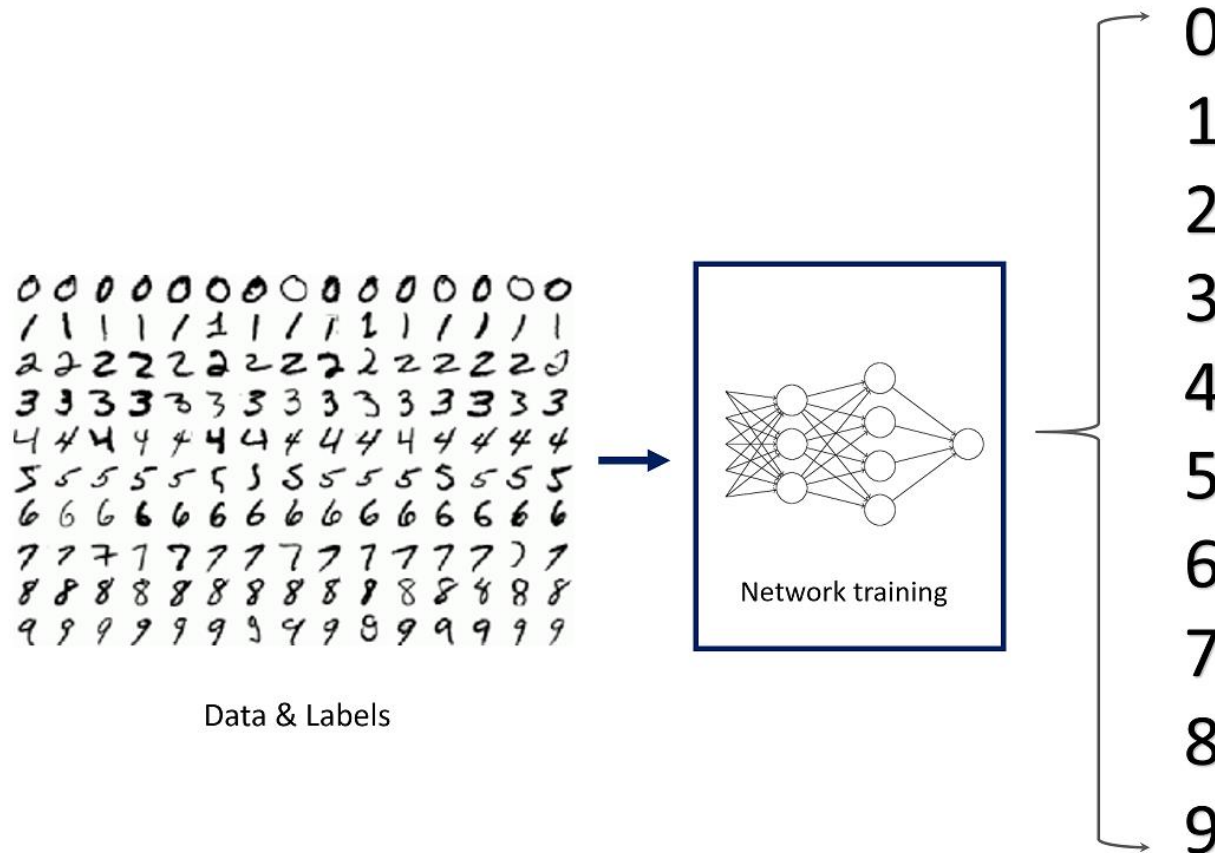


딥러닝의 Hello World
MNIST 이해를 위한
손글씨 보기

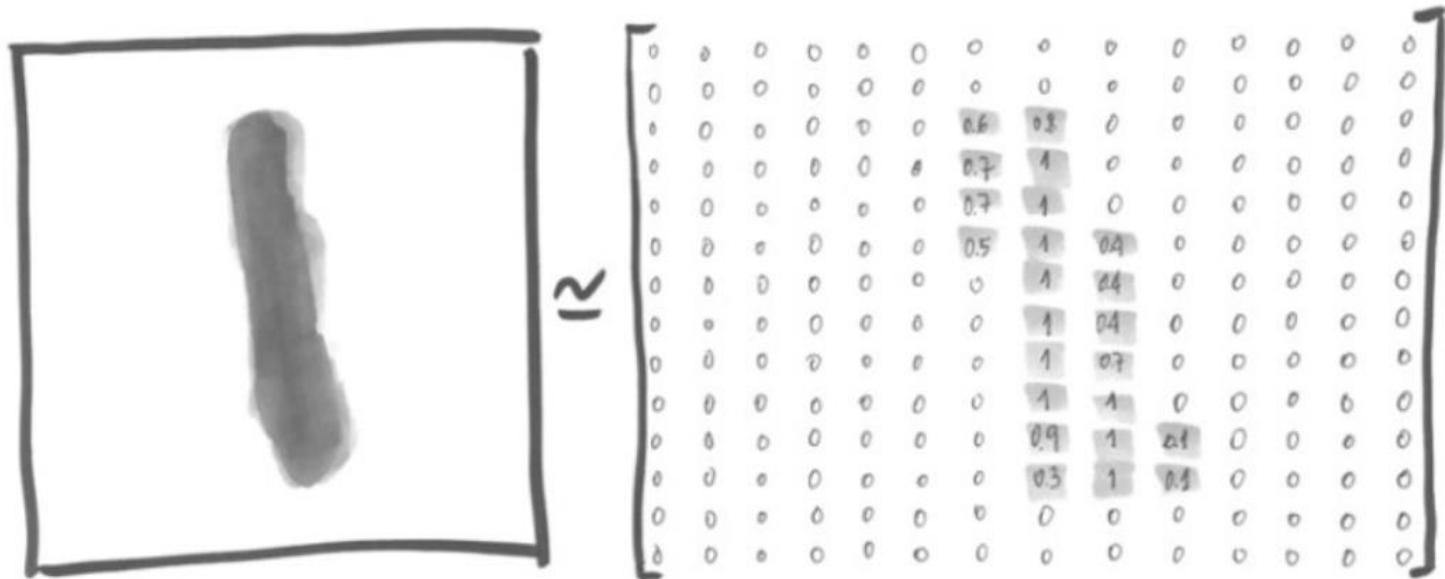
Hello World

- MNIST 데이터셋
 - 딥러닝 손글씨 인식에 사용되는 데이터셋



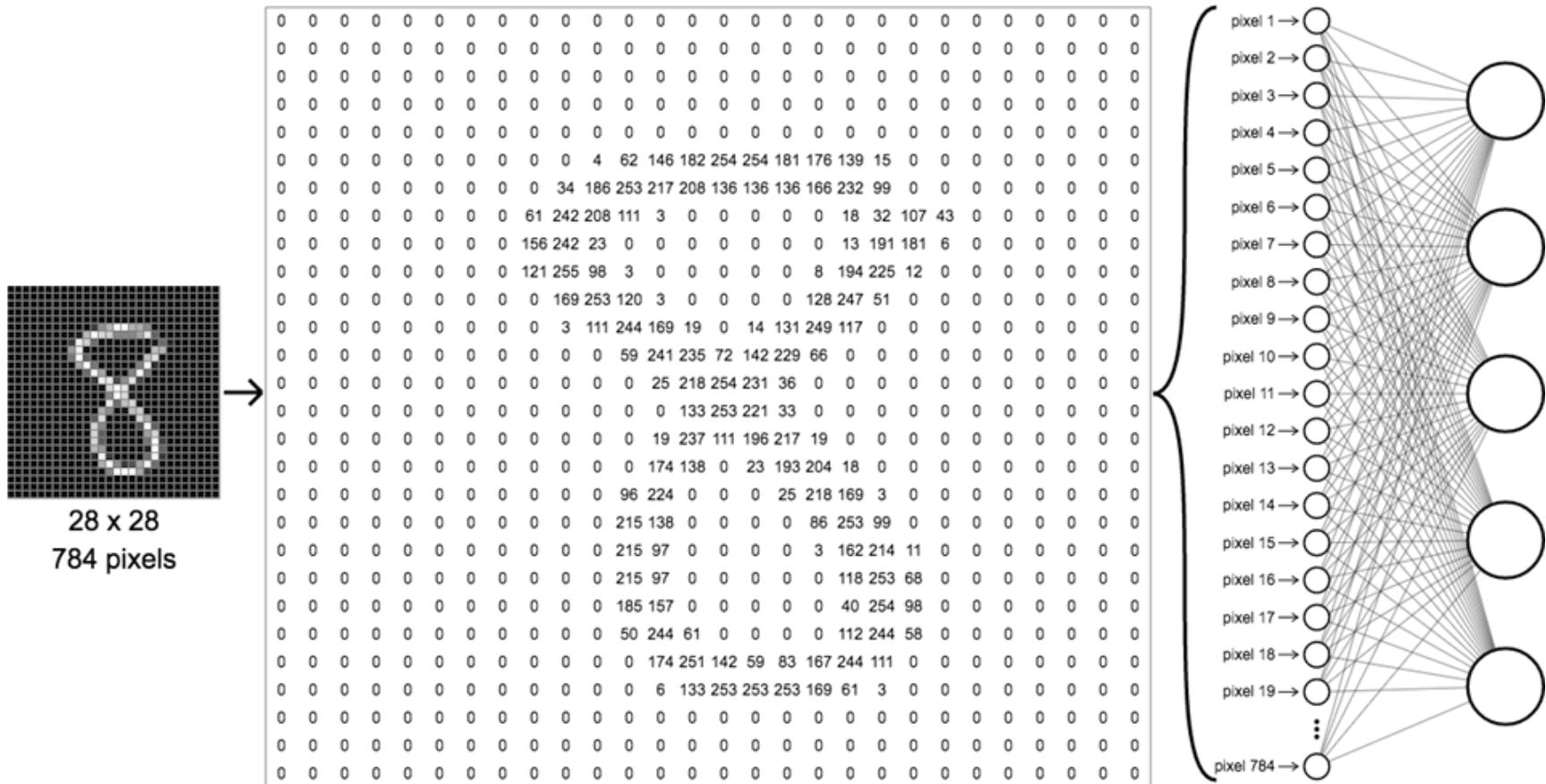
손글씨 하나의 구조

- 784 픽셀
 - 28 X 28
 - 내부 값은 0~255
 - 이 값을 0~1로 수정해서 사용



MNIST 이미지와 딥러닝

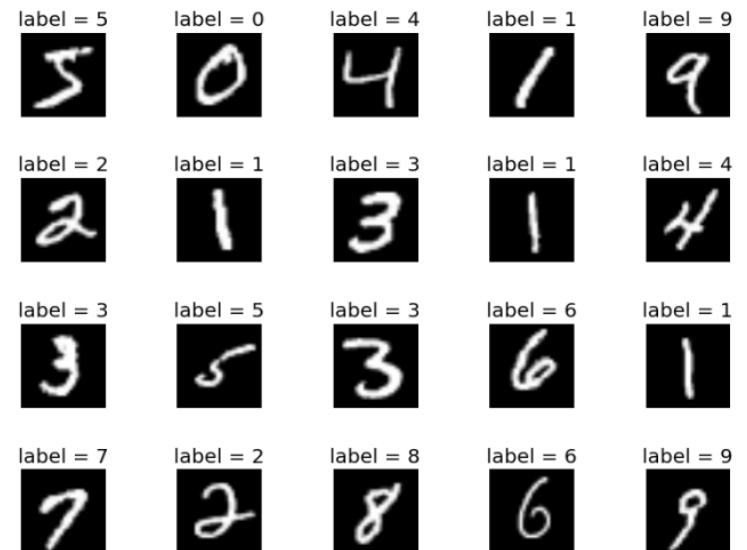
- 흑백 28 * 28



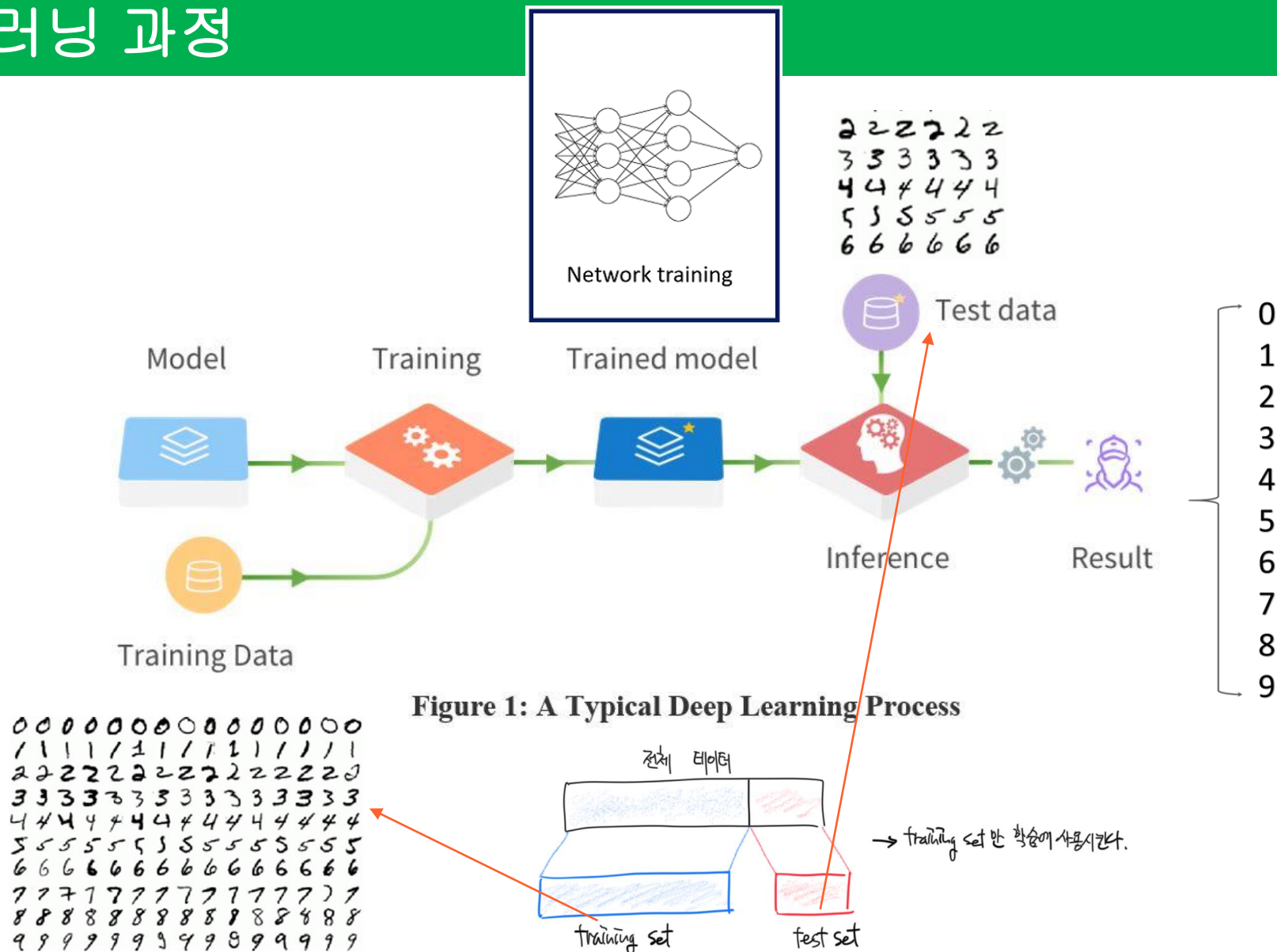
MNIST 데이터 셋

• MNIST(Modified National Institute of Standards and Technology)

- 손으로 쓴 자릿수에 대한 데이터 집합
 - Yann Lecun의 The MNIST DATABASE of handwritten numerics 웹 사이트에서 배포
