딥러닝 이해1

데이터 이해

최 석 재

lingua@naver.com

MNIST 데이터 실습

Colab에서 딥러닝 설정

- Colab에는 Tensorflow, Keras, Python이 모두 설치되어 있음
- 다음의 과정을 통해 GPU를 이용하는 버전으로만 변경하면 GPU 버전의 Tensorflow를 이용할 수 있음

- 런타임 > 런타임 유형 변경 > 하드웨어 가속기 > GPU > 저장

- GPU 버전은 12시간 연속 사용할 수 있으며, 잠시 중단한 다음 다시 사용

- 2019년 하반기에 기본 Tensorflow가 2.x로 업그레이드 되었음

MNIST 데이터셋 *Modified National Institute of Standards and Technology*

- MNIST 데이터셋은 1980년대 미국 국립표준기술연구소(National Institute of Standards and Technology, NIST)에서 수집
- 6만개의 훈련 이미지와 1만개의 테스트 이미지로 구성되어 있음

- 데이터셋은 이미 구성이 완료되어 있고, 이를 기반으로 알고리즘의 작동 방법을 숙지하고, 성능을 확인하는 용도로 사용됨

- 이 데이터셋은 keras에 이미 Numpy 배열 형태로 포함되어 있음

MNIST data loading from Keras

- from keras.datasets import mnist
- (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

- print(train_images.shape) # (60000, 28, 28)
- print(test_images.shape) # (10000, 28, 28)

keras에 내장된 데이터를 train_images, train_labels 와 같이 불러들임

train 데이터는 6만 개, test 데이터는 1만 개로 되어 있으며,
각각 가로x세로가 28x28인 3차원 데이터임

```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
(60000, 28, 28)
(10000, 28, 28)
```

MNIST data 구조

- 3차원 데이터는 하나의 대상이 다시 2차원 구조로 되어 있다는 의미

- 우리가 보통 대하는 데이터는 다음과 같은 2차원 데이터
- 자료가 10개 있으며 각각의 자료는 그 특성을 표현하는 컬럼을 가진다
- 따라서 각 자료를 가리키는 행(1번째 차원)과 특성(2번째 차원)으로 구성

ID	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

출력 설정

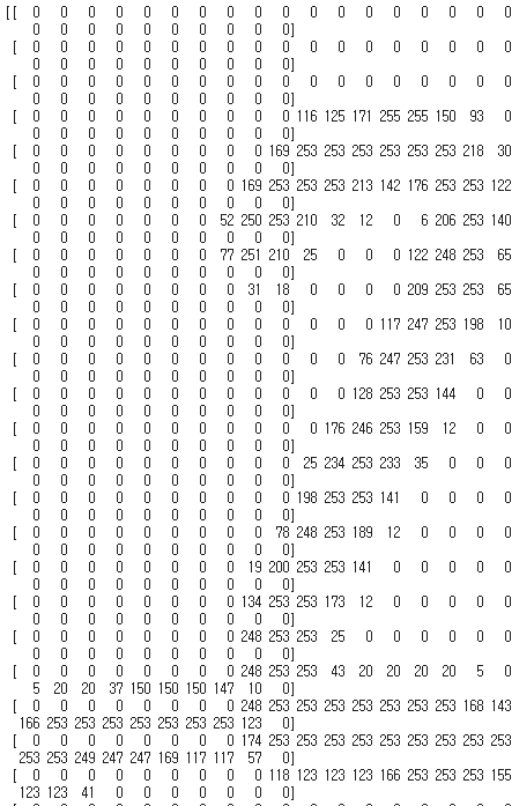
- 그런데 MNIST 데이터는 특성을 표현하는 부분을 다시 2차원(2D)으로 구성
- 특성 표현 부분을 보기 위하여 하나의 데이터를 열어본다
- 줄이 다음 칸으로 밀려 정확한 확인이 어렵다

- print(test_images[1])

- 데이터의 타입을 확인하여 넘파이 배열임을 확인
- 넘파이 배열 하나의 행을 하나의 줄에 출력되도록 설정

- print(type(test_images)) `<class 'numpy.ndarray'>`
- import numpy as np
- np.set_printoptions(linewidth=np.inf)

linewidth : 한 줄에 출력될 문자의 최대 수



재출력

- print(test_images[1])

