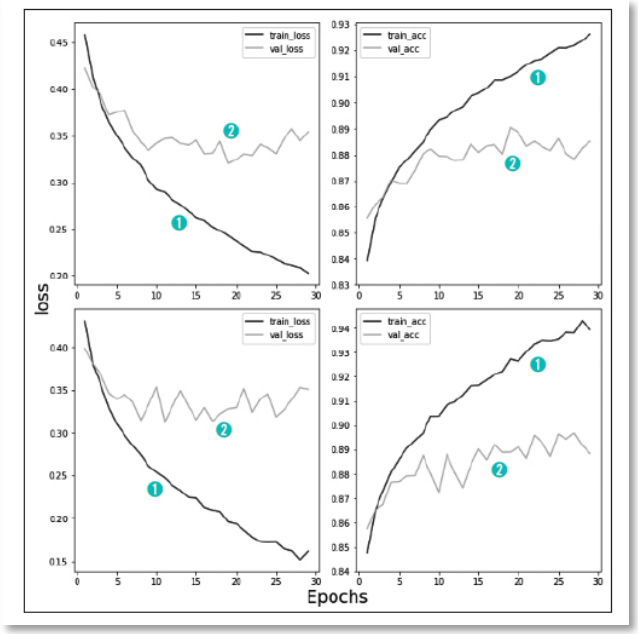
[함께 해봐요] **두 번째 모델 구성하기** fashion-mnist.ipynb

```
second_model = Sequential()
second_model.add(Flatten(input_shape = (28, 28)))
second_model.add(Dense(128, activation = 'relu'))
second_model.add(Dense(128, activation = 'relu'))
second_model.add(Dense(32, activation = 'relu'))
second_model.add(Dense(10, activation = 'softmax'))
```

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

Fashion-MNIST 데이터셋 – End

- 결과 비교
  - 제공되는 코드를 통해 결과를 그려보세요!
  - 어느 지점에서 벌어지기 시작하고 있나요?
  - 어떻게 해야할까요?



- 결과 해석
  - **모델을 깊게 구성 → 높은 성능, <u>But 과대적합(파라미터 수 증가)</u>**
  - 모델의 깊이는 데이터에 적합하게 결정해야 함
  - 유명한 데이터셋이나 유사 분야에서 높은 성능을 보여준 모델 구조를 참고하여 구성해보고 실험을 진행

보스턴 주택 가격 예측

- 1970년대 보스턴 지역의 범죄율, 토지 지역의 비율, 방 개수 등 14개 특성으로 이루어진 데이터
  - **연속적인 값을 예측하는 회귀 문제(Regression)**

- 데이터 다운로드

[함께 해봐요] **보스턴 주택 가격 데이터셋 다운받기** boston.ipynb

```
01 from tensorflow.keras.datasets.boston_housing import load_data
02
03 # 데이터를 다운받습니다.
04 (x_train, y_train), (x_test, y_test) = load_data(path='boston_housing.npz',
05                                                  test_split=0.2,
06                                                  seed=777)
```

- 총 데이터는 506개로 상당히 적은 편에 속함

  - 학습 데이터 : 404개

  - 테스트 데이터 : 102개

보스턴 주택 가격 예측

- 다시 데이터 설명으로
  - 13개의 특성, 레이블은 주택 가격의 중간 가격($1000 단위)
  - 특성마다 값을 나타내는 방법이 다르므로 **표준화(Standardization)를 수행**함
    - ex) 범죄율 0 ~ 1, 방 개수 3 ~ 9 등
  - 표준화는 데이터의 평균을 빼고, 표준편차로 나눠줍니다 → Z-Normalization, 표준정규분포 참고
    - 다른 머신러닝 전처리 방법에도 자주 사용되니 꼭 알아둡시다.
    - (x – mean) / std

- **주의!** 테스트 데이터셋에도 학습 데이터의 평균, 표준편차를 활용

[함께 해봐요] **데이터 전처리 및 검증 데이터셋 만들기** boston.ipynb

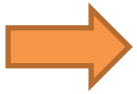
```
01 import numpy as np
02
03 # 데이터 표준화
04 mean = np.mean(x_train, axis = 0)
05 std = np.std(x_train, axis = 0)
06
07 x_train = (x_train - mean) / std
08 x_test = (x_test - mean) / std
09
10 # 검증 데이터셋을 만듭니다.
11 from sklearn.model_selection import train_test_split
12
13 x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
14                                                   test_size = 0.33,
15                                                   random_state = 777)
```

