4장. 신경망 적용해보기

학습목표

다중 분류(Multi Classification): MNIST와 Fashion-MNIST

- 이번 장에서는 여러 가지 데이터셋(dataset)을 다뤄봅니다.

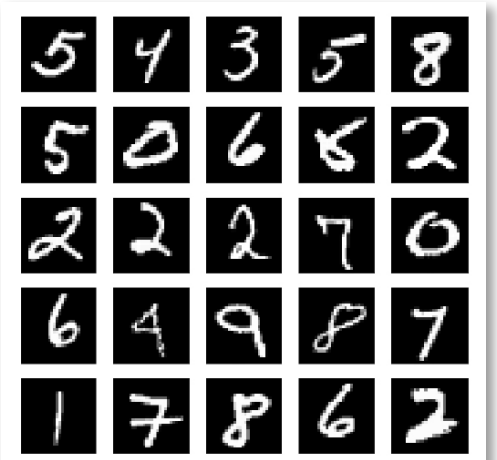
  사용할 데이터셋은 다양한 구조의 신경망을 실험할때 자주 사용되니 적극적으로 활용하기를 바랍니다.

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 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

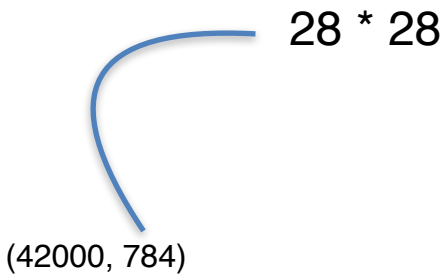
```
01 from sklearn.model_selection import train_test_split
02
03 # 훈련/테스트 데이터를 0.7/0.3의 비율로 분리합니다.
04 x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
05                                                  test_size = 0.3,
06                                                  random_state = 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}')
```

훈련 데이터 (42000, 28, 28) 레이블 (42000,)
훈련 데이터 (18000, 28, 28) 레이블 (18000,)

- 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

여러 가지 전처리 방법 – 스케일링

Normalization은 데이터의 범위를 사용자가 원하는 범위로
제한하는 것이다.

예를 들어 이미지 데이터의 경우 픽셀 정보를 0~255사이의
값으로 가지는데, 이를 255로 나누어주면

0.0~1.0사이의 값을 가지게 될 것이다.

이러한 행위를 feature들의 scale을 정규화(Normalization) 한다고 한다.

Normalization(MinMax)

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

MNIST 데이터셋

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

```
from keras.utils import to_categorical

print(y_train)
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)

print(y_train)
```

```
[2 7 6 ... 3 4 5]

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

MNIST 데이터셋

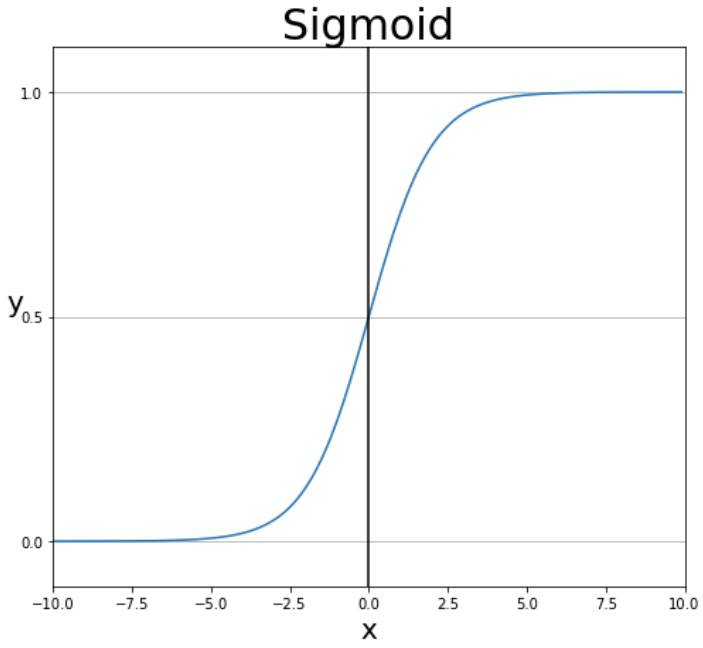
- 모델을 구성합니다.
  - 모델은 **784차원의 데이터를 입력(input)**으로 받고, **열 개의 출력(output)**을 가짐

[함께 해봐요] **모델 구성하기** mnist.ipynb