 - <http://yann.lecun.com/exdb/mnist/>
- "필기 숫자 이미지"와 정답인 "레이블"의 쌍으로 구성
 - 숫자의 범위는 0에서 9까지, 총 10 개의 패턴을 의미
- 필기 숫자 이미지
 - 크기가 28 x 28 픽셀인 회색조 이미지
- Label
 - 이미지의 정답: 필기 숫자 이미지가 나타내는 실제 숫자, 0에서 9
- 대표적인 두 가지의 데이터 가져오는 방법
 - `tensorflow.keras.datasets.mnist`
 - `tensorflow.examples.tutorials.mnist`



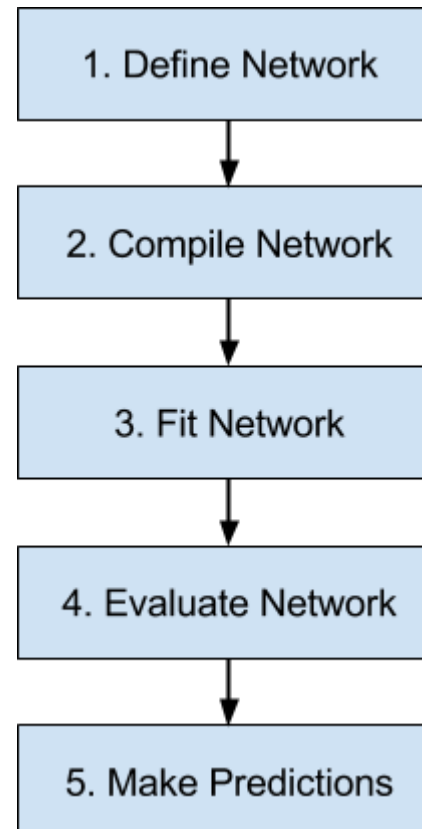
딥러닝 과정



케라스 딥러닝 구현

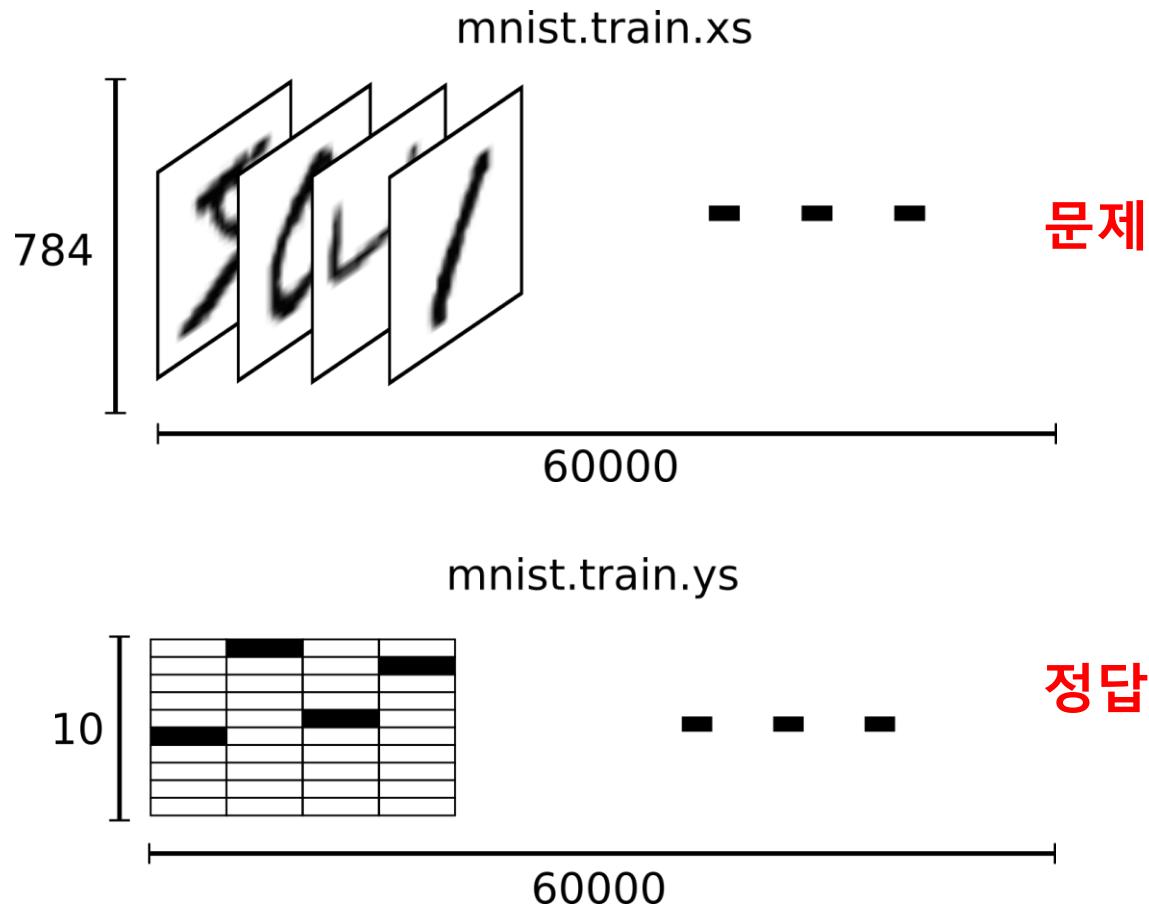
• 5개 과정

- 딥러닝을 모델을 만들어
 - **define**
- 주요 훈련방법을 설정하고
 - **compile**
 - 최적화 방법
 - 손실 함수
- 훈련시켜
 - **fit**
- 테스트 데이터를 평가하고
 - **evaluate**
- 정답을 예측
 - **predict**