보스턴 주택 가격 예측 – End

- 모델을 구성하고, 학습을 진행

[함께 해봐요] **모델 구성하기** boston.ipynb

```
01 from tensorflow.keras.models import Sequential
02 from tensorflow.keras.layers import Dense
03
04 model = Sequential()
05 # 입력 데이터의 형태를 꼭 명시해야 합니다.
06 # 13차원의 데이터를 입력으로 받고, 64개의 출력을 가지는 첫 번째 Dense층
07 model.add(Dense(64, activation = 'relu', input_shape = (13, )))
08 model.add(Dense(32, activation = 'relu'))  # 32개의 출력을 가지는 Dense층
09 model.add(Dense(1))  # 하나의 값을 출력합니다.
10
11 model.compile(optimizer = 'adam', loss = 'mse', metrics = ['mae'])
```



[함께 해봐요] **학습하고 평가하기** boston.ipynb

```
01 history = model.fit(x_train, y_train, epochs = 300,
                       validation_data = (x_val, y_val))
02 model.evaluate(x_test, y_test)
```

- 모델의 마지막 Dense 층에서 별도의 활성화 함수를 사용하지 않음
  - 인자를 설정하지 않은 경우, default는 **'linear'**로 설정
- 손실 함수는 회귀 문제에서 주로 사용되는 평균 제곱 오차(MSE; Mean Squared Error)를 사용
- **결과: 2.261**
  - 해석 : 실제 가격과 예측 가격의 차이가 평균적으로 2,200 달러정도 차이가 있음을 의미

```
[9.3971145994523, 2.2616422]
```

K-Fold 사용하기

- 데이터 개수가 적은 경우, 성능을 향상시킬 수 있는 좋은 방법: 교차 검증
  - 여기서는 K-Fold 교차 검증을 실습해보자
  - 제공되는 코드를 참고

```
# K-폴드를 진행해봅니다.
k = 3

# 주어진 데이터셋을 k만큼 등분합니다.
# 여기서는 3이므로 학습 데이터셋(404개)을 3등분하여
# 한 개는 검증셋으로, 나머지 두 개는 학습 데이터셋으로 활용합니다.
kfold = KFold(n_splits=k, random_state = 777)
```

K = 3, KFold 정의

```
# K-폴드 과정에서 재사용을 위해 모델을 반환하는 함수를 정의합니다.
def get_model():
    model = Sequential()
    model.add(Dense(64, activation = 'relu', input_shape = (13, )))
    model.add(Dense(32, activation = 'relu'))
    model.add(Dense(1))

    model.compile(optimizer = 'adam', loss = 'mse', metrics = ['mae'])

    return model
```

모델의 재사용을 위해 get_model() 함수 정의

kfold.split()을 통해 학습 데이터, 검증 데이터를 분리하여 학습을 진행

```
# k번 진행합니다.
for train_index, val_index in kfold.split(x_train):
    # 해당 인덱스는 무작위로 생성됩니다.
    # 무작위로 생성해주는 것은 과대적합을 피할 수 있는 좋은 방법입니다.
    x_train_fold, x_val_fold = x_train[train_index], x_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]
```

K-Fold 사용하기 – End

- 결과는 mae_list에 저장

```
_, test_mae = model.evaluate(x_test, y_test)
mae_list.append(test_mae)
```

- 결과를 확인해보면, 모든 모델이 전부 좋은 성능을 가지진 않음 `[2.057369, 1.9423964, 2.1546433]`
  - 각 폴드에서 사용한 학습, 검증 데이터가 다르기 때문
  - 두 번째 모델은 상대적으로 테스트 모델과 더 비슷한 분포의 데이터를 학습했다고 볼 수 있음

- 이 때문에, 최종 학습 결과는 평균을 사용
  - 최종 성능은 `2.0514696`

- 이처럼 교차 검증과 같이 성능을 향상시킬 수 있는 수많은 방법이 존재하지만,
  - 이보다 더 중요하고, 모델의 성능을 극적으로 향상시킬 수 있는 방법은 **데이터의 특성을 잘 파악하는 것!**

무슨 옷과 무슨색?