- 두 개 차원을 이용해 숫자 '2'를 표현
- 행 쪽 차원을 먼저 보고, 열 쪽 차원을 본다
- 각 셀의 숫자는 색상 값
- 숫자로 대상을 인식한다

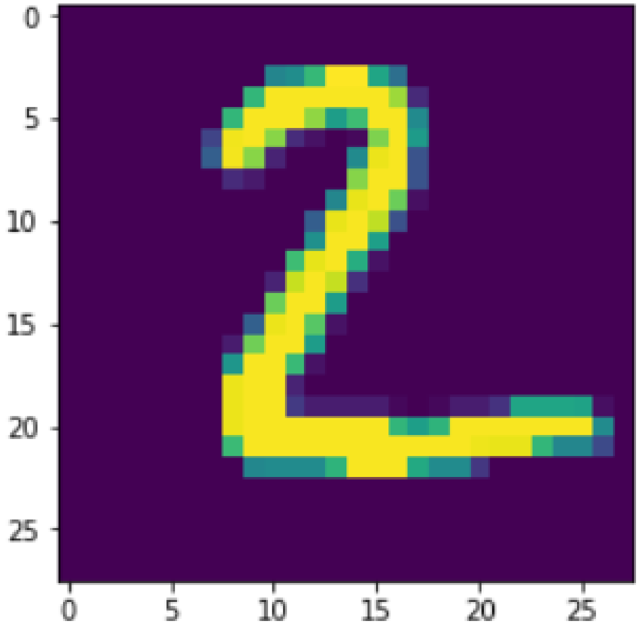
- import matplotlib.pyplot as plt
- plt.imshow(test_images[1])

②

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   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   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   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 116 125 171 255 255 150  93   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 169 253 253 253 253 253 253 218  30   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 169 253 253 253 213 142 176 253 253 122   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  52 250 253 210  32  12   0   6 206 253 140   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  77 251 210  25   0   0   0 122 248 253  65   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  31  18   0   0   0   0 209 253 253  65   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 117 247 253 198  10   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  76 247 253 231  63   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 128 253 253 144   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 176 246 253 159  12   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  25 234 253 233  35   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 198 253 253 141   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  78 248 253 189  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  19 200 253 253 141   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 134 253 253 173  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  25   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  43  20  20  20  20   5   0   5  20  20  37 150 150 150 147  10   0]
 [  0   0   0   0   0   0   0   0 248 253 253 253 253 253 253 253 168 143 166 253 253 253 253 253 253 253 123   0]
 [  0   0   0   0   0   0   0   0 174 253 253 253 253 253 253 253 253 253 253 253 249 247 247 169 117 117  57   0]
 [  0   0   0   0   0   0   0   0   0 118 123 123 123 166 253 253 253 155 123 123  41   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   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   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   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   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   0   0   0   0   0   0   0   0   0   0   0   0]]
```

Green 255, Blue 0 일 때의 Red 값



차원의 차원

- 두 개의 차원이 형성하는 의미를 생각해본다
- 행 차원이 28개이고, 열 차원이 28개이다
- 이 두 개 차원은 동등한 수준에 있는 것이 아니라,
- 하나의 행 차원(1/28)에 28개의 원소가 있는 것을 말한다
- 즉, 하나의 행이 28개의 특성으로 표현된다
- 이러한 행이 28개 모여 숫자 '2'를 표현한다

②

①

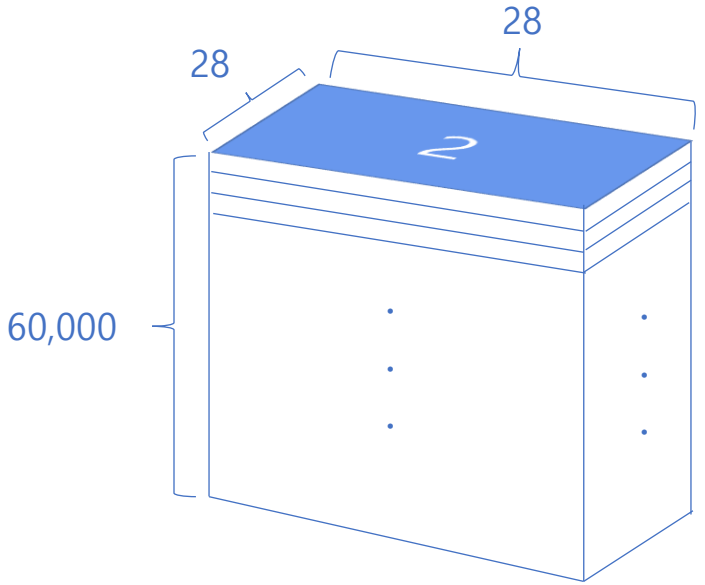
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 116 125 171 255 255 150  93   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 169 253 253 253 253 253 253 218  30   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 169 253 253 253 213 142 176 253 253 122   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  52 250 253 210  32  12   0   6 206 253 140   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  77 251 210  25   0   0   0 122 248 253  65   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  31  18   0   0   0   0 209 253 253  65   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 117 247 253 198  10   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  76 247 253 231  63   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 128 253 253 144   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 176 246 253 159  12   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  25 234 253 233  35   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 198 253 253 141   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  78 248 253 189  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  19 200 253 253 141   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 134 253 253 173  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  25   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  43  20  20  20  20   5   0   5  20  20  37 150 150 150 147  10   0]
 [  0   0   0   0   0   0   0   0 248 253 253 253 253 253 253 253 168 143 166 253 253 253 253 253 253 253 123   0]
 [  0   0   0   0   0   0   0   0 174 253 253 253 253 253 253 253 253 253 253 253 249 247 247 169 117 117  57   0]
 [  0   0   0   0   0   0   0   0   0 118 123 123 123 166 253 253 253 155 123 123  41   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   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   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   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   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   0   0   0   0   0   0   0   0   0   0   0   0]]
```