Mnist 데이터

- **훈련 데이터 구조**
 - 총 6만개의 손글씨



MNIST 손글씨 데이터 로드 코드

- **훈련 데이터 손글씨와 정답**
 - x_train, y_train
 - 6만개
- **테스트 데이터 손글씨와 정답**
 - x_test, y_test
 - 1만개

```
import tensorflow as tf
```

```
mnist = tf.keras.datasets.mnist
```

```
# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

MNIST 손글씨 데이터 구조 확인

• 첫 손글씨와 데이터와 정답

– x_train[0], y_train[0]

```
[62] 1 x_train.shape
```

```
↳ (60000, 28, 28)
```

```
[68] 1 y_train.shape
```

```
↳ (60000,)
```

```
[70] 1 x_train[0]
```

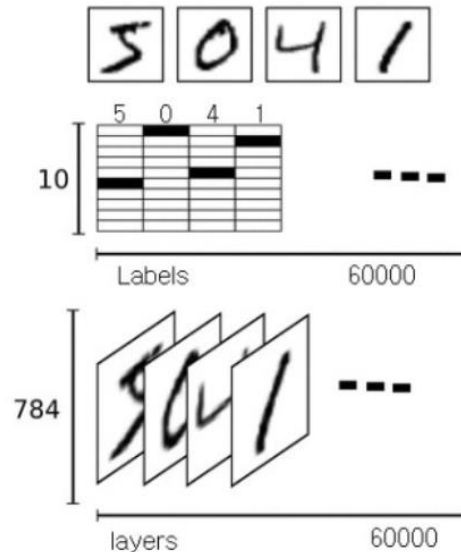
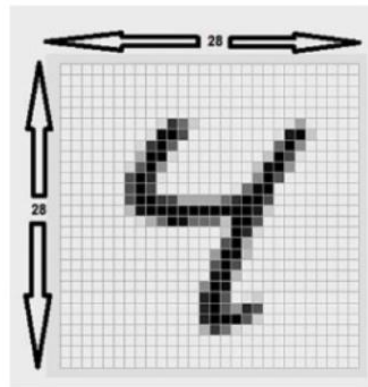
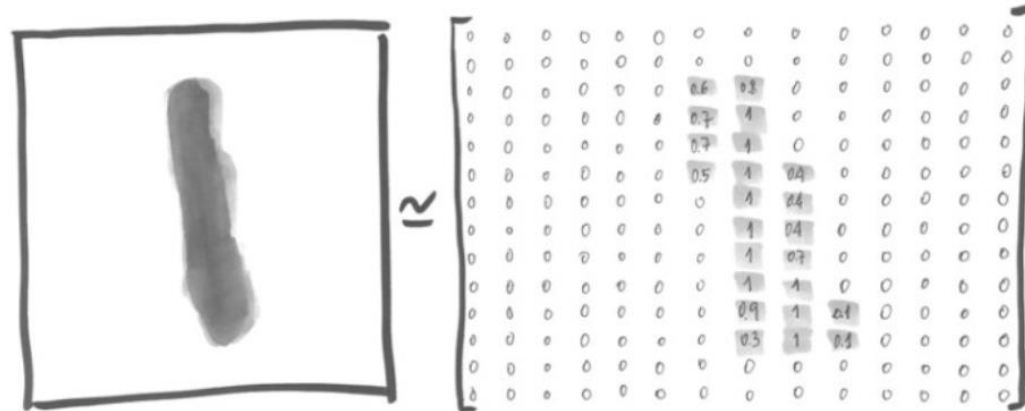
```
↳
[[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  3,
 18, 18, 18, 126, 136, 175, 26, 166, 255, 247, 127,  0,  0,
  0,  0],
 [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  30, 36, 94, 154, 170,
 253, 253, 253, 253, 253, 225, 172, 253, 242, 195, 64,  0,  0,
  0,  0],
 [ 0,  0,  0,  0,  0,  0,  0,  49, 238, 253, 253, 253, 253,
 253, 253, 253, 253, 251, 93, 82, 82, 56, 39,  0,  0,  0,
  0,  0],
 -
 0,  0],
```

```
[71] 1 y_train[0]
```

```
↳ 5
```

MNIST 데이터 기본 확인

- 배열 구조
 - 속성 shape



MNIST 손글씨 데이터 구조 확인 코드

```
# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
# MNIST 형태를 알아 봅시다. 데이터 수, 행렬 형태 등
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
# MNIST 훈련 데이터의 내부 첫 내용도 알아보자.
print(x_train[0])
print(y_train[0])
```

```
# MNIST 테스트 데이터의 내부 첫 내용도 알아보자.
print(x_test[0])
print(y_test[0])
```

이차원 28 x 28

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

```
[[...]]
5
```

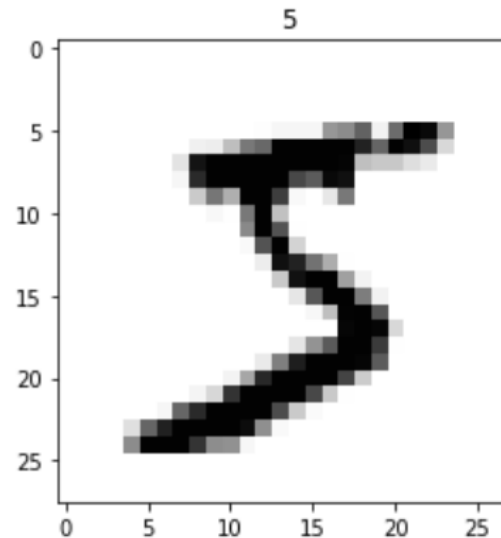
```
[[...]]
7
```

훈련 데이터 첫 손글씨 보기

- `x_train[0], y_train[0]`
 - 손글씨 5

```
[67] 1 n = 0  
      2 ttl = str(y_train[n])  
      3 plt.figure(figsize=(6, 4))  
      4 plt.title(ttl)  
      5 plt.imshow(x_train[n], cmap='Greys')
```

↳ <matplotlib.image.AxesImage at 0x7faf8ba0bda0>

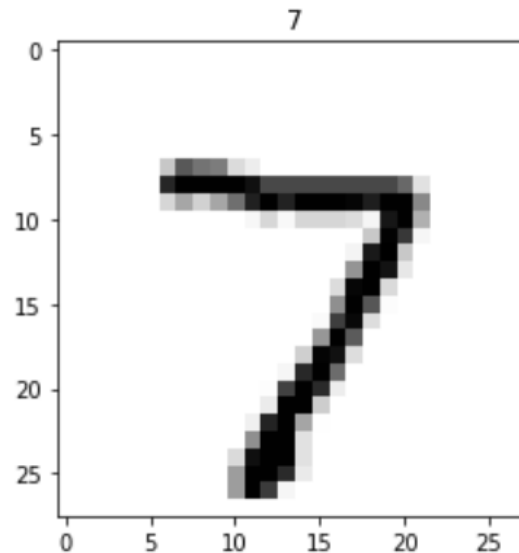


테스트 데이터 첫 손글씨 보기

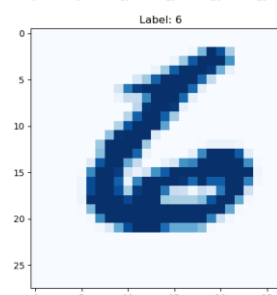
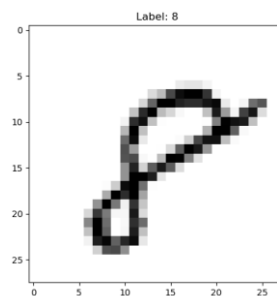
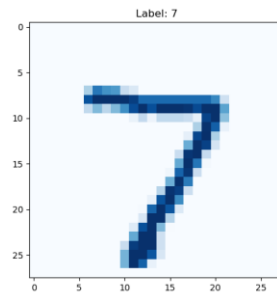
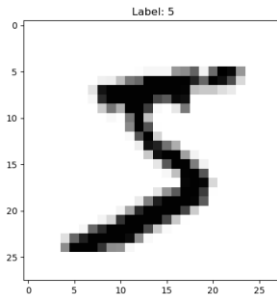
- `x_test[0], y_test[0]`
 - 손글씨 7

```
[74] 1 n = 0  
      2 ttl = str(y_test[n])  
      3 plt.figure(figsize=(6, 4))  
      4 plt.title(ttl)  
      5 plt.imshow(x_test[n], cmap='Greys')
```

☞ <matplotlib.image.AxesImage at 0x7faf8c402be0>



첫 손글씨와 마지막 손글씨 그려 보기



MNIST 데이터(훈련, 테스트)의 내부 첫 내용을 그려보자.
`import matplotlib.pyplot as plt`

```
tmp = "Label: " + str(y_train[0])
plt.title(tmp)
plt.imshow(x_train[0], cmap="Greys")
plt.show()
```

```
tmp = "Label: " + str(y_test[0])
plt.title(tmp)
plt.imshow(x_test[0], cmap='Blues')
plt.show()
```

MNIST 데이터(훈련, 테스트)의 내부 마지막 내용을 그려보자.