```
01 from keras.models import Sequential
02 from 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

- 시그모이드 함수
  - 시그모이드 함수는 0에서 1사이의 함수이며, 값이 들어왔을 때, 0~1 사이의 값을 반환하다.
  - 연속형 데이터이기 때문에 계단 함수가 끊기지 않는 매끄러운 모양으로 바뀐 것을 알 수 있다.
  - 동시에 이상치가 들어온다 할지라도, 시그모이드 함수는 0과 1에 수렴하므로, 이상치 문제도 해결하면서, 연속된 값을 전달할 수 있다.
  - 시그모이드 함수를 활성화 함수로 사용하면, 0과 1에 가까운 값을 통해 이진 분류를 할 수 있다.



이진 분류: sigmoid + binary_crossentropy

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:

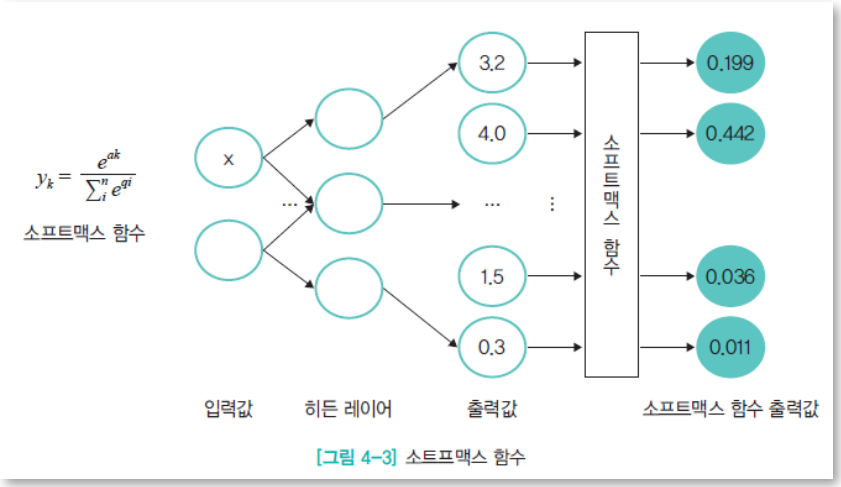
  - (불고기버거, 치즈버거, 치킨버거) = (3.1, 3.0, 2.9)
  - (불고기버거, 치즈버거, 치킨버거) = (2.0, 1.0, 0.7)

- 위 예시에서 위와 아래의 숫자 비교에서 어느 것이 더 명확한가요?

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

  - 제공되는 코드를 통해 값을 비교해보세요.

다중 분류: softmax + categorical_crossentropy



[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

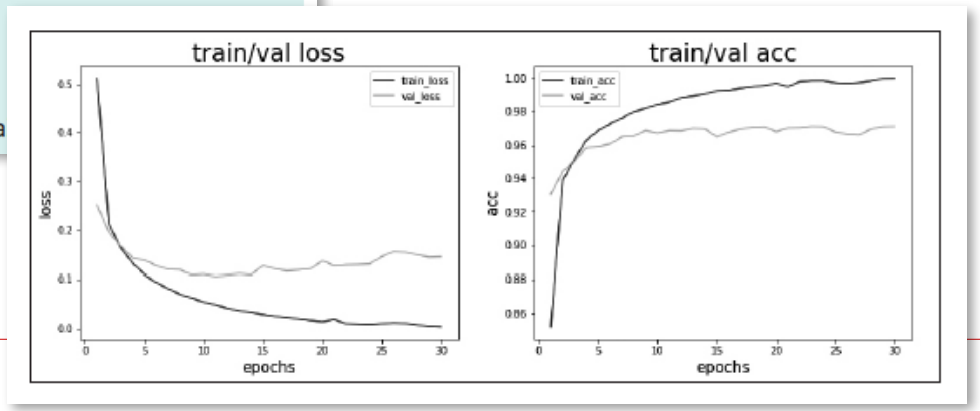
```
01 history = model.fit(x_train, y_train,
02                     epochs = 30,
03                     batch_size = 128,
04                     validation_data = (x_val, y_val))
```

```
loss: 0.0081 - acc: 0.9977 - val_loss: 0.1506 - val_acc: 0.9686
```