- 이번에 다뤄볼 문제는 다중 레이블 분류 문제(Multi-label Classification)
  - 다중 클래스 vs 다중 레이블

[그림 4-6] 다중 클래스와 다중 레이블 문제

- 데이터는 캐글 데이터 페이지를 통해 다운로드
  - https://www.kaggle.com/airplane2230/apparel-image-dataset-2
  - 실제 환경에서 볼 수 있는 배경 또는 인물이 존재하는 데이터셋

색깔	드레스	셔츠	바지	반바지	신발
검은색	–	–	–	–	–
파란색	–	–	–	–	–
갈색			–	–	–
초록색		–	–	–	–
빨간색	–		–		–
흰색	–		–	–	–

무슨 옷과 무슨색?

- 11,385장의 이미지를 메모리에 할당하기엔 너무 무거움 → **제네레이터 사용(5장 참고)**
  - 모든 코드는 clothes_classification 폴더 참고

- 데이터 불러오기
  - Pandas 라이브러리 사용

[함께 해봐요] **데이터 불러오기** clothes1.ipynb

```
01 import pandas as pd
02
03 DATA_PATH = './csv_data/nocolorinfo'
04
05 train_df = pd.read_csv(DATA_PATH + '/train.csv')
06 val_df = pd.read_csv(DATA_PATH + '/val.csv')
07 test_df = pd.read_csv(DATA_PATH + '/test.csv')
08
09 train_df.head()
```

	image	black	blue	brown	green	red	white	dress	shirt	pants	shorts	shoes
0	/clothes_dataset\blue_shorts\256d854b55ac32ea...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	./clothes_dataset\red_pants\584f778aece14f07c2...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
2	/clothes_dataset\green_pants\ec543ca241cefb2b...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	/clothes_dataset\brown_shorts\c8db9e0f7010592...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	/clothes_dataset\white_dress\551373c80717c5b0...	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0

  - 'image'는 이미지가 존재하는 전체 경로를 나타내며, 그 외 column은 이미지에 포함되어 있는 의류의 색과 종류를 나타냄

무슨 옷과 무슨 색?

- 이미지 제네레이터 정의

```
03  train_datagen = ImageDataGenerator(rescale = 1./255)
04  val_datagen = ImageDataGenerator(rescale = 1./255)
```

- 모델 정의

```
15  model = Sequential()
16  # 입력 데이터의 형태를 꼭 명시해야 합니다.
17  model.add(Flatten(input_shape = (112, 112, 3)))  # (112, 112, 3) -> (112 * 112 * 3)
18  model.add(Dense(128, activation = 'relu'))       # 128개의 출력을 가지는 Dense층
19  model.add(Dense(64, activation = 'relu'))        # 64개의 출력을 가지는 Dense층
20  model.add(Dense(11, activation = 'sigmoid'))     # 11개의 출력을 가지는 신경망
```

- 학습 과정 설정
  - 다중 레이블 분류 문제는 **binary_crossentropy 손실 함수**를 사용

```
22  model.compile(optimizer = 'adam',
23                loss = 'binary_crossentropy',
24                metrics = ['acc'])
```

무슨 옷과 무슨 색?

- 데이터프레임을 활용하여 학습시킬 수 있는 **flow_from_dataframe() 함수**를 사용

```
05 # Make Generator
06 train_generator = train_datagen.flow_from_dataframe(
07     dataframe=train_df,
08     directory='',
09     x_col = 'image',
10     y_col = class_col,
11     target_size = (112, 112),
12     color_mode='rgb',
13     class_mode='other',
14     batch_size=batch_size,
15     seed=42)
```

- **dataframe:** 사용할 데이터프레임 전달
- **directory:** 이미지가 존재하는 폴더 경로를 전달
- **x_col, y_col:** 데이터프레임에서 학습에 사용할 데이터가 존재하는 열과 레이블이 존재하는 열을 전달
- **target_size:** 이미지 크기를 전달받은 크기로 조절
- **class_mode:** 이진 분류(binary), 다중 분류(categorical), 다중 레이블 분류(other)