```
idx = len(x_train) - 1
tmp = "Label: " + str(y_train[idx])
plt.title(tmp)
plt.imshow(x_train[idx], cmap="Greys")
plt.show()
```

```
idx = len(x_test) - 1
tmp = "Label: " + str(y_test[idx])
plt.title(tmp)
plt.imshow(x_test[idx], cmap='Blues')
plt.show()
```

훈련용 데이터 60000 개 중에서 임의 손글씨 출력

- 0~59999 중의 임의 수 20개 선택

- 선택할 번호 선정

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
from random import sample
nrows, ncols = 4, 5 #출력 가로 세로 수
```

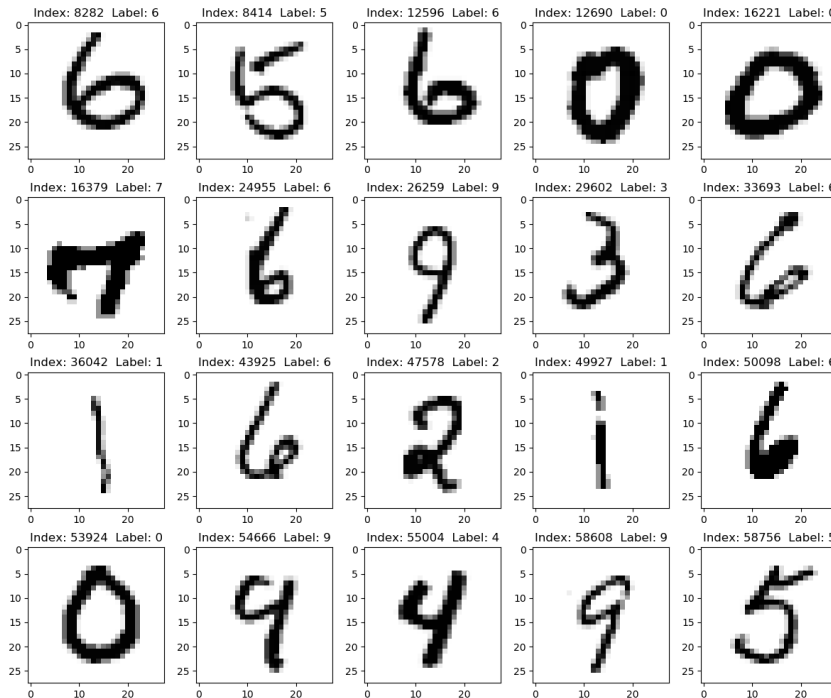
```
# 출력할 첨자 선정
```

```
idx = sorted(sample(range(len(x_train)), nrows * ncols))
print(idx)
```

```
[577, 4733, 6096, 7075, 15445, 15592, 22448, 22721, 23361, ... ]
```


손글씨 그려 보기

• 랜덤하게 20개



#랜덤하게 20개의 훈련용 자료를 그려 보자.

```
from random import sample
```

```
nrows, ncols = 4, 5 #출력 가로 세로 수
```

```
# 출력할 첨자 선정
```

```
idx = sorted(sample(range(len(x_train)), nrows *  
ncols))
```

```
#print(idx)
```

```
count = 0
```

```
plt.figure(figsize=(12, 10))
```

```
for n in idx:
```

```
    count += 1
```

```
    plt.subplot(nrows, ncols, count)
```

```
    tmp = "Index: " + str(n) + " Label: " +
```

```
str(y_train[n])
```

```
    plt.title(tmp)
```

```
    plt.imshow(x_train[n], cmap='Greys')
```

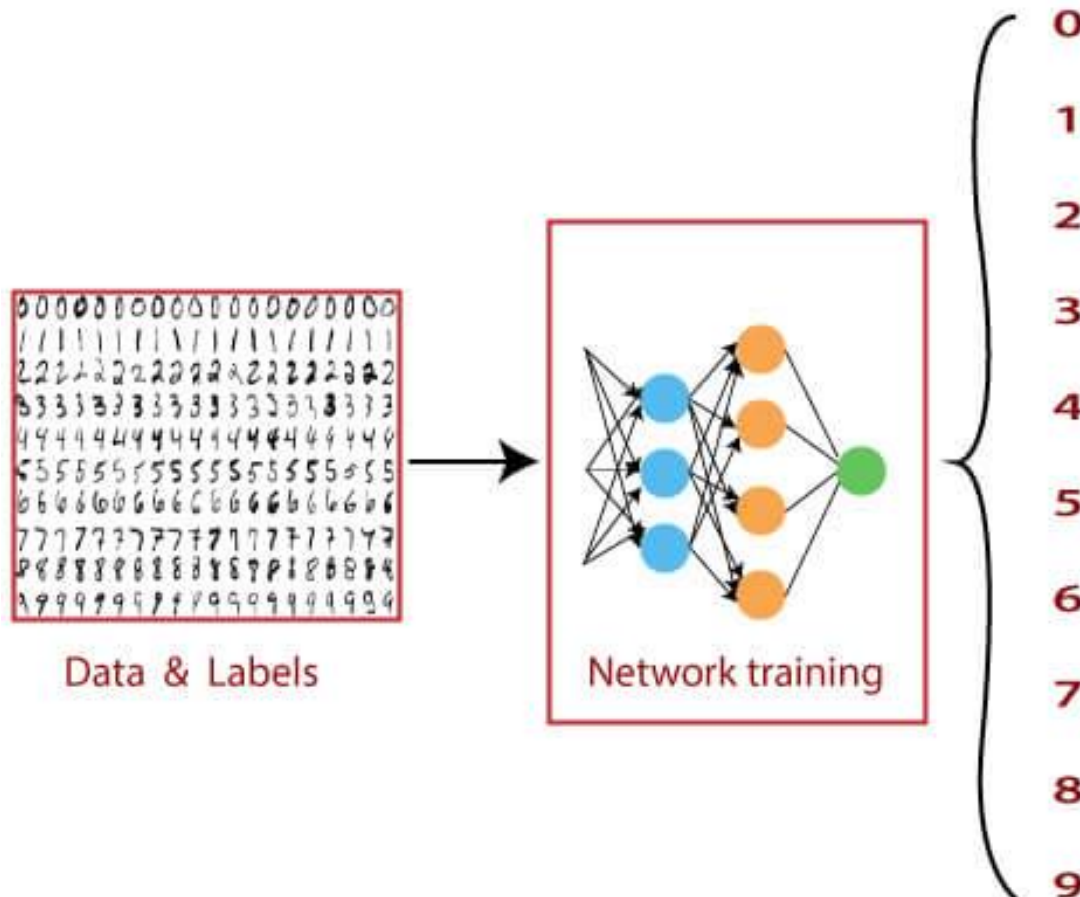
```
plt.tight_layout()
```

```
plt.show()
```

MNIST 데이터 딥러닝 모델 적용 예측

MNIST 데이터 셋을 위한 딥러닝

- 0에서 9까지의 분류
 - Number of classes: 10



딥러닝 과정

- **모델 구성(개발)**
 - 블랙 박스
- **모델 훈련: train**
 - 모델이 문제를 해결하도록 훈련
 - 어린 아이가 부모에게 훈련 받는 것에 비유
 - 다양한 훈육 방법
 - 경사하강법(내리막 경사 따라 가기)
 - 손실 함수(Loss Function)
- **예측: inference, prediction**

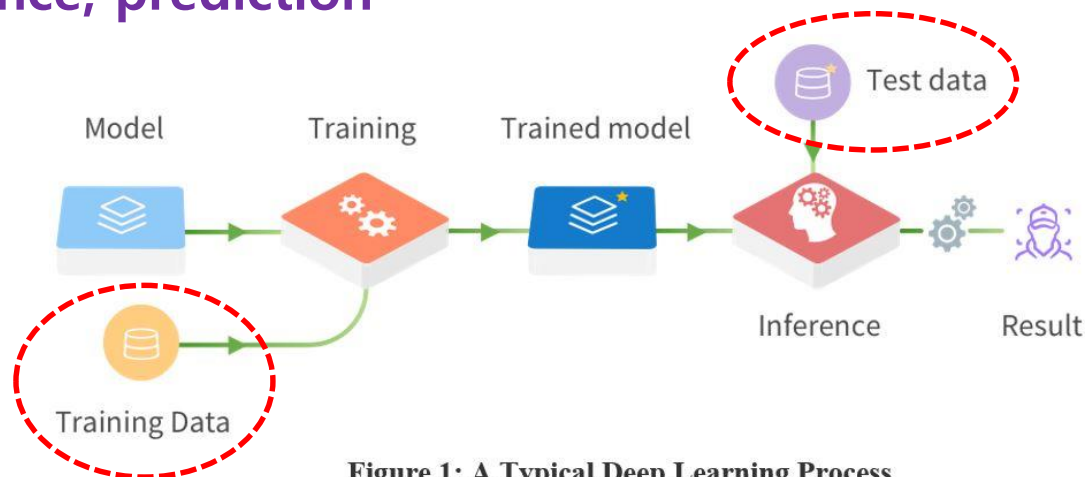
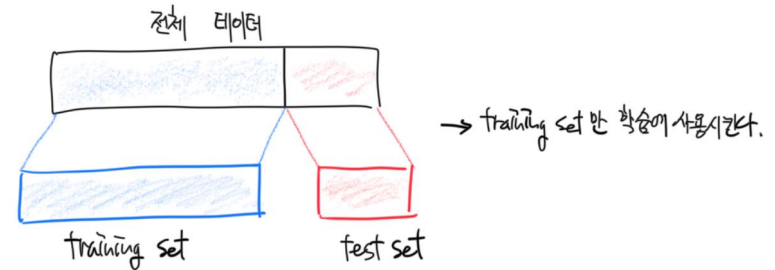
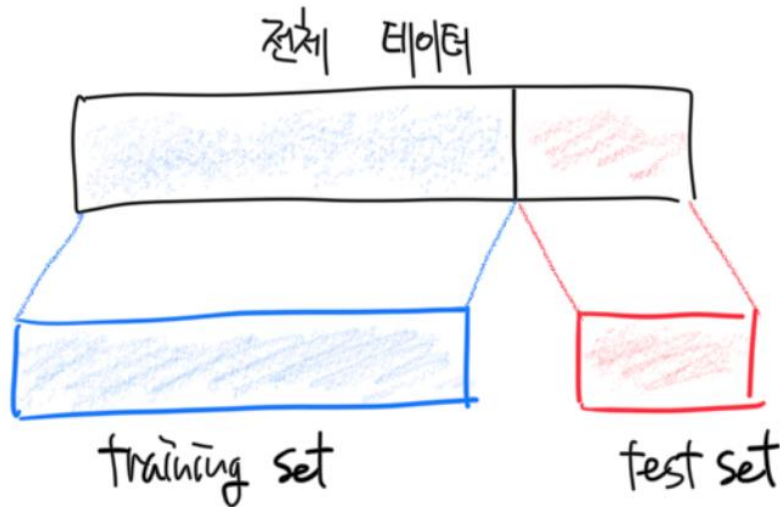


Figure 1: A Typical Deep Learning Process

훈련 데이터와 테스트 데이터로 구분

- MNIST에는 총 70000 개의 데이터
 - 훈련 데이터 세트(training data set) 크기: 60000
 - 테스트 데이터 세트(test data set) 크기: 10000



→ training set 만 학습에 사용시킨다.