이 첫 번째 행에 28개의 원소가 존재
다음 행에도 역시 28개의 원소가 존재
이러한 행이 28개 모여 숫자 '2' 표현

차원의 [차원의 [차원]]

사이즈 60,000　　행 28　　열 28

- 이러한 숫자가 train data에는 60,000개, test data에는 10,000개가 있음



train data

차원의 [차원의 [차원]]

사이즈 60,000 행 28 열 28

- 전체 데이터를 출력하여 데이터의 구조를 확인해본다
- print(test_images)

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

 ...
```

하나의 행은 28개의 열(원소)을 갖는다

하나의 숫자는 28 개의 행을 갖는다

테스트 데이터는 전체 10,000 개의 숫자(사이즈)를 갖는다

이 숫자의 소속은
전체 사이즈 10,000 중 첫 번째 숫자 '2'의
28 개 행 중 첫 번째 행에 속하며
28 개 열(원소) 중 첫 번째 열에 속한 값이다

이 소속은 '절대' 바뀌지 않는다

데이터 슬라이싱

특정 구간의 데이터만 가져와 본다

- my_slice = test_images[0:100] # 0~99번까지 가져온다
- print(my_slice.shape) # (100, 28, 28)

3차원 데이터이므로, 위의 코드는 다음과도 동일하다

- my_slice = test_images[0:100, 0:28, 0:28]
- my_slice = test_images[0:100, :, :]

Label 정보 출력

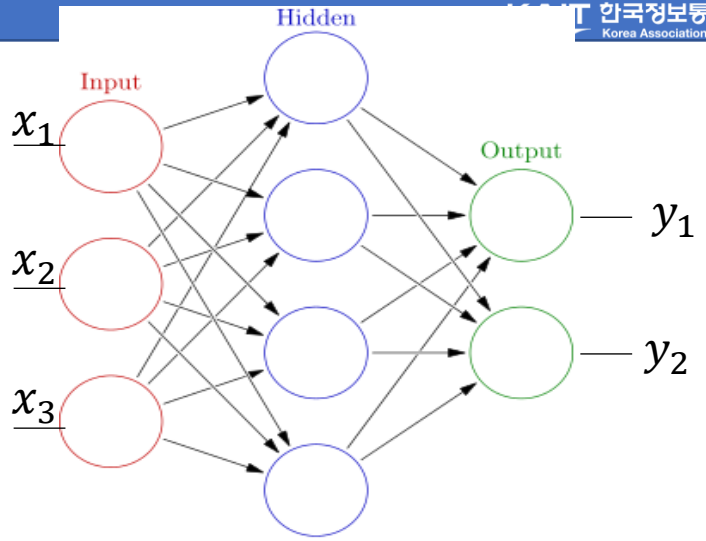
- print(len(train_labels)) # 60000
- print(len(test_labels)) # 10000

- print(train_labels) # [5, 0, 4, ..., 5, 6, 8]
- print(test_labels) # [7, 2, 1, ..., 4, 5, 6]

train, test 데이터 모두 label(정답)을 가지고 있으며, 0~9의 숫자임