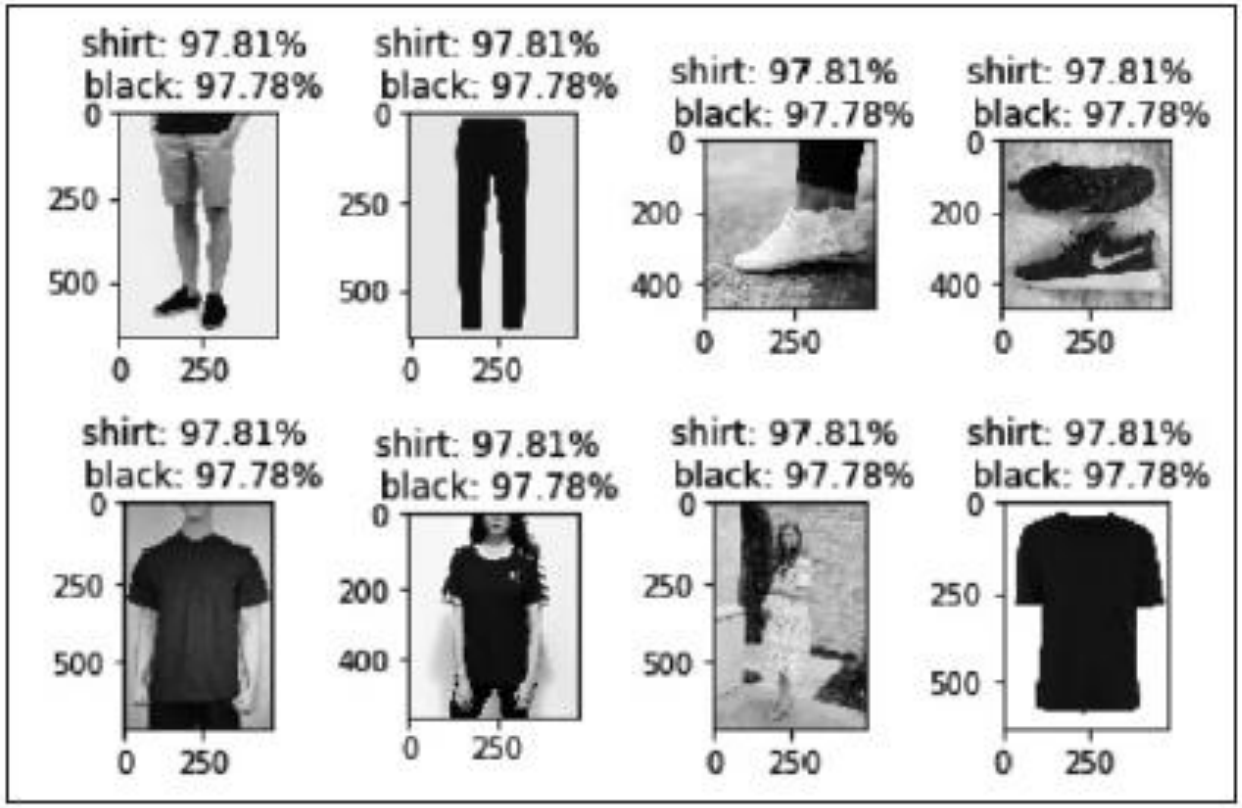
- 학습을 진행

[함께 해봐요] **제네레이터를 통해 모델 학습시키기** clothes1.ipynb

```
01 model.fit(train_generator,
02           steps_per_epoch=get_steps(len(train_df), batch_size),
03           validation_data = val_generator,
04           validation_steps=get_steps(len(val_df), batch_size),
05           epochs = 10)
```

무슨 옷과 무슨 색? – End

- 결과 확인
  - 첫 번째 행의 네 번째 결과는 신발을 셔츠로 인식!



요약 정리

1. 특정 분야가 아닌 이상, 문제에 사용되는 대표적인 데이터셋은 분명히 존재합니다. 어느 부분부터 접근해야 할지 모르겠다면, **해당 문제에 사용되는 대표적인 데이터셋과 문제에 적용된 모델을 벤치마킹**하는 것이 가장 빠른 접근 방법일 수 있습니다.
2. 신경망은 **스케일에 매우 민감하므로 적절한 전처리 과정은 필수**입니다.
3. 이진 분류: sigmoid + binary_crossentropy
   다중 분류: softmax + categorical_crossentropy
   회귀 문제: mse + mae
   다중 레이블 분류: sigmoid + binary_crossentropy
4. 모델의 **History 객체**를 활용하면 학습 과정을 더욱 직관적으로 관찰할 수 있습니다.
5. **데이터가 복잡하지 않고 충분하지도 않을 때, 모델을 깊게 구성하면 과대적합에 크게 노출될 수 있습니다.**
6. 데이터가 충분하지 않을 때, **교차 검증**은 이를 보완할 좋은 방법입니다.
7. 모델의 성능을 극적으로 향상시킬 수 있는 방법은 **데이터의 특성을 잘 파악하는 것**입니다.
8. **캐글**은 이러한 모든 과정을 경험할 수 있는 최고의 공간입니다.