딥러닝 구현 순서

- 0 필요 모듈 임포트
- ① 훈련과 정답 데이터 지정
 - ① - 1 데이터 전처리(옵션)
- ② 모델 구성
- ③ 학습에 필요한 최적화 방법과 손실 함수 등 설정
 - ③ - 1 구성된 모델 요약(옵션)
- ④ 생성된 모델로 훈련 데이터 학습
- ⑤ 테스트 데이터로 성능 평가
 - ⑤ - 1 테스트 데이터 또는 다른 데이터로 결과 예측(옵션)

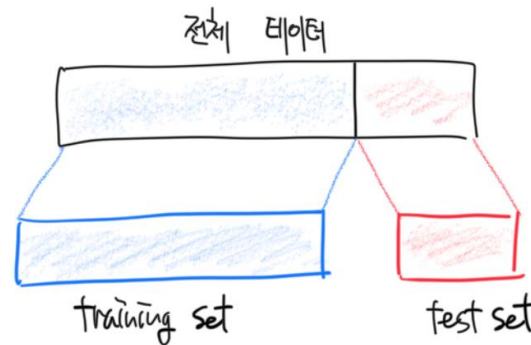
주요 용어

• 데이터셋

- 훈련용과 테스트용
 - Train data set, Test data set
 - x(입력, 문제), y(정답, 레이블)
- 전처리

• 모델

- 딥러닝 핵심 신경망, 여러 층 구성
 - 완전연결층
 - Dense()
 - 1차원 배열로 평탄화
 - Flatten()



• 학습 방법의 여러 요소들

- 옵티마이저(optimizer), 최적화 방법
 - 경사하강법: 내리막 경사 따라 가기
- 손실 함수(Loss Function)
 - Cross entropy(크로스엔트로피), MSE(Mean Square Error 평균제곱오차)

• 딥러닝 훈련

- Epochs
 - 총 훈련 횟수, 훈련 데이터를 한번 모두 훈련시키는 것이 1 에폭

① 훈련과 정답 데이터 지정

① - 1 데이터 전처리(옵션)

- MNIST 데이터셋을 로드하여 준비

- 전처리

- 샘플 값을 정수에서 부동소수로 변환

- 한 비트의 값을 255로 나눔

```
import tensorflow as tf
```

```
mnist = tf.keras.datasets.mnist
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
# 샘플 값을 정수 (0~255) 에서 부동소수 (0~1) 로 변환
```

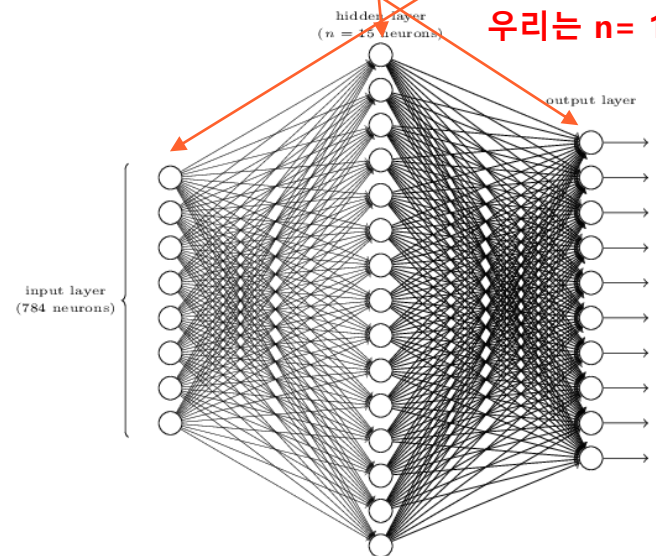
```
x_train, x_test = x_train / 255.0, x_test / 255.0
```


② 모델 구성

- 층을 차례대로 쌓아 `tf.keras.models.Sequential` 모델을 생성
 - 신경망 구성

층을 차례대로 쌓아 `tf.keras.models.Sequential` 모델을 생성

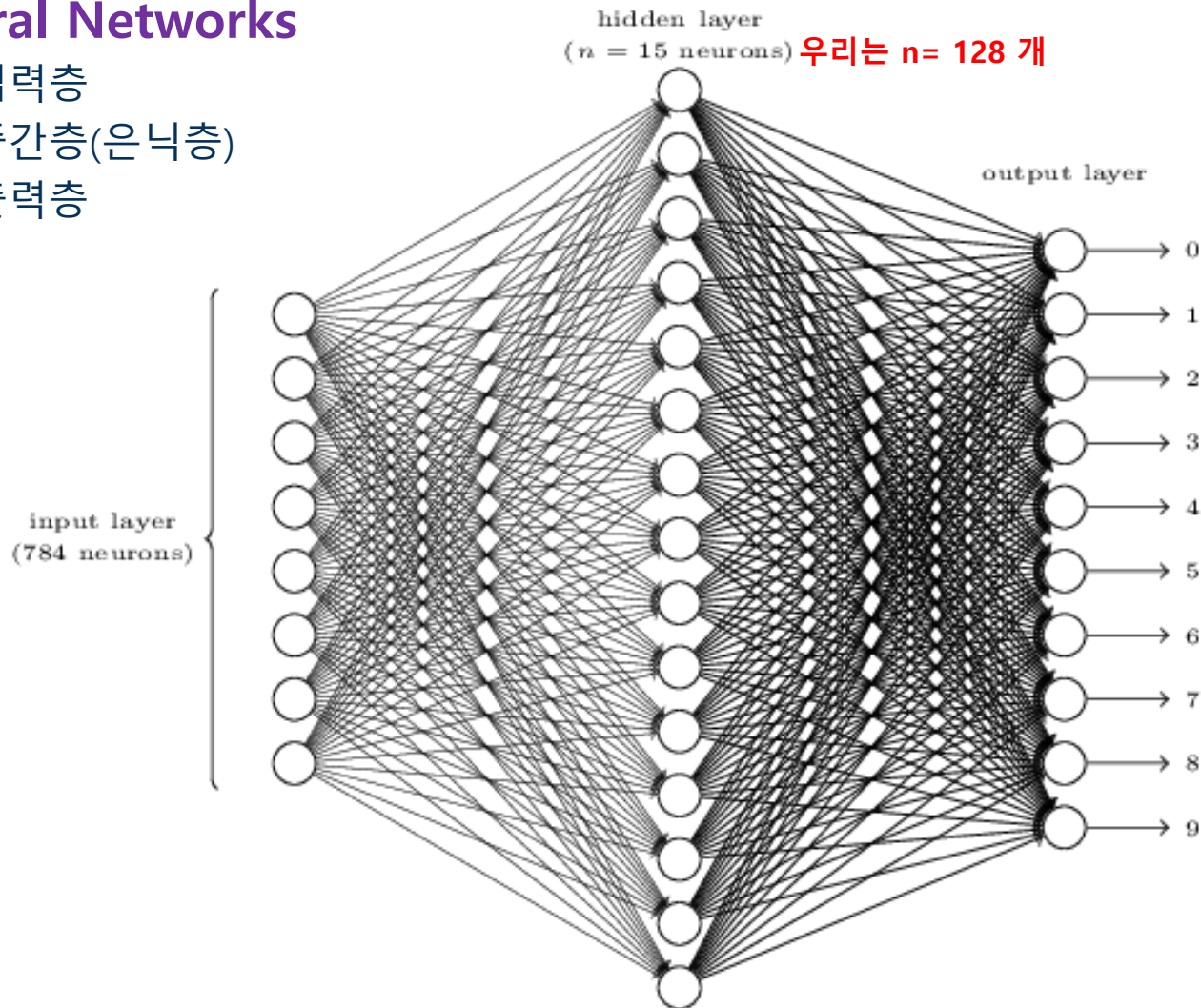
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