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

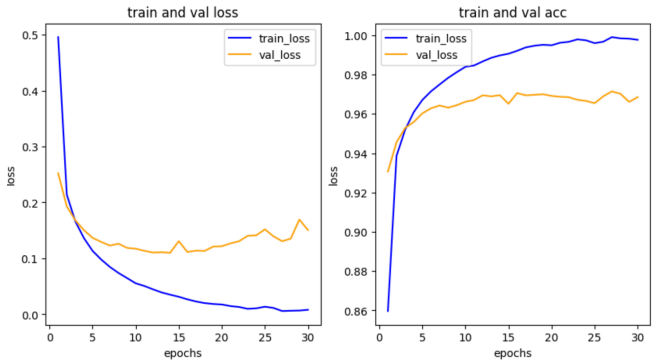
[함께 해봐요] 학습 결과 그려보기 — mnist.ipynb

```
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 경우 va
```



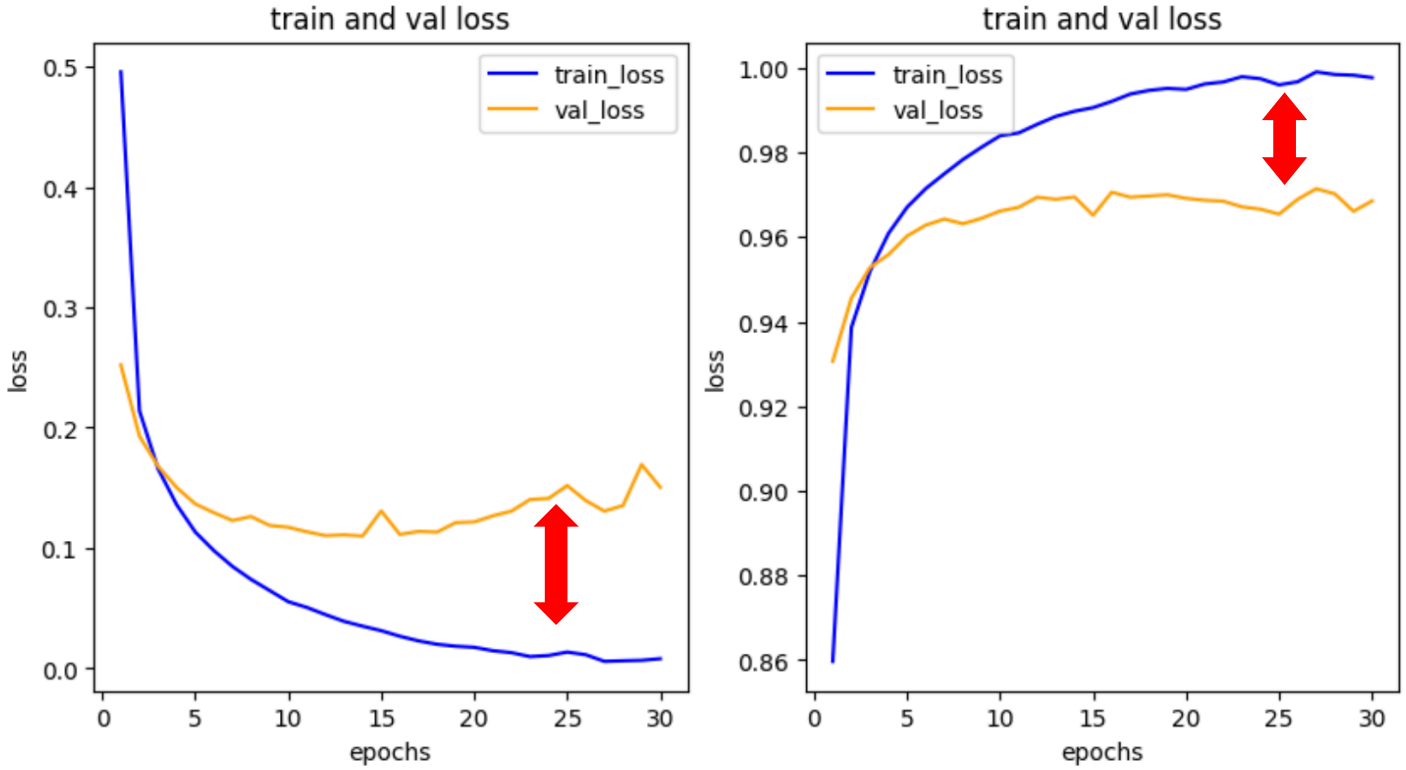
MNIST 데이터셋

```
import matplotlib.pyplot as plt

his_dict = history.history
loss = his_dict['loss']
val_loss = his_dict['val_loss']

epochs = range(1, len(loss)+1)
fig = plt.figure(figsize=(10,5))

ax1 = fig.add_subplot(1,2,1)
ax1.plot(epochs, loss, color='blue', label='train_loss')
ax1.plot(epochs, val_loss, color='orange', label='val_loss')
ax1.set_title('train and val loss')
ax1.set_xlabel('epochs')
ax1.set_ylabel('loss')
ax1.legend()

acc = his_dict['acc']
val_acc = his_dict['val_acc']

ax2 = fig.add_subplot(1,2,2)
ax2.plot(epochs, acc, color='blue', label='train_loss')
ax2.plot(epochs, val_acc, color='orange', label='val_loss')
ax2.set_title('train and val loss')
ax2.set_xlabel('epochs')
ax2.set_ylabel('loss')
ax2.legend()

plt.show()
```