```
60000
10000
[5 0 4 ... 5 6 8]
[7 2 1 ... 4 5 6]
```

Layer 쌓기



- from keras import models, layers

- model = models.Sequential()

모델 초기화

은닉층 설정
처음에는 입력 shape을 설정한다. 3차원 데이터를 2차원으로 변형할 것이다. input_shape=(특성의 수, 샘플의 수)
단, 샘플의 개수는 몇 개가 올지 알 수 없으므로 비워둔다

- model.add(layers.Dense(256, activation='relu', input_shape=(28*28,)))

출력층 설정. 숫자의 종류가 10개이므로, 10개의 유닛을 갖는 출력층을 설정한다
10개 각각에 대한 확률정보 출력. Softmax 층은 확률 점수를 출력한다

- model.add(layers.Dense(10, activation='softmax'))

Compile model

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

loss: loss function. 훈련 데이터에서 신경망의 성능을 측정하는 손실 함수

optimizer: 입력된 데이터와 손실 함수를 기반으로 가중치를 업데이트하는 방법

metrics: 훈련과 테스트 과정을 모니터링할 지표. ‘acc’라고도 씀

데이터 전처리

\# 전처리 전

- print(test_images[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   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   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   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 116 125 171 255 255 150  93   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 169 253 253 253 253 253 253 218  30   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 169 253 253 253 213 142 176 253 253 122   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  52 250 253 210  32  12   0   6 206 253 140   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  77 251 210  25   0   0   0 122 248 253  65   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  31  18   0   0   0   0 209 253 253  65   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 117 247 253 198  10   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  76 247 253 231  63   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 128 253 253 144   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 176 246 253 159  12   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  25 234 253 233  35   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 198 253 253 141   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  78 248 253 189  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  19 200 253 253 141   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 134 253 253 173  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  25   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  43  20  20  20  20   5   0   5  20  20  37 150 150 150 147  10   0]
 [  0   0   0   0   0   0   0   0 248 253 253 253 253 253 253 253 168 143 166 253 253 253 253 253 253 253 123   0]
 [  0   0   0   0   0   0   0   0 174 253 253 253 253 253 253 253 253 253 253 253 249 247 247 169 117 117  57   0]
 [  0   0   0   0   0   0   0   0   0 118 123 123 123 166 253 253 253 155 123 123  41   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   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   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   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   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   0   0   0   0   0   0   0   0   0   0   0   0]]
```

\# 현재의 데이터는 (60000, 28, 28)의 3차원이면서,
\# 0~255 사이의 숫자(색의 번호)로 되어 있음

\# 이 데이터를 입력층에 맞도록
\# (60000, 28*28)의 2차원 데이터로 변환하고,

\# 값을 0 ~ 1이 되도록 255로 나눔

※ 컴퓨터가 읽는 것은 결국 숫자이다

데이터 변환

- train_images = train_images.reshape((60000, 28*28))
- train_images = train_images.astype('float32')/255

- test_images = test_images.reshape((10000, 28*28))
- test_images = test_images.astype('float32')/255

reshape 함수를 이용해 데이터를 (60000, 784) 크기로 변환하고,
각 값을 255로 나누되, type을 소수점을 받는 float32 형으로 맞춤