MNIST 해결을 위한 인공 신경망

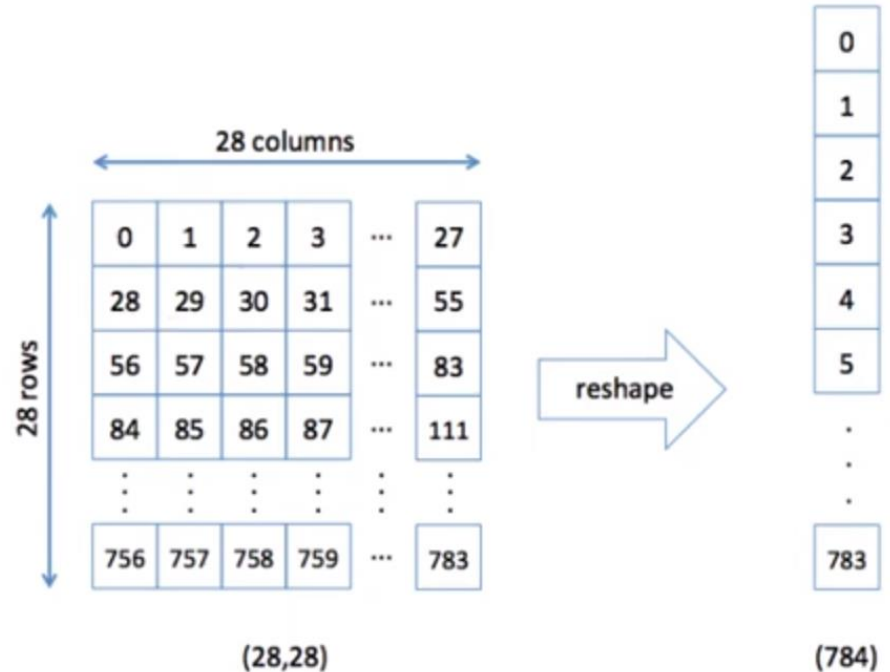
- **Neural Networks**

- 입력층
- 중간층(은닉층)
- 출력층



모델에서 2차원 그림을 1차원으로 평탄화

- **Flatten(input_shape=(28, 28)),**
 - 60000 개의 (28, 28) 크기를 가진 배열
 - 60000개의 (28 * 28) 크기의 배열로 수정

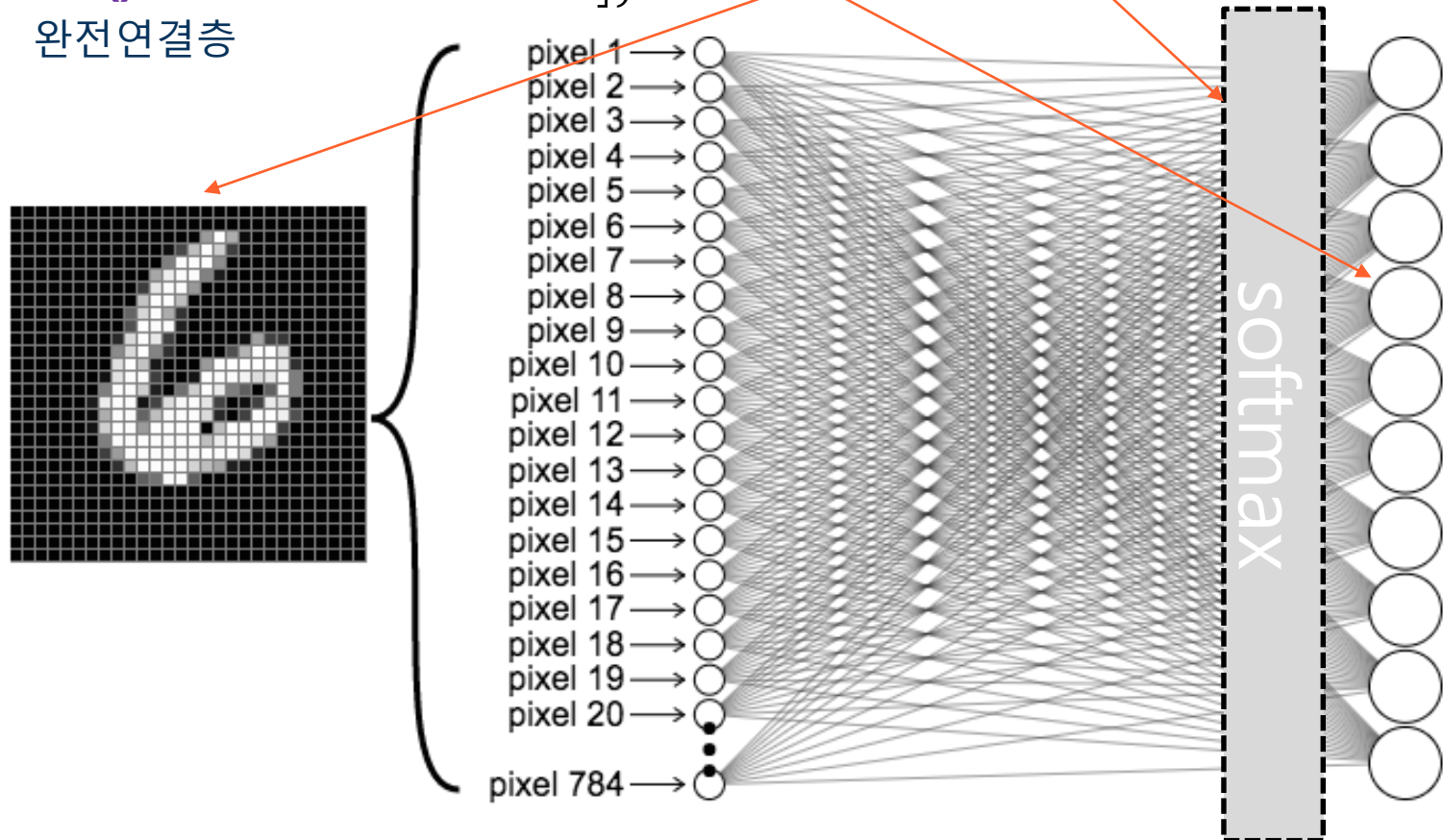


```
# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

단순 신경망 모델과 Dense() 층

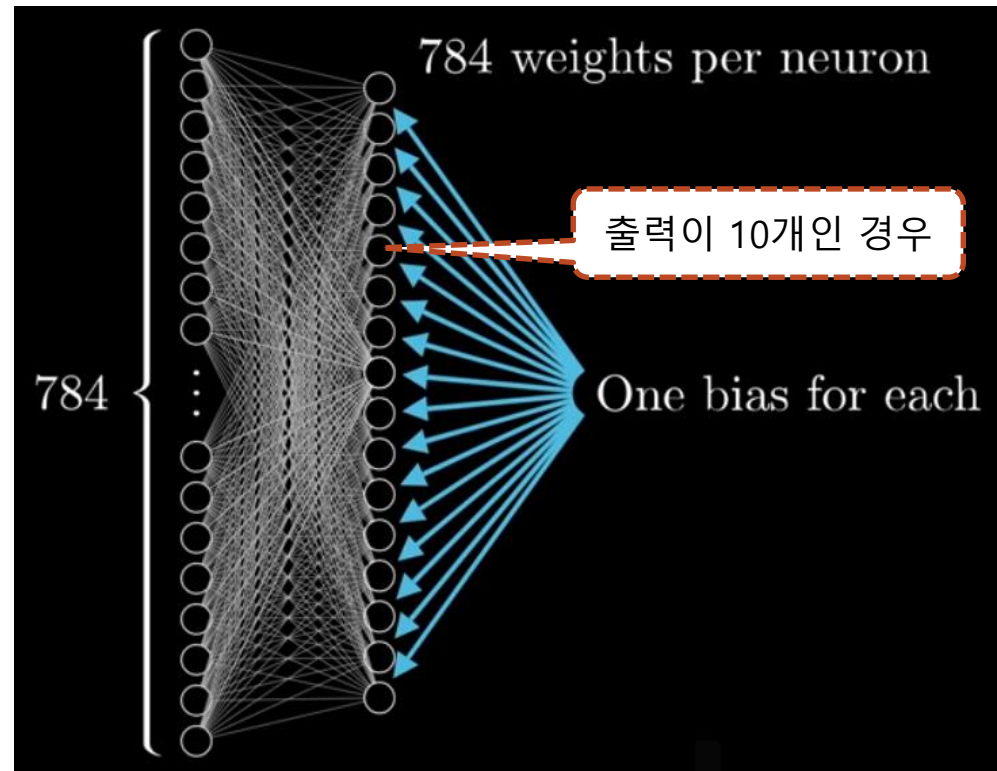
- **중간 은닉층이 없는 구조**
 - 입력층과 출력층만 존재
- **Dense()**
 - 완전연결층