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

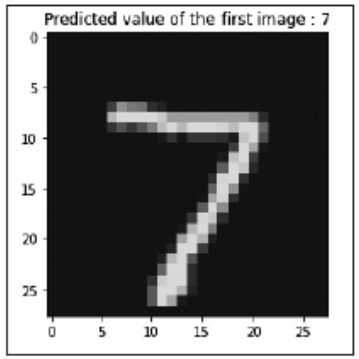
```
01  model.evaluate(x_test, y_test)
```

```
[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

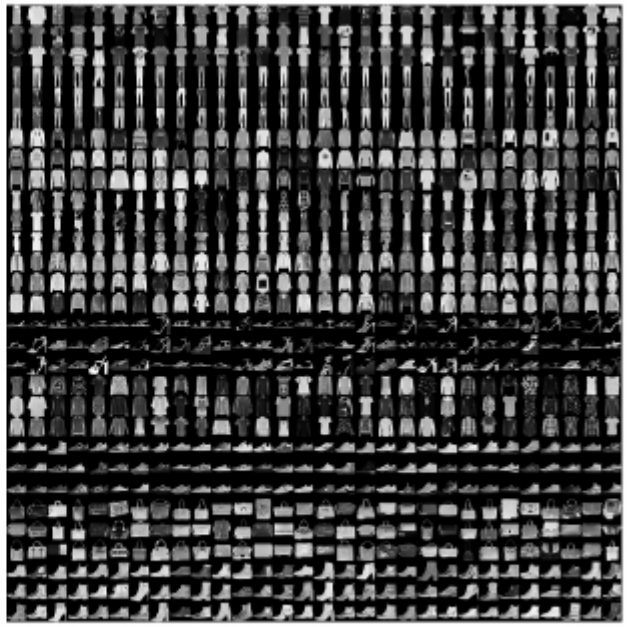
- 모델을 해석해보자
  - 혼동 행렬(Confusion Matrix)
  - 분류 보고서(Classification Report)

	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

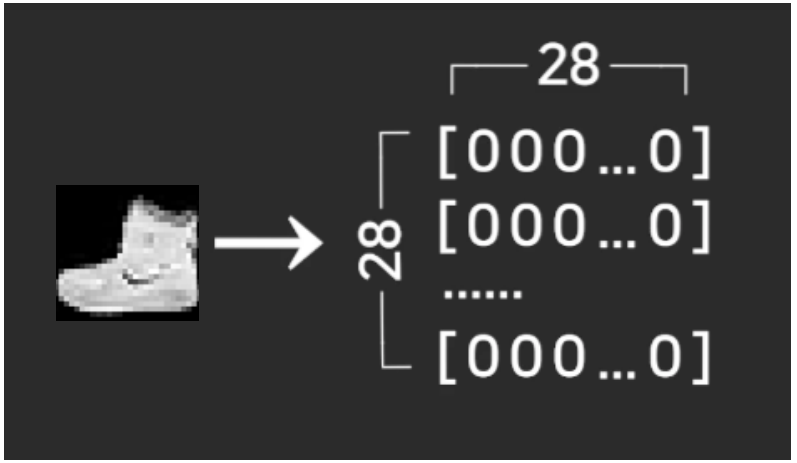


[그림 4-4] Fashion-MNIST 데이터셋(Zalando, MIT License)

Fashion-MNIST 데이터셋