전처리 후

- np.set_printoptions(linewidth=310)
- print(test_images[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.         0.         0.         0.         0.         0.
0.         0.         0.         0.45490196 0.49019608 0.67058825 1.         1.         0.5882353  0.3647059  0.         0.         0.         0.         0.         0.
0.         0.         0.6627451  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.85490197 0.11764706 0.         0.         0.         0.         0.
0.         0.6627451  0.99215686 0.99215686 0.99215686 0.8352941  0.5568628  0.6901961  0.99215686 0.99215686 0.47843137 0.         0.         0.         0.         0.
0.20392157 0.98039216 0.99215686 0.8235294  0.1254902  0.04705882 0.         0.02352941 0.80784315 0.99215686 0.54901963 0.         0.         0.         0.         0.
0.3019608  0.9843137  0.8235294  0.09803922 0.         0.         0.         0.47843137 0.972549   0.99215686 0.25490198 0.         0.         0.         0.         0.
0.         0.12156863 0.07058824 0.         0.         0.         0.         0.81960785 0.99215686 0.99215686 0.25490198 0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.         0.45882353 0.96862745 0.99215686 0.7764706  0.03921569 0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.29803923 0.96862745 0.99215686 0.90588236 0.24705882 0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.5019608  0.99215686 0.99215686 0.5647059  0.         0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.6901961  0.9647059  0.99215686 0.62352943 0.04705882 0.         0.         0.         0.         0.         0.         0.
0.         0.         0.         0.09803922 0.91764706 0.99215686 0.9137255  0.13725491 0.         0.         0.         0.         0.         0.         0.         0.
0.         0.         0.         0.7764706  0.99215686 0.99215686 0.5529412  0.         0.         0.         0.         0.         0.         0.         0.         0.
0.         0.         0.30588236 0.972549   0.99215686 0.7411765  0.04705882 0.         0.         0.         0.         0.         0.         0.         0.         0.
0.         0.07450981 0.78431374 0.99215686 0.99215686 0.5529412  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
0.         0.5254902  0.99215686 0.99215686 0.6784314  0.04705882 0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
0.         0.972549   0.99215686 0.99215686 0.09803922 0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
0.         0.972549   0.99215686 0.99215686 0.16862746 0.07843138 0.07843138 0.07843138 0.07843138 0.01960784 0.         0.01960784 0.07843138 0.07843138 0.14509805 0.5
0.         0.972549   0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.65882355 0.56078434 0.6509804  0.99215686 0.99215686 0.99215686 0.9
0.         0.68235296 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.9764706  0.96862745 0.9
0.         0.         0.4627451  0.48235294 0.48235294 0.48235294 0.6509804  0.99215686 0.99215686 0.99215686 0.60784316 0.48235294 0.48235294 0.16078432 0.         0.
0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.
```

종속변수의 원-핫 인코딩

- from tensorflow.keras.utils import to_categorical
- train_labels = to_categorical(train_labels)
- test_labels = to_categorical(test_labels)

```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 1., ..., 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.]], dtype=float32)
```

※ 10개의 원소 중 하나만 1이다

데이터 학습

- model.fit(train_images, train_labels, epochs=5, batch_size=128)

```
60000/60000 [==============================] - 8s 133us/step - loss: 0.2563 - acc: 0.9257
Epoch 2/5
60000/60000 [==============================] - 3s 45us/step - loss: 0.1049 - acc: 0.9691
Epoch 3/5
60000/60000 [==============================] - 3s 47us/step - loss: 0.0682 - acc: 0.9797
Epoch 4/5
60000/60000 [==============================] - 3s 46us/step - loss: 0.0487 - acc: 0.9853
Epoch 5/5
60000/60000 [==============================] - 3s 47us/step - loss: 0.0370 - acc: 0.9892
```

fit 메소드를 이용하여 모델을 훈련시킴
train 데이터와 그의 label, 반복횟수 및 1번에 훈련할 데이터의 양(batch size)을 결정함
메모리 한계와 속도 저하 문제로 한 번에 전체 데이터를 학습하지는 않는다
일반적으로 8~512개의 데이터를 한 번에 학습시킨다

손실점수는 낮아지고, 정확도는 높아짐
훈련데이터에 대한 최종 정확도 98.9%

예측

- test_loss, test_acc = model.evaluate(test_images, test_labels)
- print('test_acc:', test_acc)

```
10000/10000 [==============================] - 0s 39us/step
test_acc: 0.9808
```

evaluate 함수로 테스트 데이터의 정확도를 볼 수 있음

적절한 데이터가 없다면!

- 그러나 적절한 데이터가 없다면 오분류가 일어날 수 있다