```
# ④ 인공신경망 모델 생성(구성)
# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(10, activation='softmax')
])
```



중간층(은닉층)이 없는 경우

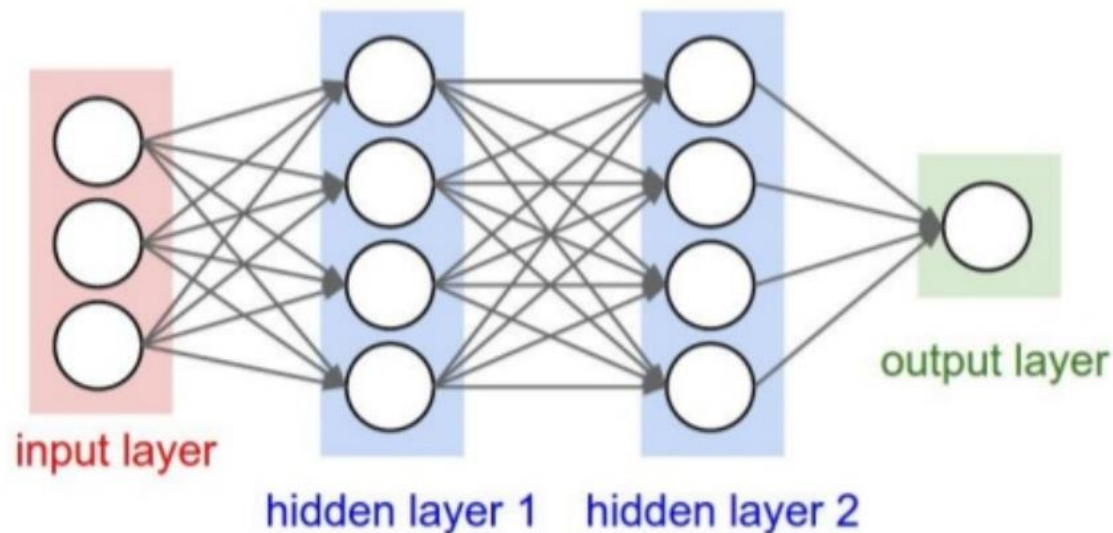
- **입력: 784**
 - 이미지의 각 픽셀 값
- **출력: 10**
 - 각 위치의 값이 될 크기
 - [1.22, .67, .45, .46, .86
.87, .45, .65, 1.14, 2.56]



Dense 층 의미

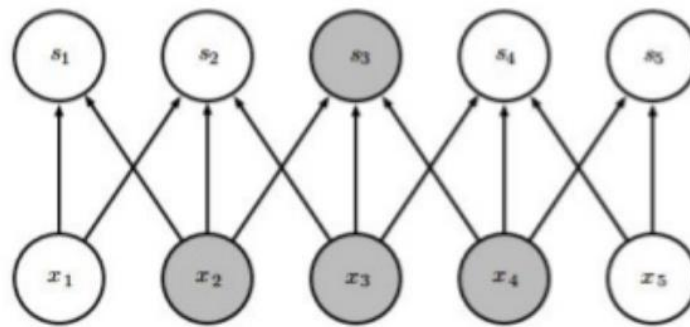
Fully Connected(Dense) Neural Network

- Typical 3-layer fully connected neural network

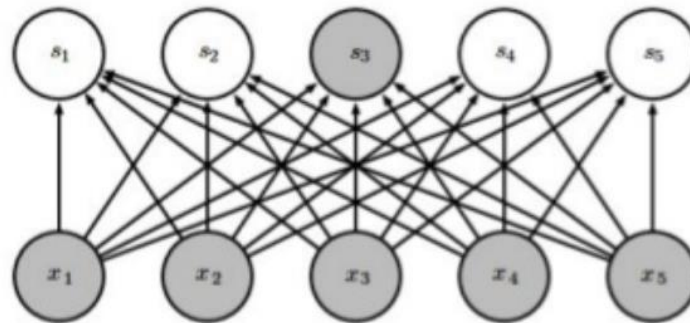


전체 연결과 부분 연결

Sparse connectivity vs. Dense connectivity



Sparse

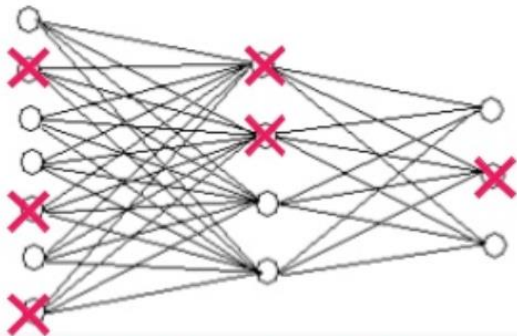


Dense

드롭아웃

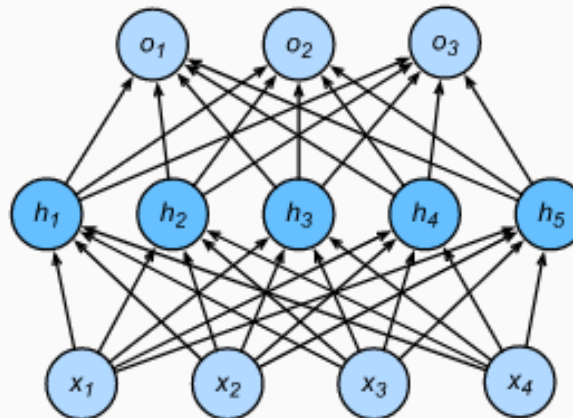
- `tf.keras.layers.Dropout(0.2)`
 - 훈련 중에 20%를 중간에 끊음

Dropout

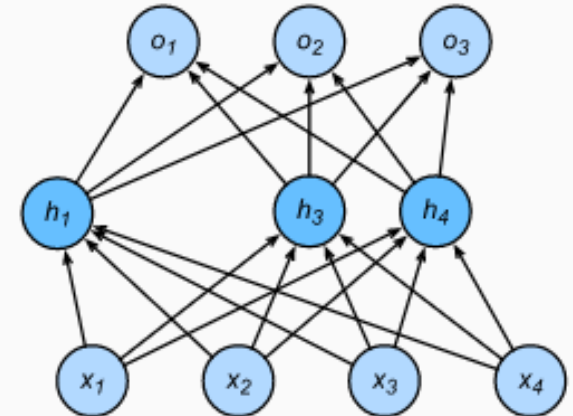


랜덤하게 뉴런을 끊음으로써,
모델을 단순하게 만든다.

MLP with one hidden layer



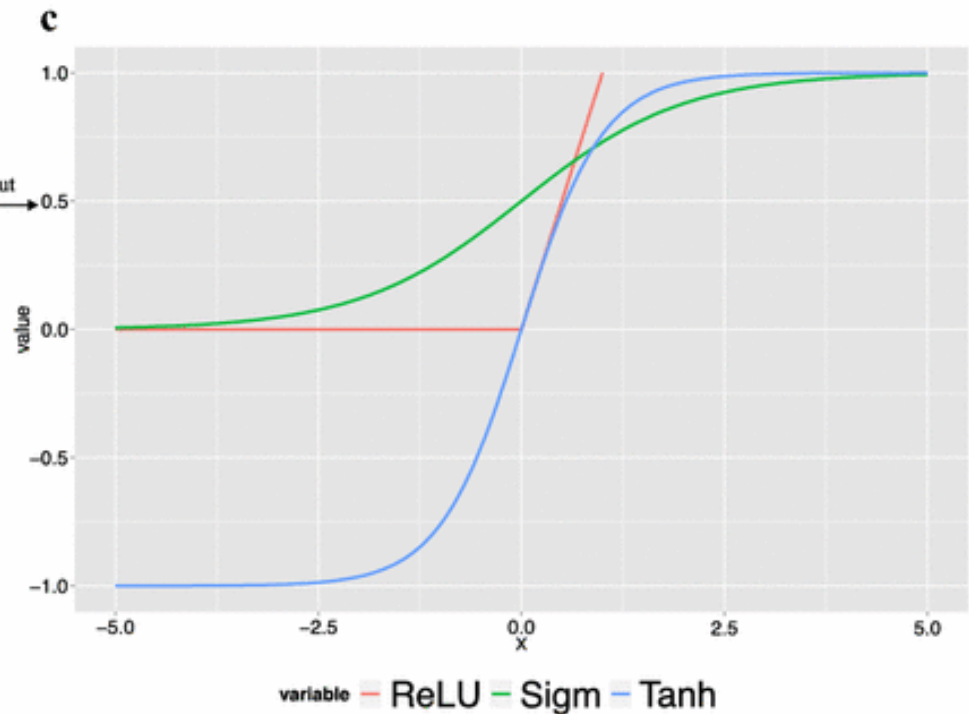
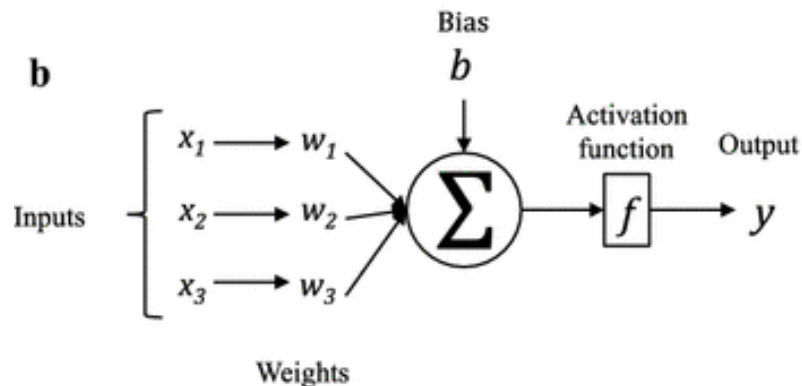
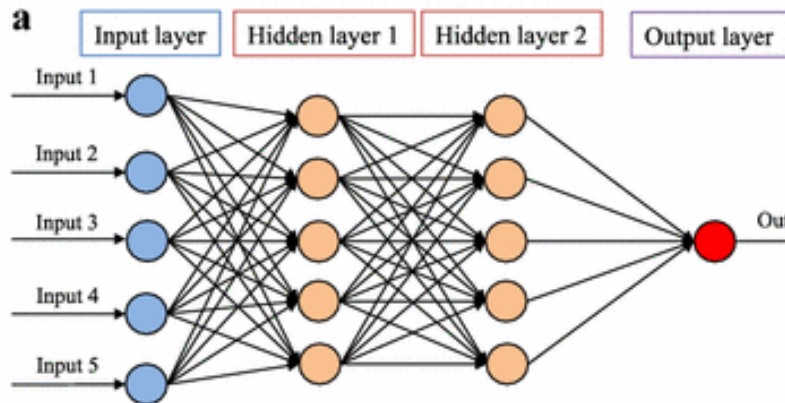
Hidden layer after dropout



활성화 함수

• activation function

```
# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



③ 학습에 필요한 최적화 방법과 손실 함수 등 설정

③ - 1 구성된 모델 요약(옵션)

- **훈련에 사용할 옵티마이저(optimizer)와 손실 함수 등을 선택**
 - 옵티마이저:
 - 입력된 데이터와 손실 함수를 기반으로 모델(w와 b)을 업데이트하는 메커니즘
 - 손실 함수:
 - 훈련 데이터에서 신경망의 성능을 측정하는 방법
 - 모델이 옳은 방향으로 학습될 수 있도록 도와 주는 기준 값
 - 훈련과 테스트 과정을 모니터링할 지표
 - 여기에서는 정확도(정확히 분류된 이미지의 비율)만 고려
- **모델 요약**

훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 모델에 설정

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# metrics=['accuracy', 'mse'])
```

모델 요약 표시