```
from keras.datasets.fashion_mnist import load_data

# Fashion-MNIST 데이터를 다운받습니다.
(x_train, y_train), (x_test, y_test) = load_data()
print(x_train.shape, x_test.shape) #(60000, 28, 28) (10000, 28, 28)
```

```
print(x_train[0])

print(y_train[0]) # 9
```




```
'Ankle boot'
```

Fashion-MNIST 데이터셋

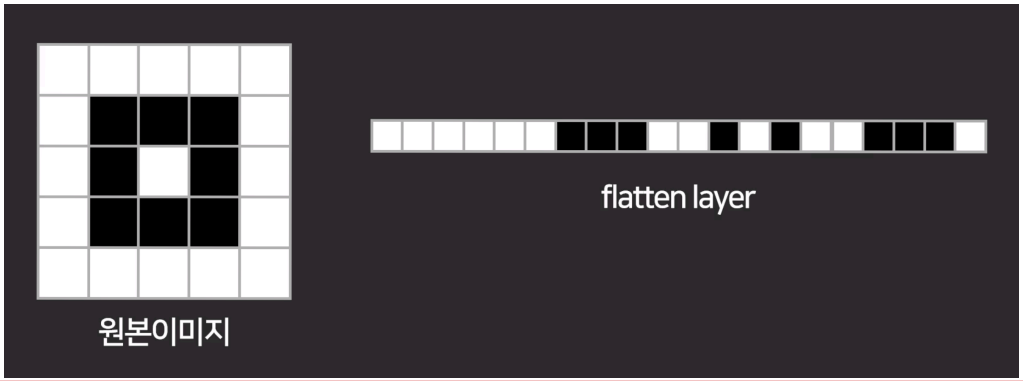
```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(777)

class_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

sample_size = 9

# 0~8 이미지 띄워보는법
for i in range(sample_size):
  plt.subplot(3,3,i+1) #3x3 격자의 서브플롯을 생성
  plt.xticks([]) # x축 눈금 제거
  plt.yticks([]) # y축 눈금 제거
  plt.imshow(x_train[i], cmap='gray')
  plt.xlabel(class_names[y_train[i]])
plt.show()
```

Fashion-MNIST 데이터셋

```
x_train = x_train/255
x_test = x_test /255

from keras.utils import to_categorical

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

from sklearn.model_selection import train_test_split

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

```
from keras.models import Sequential
from keras.layers import Dense, Flatten

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'))

first_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['acc'])
first_history = first_model.fit(x_train, y_train, epochs=30, batch_size=128,
validation_data=(x_val, y_val))
```



Fashion-MNIST 데이터셋

```
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'))

second_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['acc'])
second_history = second_model.fit(x_train, y_train, epochs=30, batch_size=128,
validation_data=(x_val, y_val))
```

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

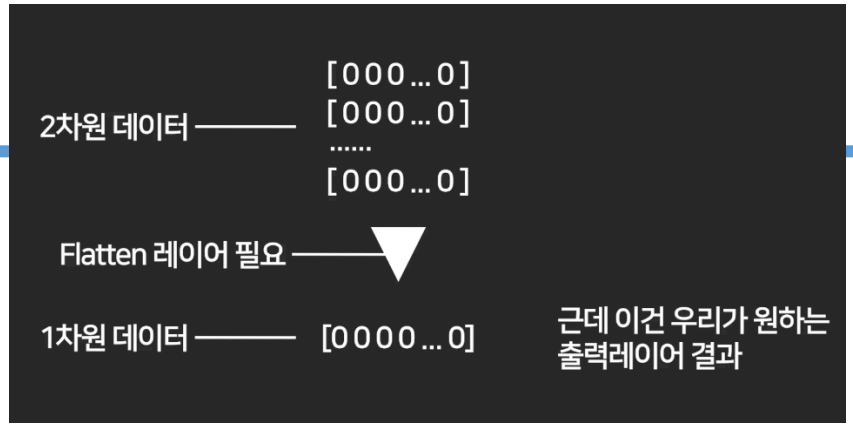
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

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'))
```

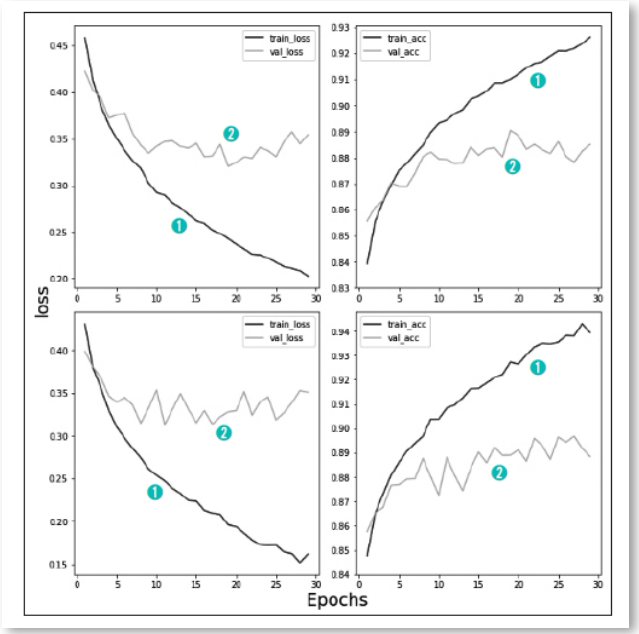
```
model.compile(optimizer='adam', loss = 'categorical_crossentropy',metrics=['acc'])
```

Fashion-MNIST 데이터셋

```
import numpy as np
import matplotlib.pyplot as plt

def draw_loss_acc(history_1, history_2, epochs):
  his_dict_1 = history_1.history
  his_dict_2 = history_2.history
  keys = list(his_dict_1.keys())

  epochs = range(1, epochs)
  fig = plt.figure(figsize=(10,10))
  ax = fig.add_subplot(1,1,1)

  ax.spines['top'].set_color('none')
  ax.spines['bottom'].set_color('none')
  ax.spines['left'].set_color('none')
  ax.spines['right'].set_color('none')
  ax.tick_params(labelcolor='w', top=False, bottom=False, left=False, right=False)

  for i in range(len(his_dict_1)):
    temp_ax = fig.add_subplot(2,2,i+1)
    temp = keys[i%2]
    val_temp = keys[(i+2)%2 + 2]
    temp_history = his_dict_1 if i<2 else his_dict_2
    temp_ax.plot(epochs, temp_history[temp][1:], color='blue', label='train_' + temp)
    temp_ax.plot(epochs, temp_history[val_temp][1:], color='orange', label=val_temp)

    if(i==1 or i==3):
      start, end = temp_ax.get_ylim()
      temp_ax.yaxis.set_ticks(np.arange(np.round(start, 2), end, 0.01))
    temp_ax.legend()
  ax.set_ylabel('loss', size=20)
  ax.set_xlabel('Epochs', size=20)
  plt.tight_layout()
  plt.show()