```
model.summary()
```

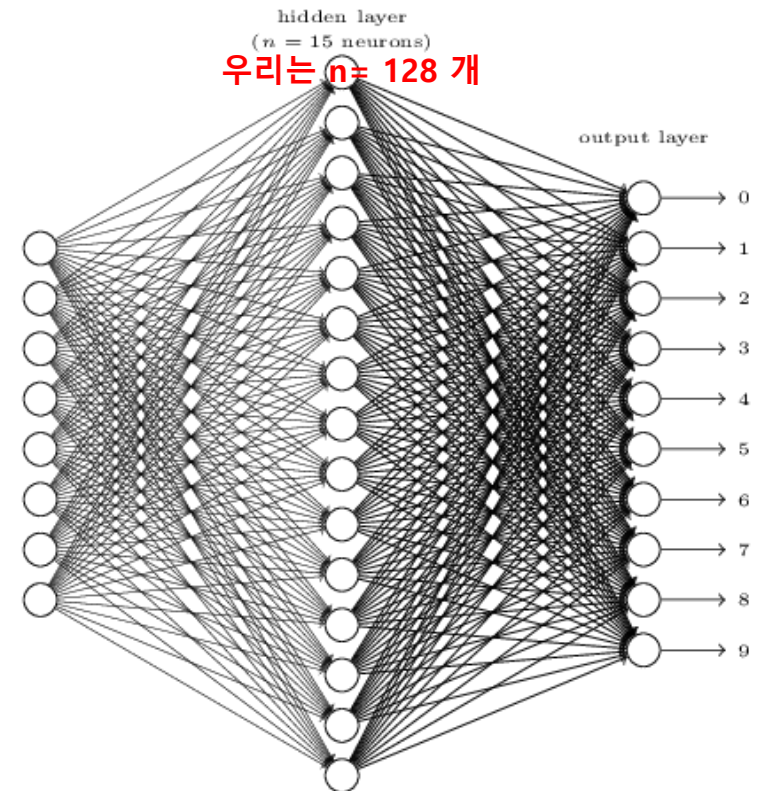
③ - 1 구성된 모델 요약(옵션)

- **model.summary()**
 - 각 층의 구조와 패라미터 수 표시
 - 총 패라미터 수
 - 모델이 구해야 할 수의 개수
 - 101,770

➡ Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0



④ 생성된 모델로 훈련 데이터 학습

- 모델을 훈련

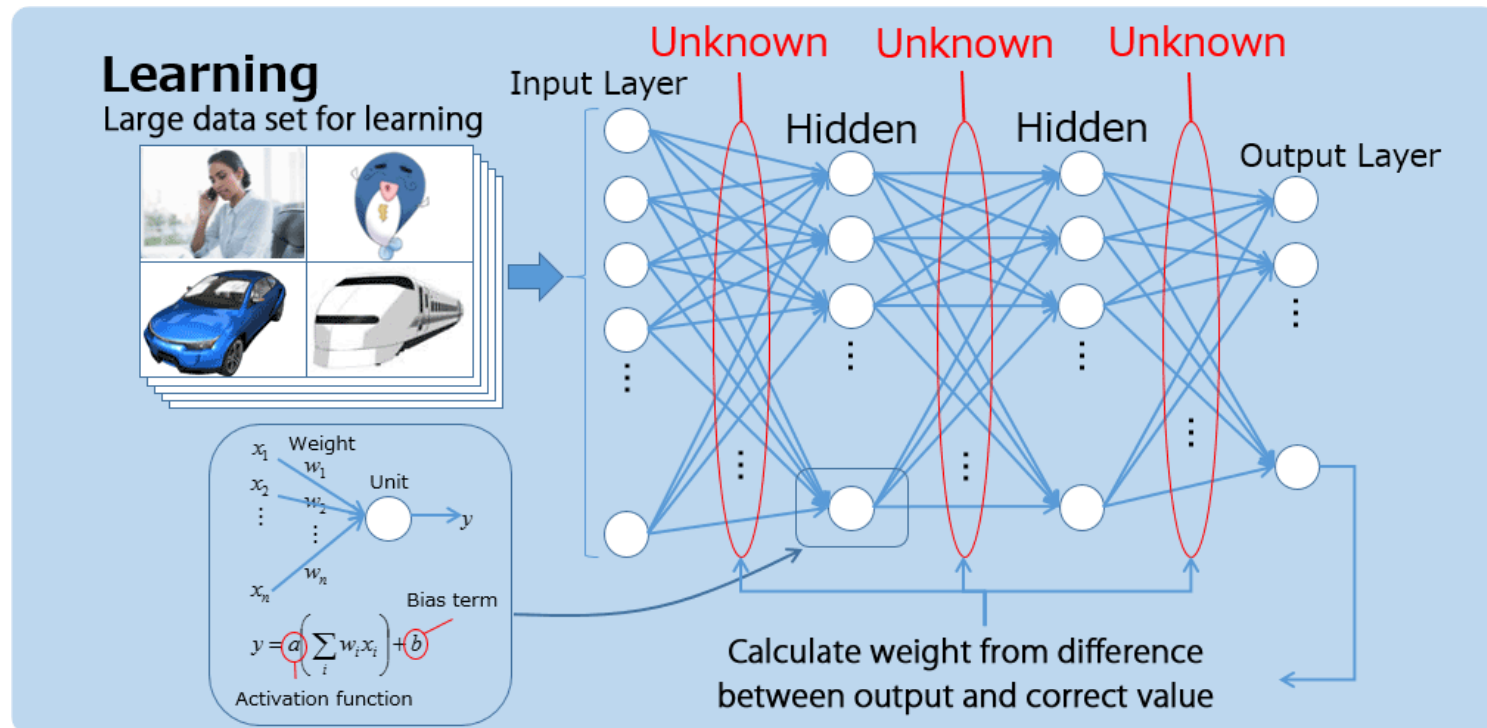
- model.fit()
 - 훈련 횟수 epochs에 지정

```
# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)
```

```
↳ Epoch 1/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0248 - accuracy: 0.9913
Epoch 2/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.0234 - accuracy: 0.9920
Epoch 3/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.0236 - accuracy: 0.9918
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0232 - accuracy: 0.9920
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0229 - accuracy: 0.9922
<tensorflow.python.keras.callbacks.History at 0x7f4a78e6cd30>
```

딥러닝 훈련, 학습

- **fit() 메서드 호출**
 - 훈련 데이터에 모델을 학습
 - 모델의 매개변수를 정하는 과정



⑤ 테스트 데이터로 성능 평가

• 모델을 평가

- 테스트 세트에서도 모델이 잘 작동하는지 확인
- `model.evaluate()`
 - 손실 값과 예측 정확도 반환
 - `loss, accuracy`

모델을 테스트 데이터로 평가

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

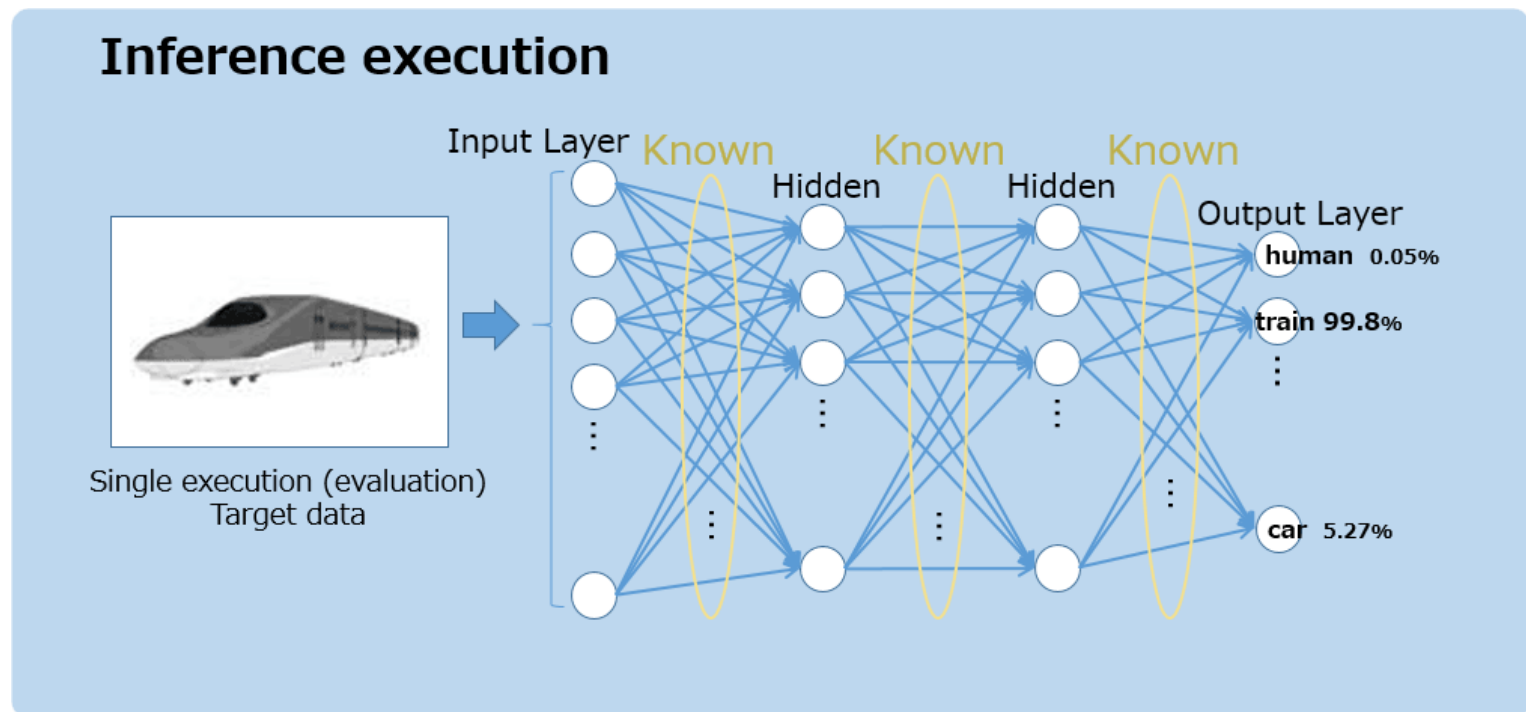


```
313/313 [=====] - 0s 1ms/step - loss: 0.0868 - accuracy: 0.9805  
[0.08675415068864822, 0.9804999828338623]
```

98%의 정확도로
손글씨를 맞춤

모델 평가 evaluate

- 테스트 데이터로 모델을 평가



MNIST 딥러닝 구현 전 소스

```
#####
import tensorflow as tf

# mnist 모듈 준비
mnist = tf.keras.datasets.mnist

# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 샘플 값을 정수 (0~255)에서 부동소수 (0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0

# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(10, activation='softmax')

])

# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 요약 표시
model.summary()

# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)

# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```