draw_loss_acc(first_history, second_history, 30)
```

Fashion-MNIST 데이터셋 – End

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



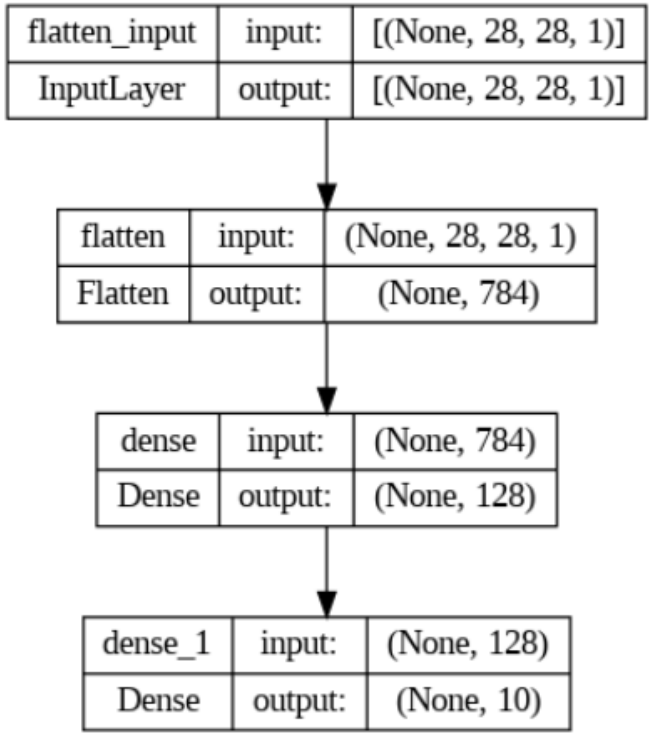
- 결과 해석
  - **모델을 깊게 구성 → 높은 성능, But 과대적합(파라미터 수 증가)**
  - 모델의 깊이는 데이터에 적합하게 결정해야 함
  - 유명한 데이터셋이나 유사 분야에서 높은 성능을 보여준 모델 구조를 참고하여 구성해보고 실험을 진행

모델 구조를 확인하는 방법

- summary()

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

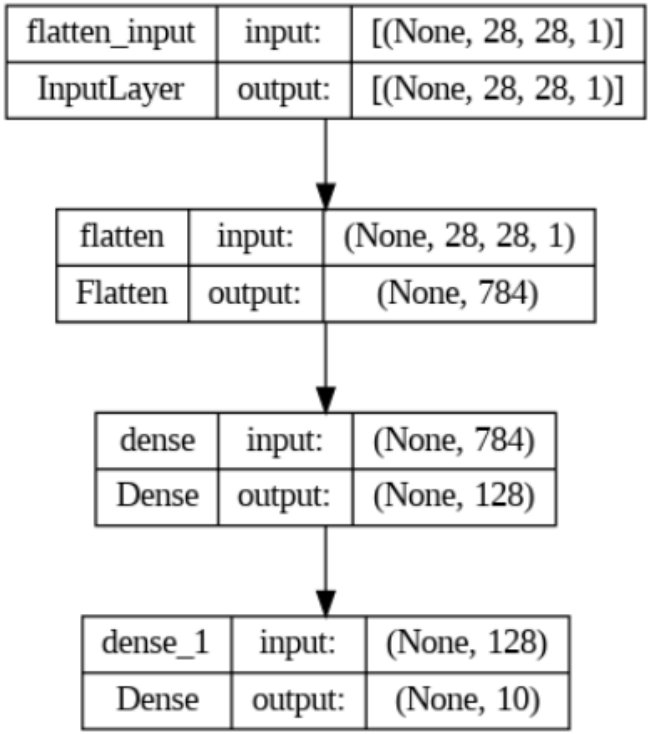



```
from keras.utils import plot_model

plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```

모델 구조를 확인하는 방법

- plot_model() 함수

```
from keras.utils import plot_model

plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```



RECAP

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

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.