케라스로 딥러닝 모델 만들기

학습 내용

- MNIST 데이터 셋을 활용하여 딥러닝 모델을 구현해 본다.

환경

- google colab
  - tensorflow 2.8.0
  - keras 2.8.0
  - python 3.8.13

목차

케라스 딥러닝 모델 만들기

- 가. 데이터 셋 준비
  - 데이터 준비(훈련셋, 검증셋, 시험셋 등)
  - 딥러닝 모델의 학습 및 평가를 위한 데이터 형태 맞추기(포맷 변환)
- 나. 모델 구성
  - 모델(Sequential)을 생성 후, 레이어를 추가하여 구성
  - 복잡한 모델을 구성시에 Keras API를 사용
- 다. 모델 학습과정 설정하기(학습에 대한 설정 - compile())
  - 학습에 대한 설정, 손실 함수 및 최적화 방법 정의
- 라. 모델 학습(모델을 훈련셋으로 학습 - fit() 함수)
- 마. 학습과정 살펴보기(훈련셋, 검증셋의 손실 및 정확도 측정)
- 바. 모델 평가(evaluate()) - 준비된 시험셋으로 학습 모델 평가
- 사. 모델 사용하기(predict())

In [ ]:
```
import tensorflow as tf
import keras as keras
import sys
```

In [ ]:
```
print(tf.__version__)
print(keras.__version__)
print(sys.version)
```

```
2.8.0
2.8.0
3.7.13 (default, Apr 24 2022, 01:04:09)
[GCC 7.5.0]
```

01 라이브러리 임포트 및 데이터 준비

In [ ]:
```
from keras.datasets import mnist                    # 데이터 셋 불러오기
```

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import np_utils
```

데이터 셋 MNIST

- 6만개의 훈련 이미지
- 1만개의 테스트 이미지
- 딥러닝계의 MNIST
- 1980년대 미국 국립표준 기술 연구소(National Institute of Standards and TEchnology, NIST)에서 수집한 데이터

클래스, 샘플, 레이블

- 분류의 문제의 범주를 클래스라하고,
- 데이터 포인터를 샘플(sample)이라고 한다.
- 특정 샘플의 클래스는 레이블(Label)이라고 한다.

In [ ]:
```
### 데이터 셋 불러오기
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train_n = X_train.copy()
y_train_n = y_train.copy()
X_test_n = X_test.copy()
y_test_n = y_test.copy()

# 데이터 셋 크기
# 60000개의 학습용 데이터 셋, 10000개의 테스트 데이터 셋
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

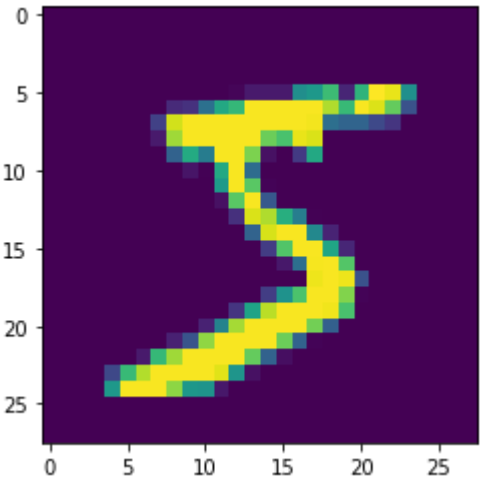
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
ets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
### x_train 의 하나의 데이터 확인
### 60000장의 이미지( 28, 28 숫자데이터)

plt.imshow(X_train[0])
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f9b37a27390>
```

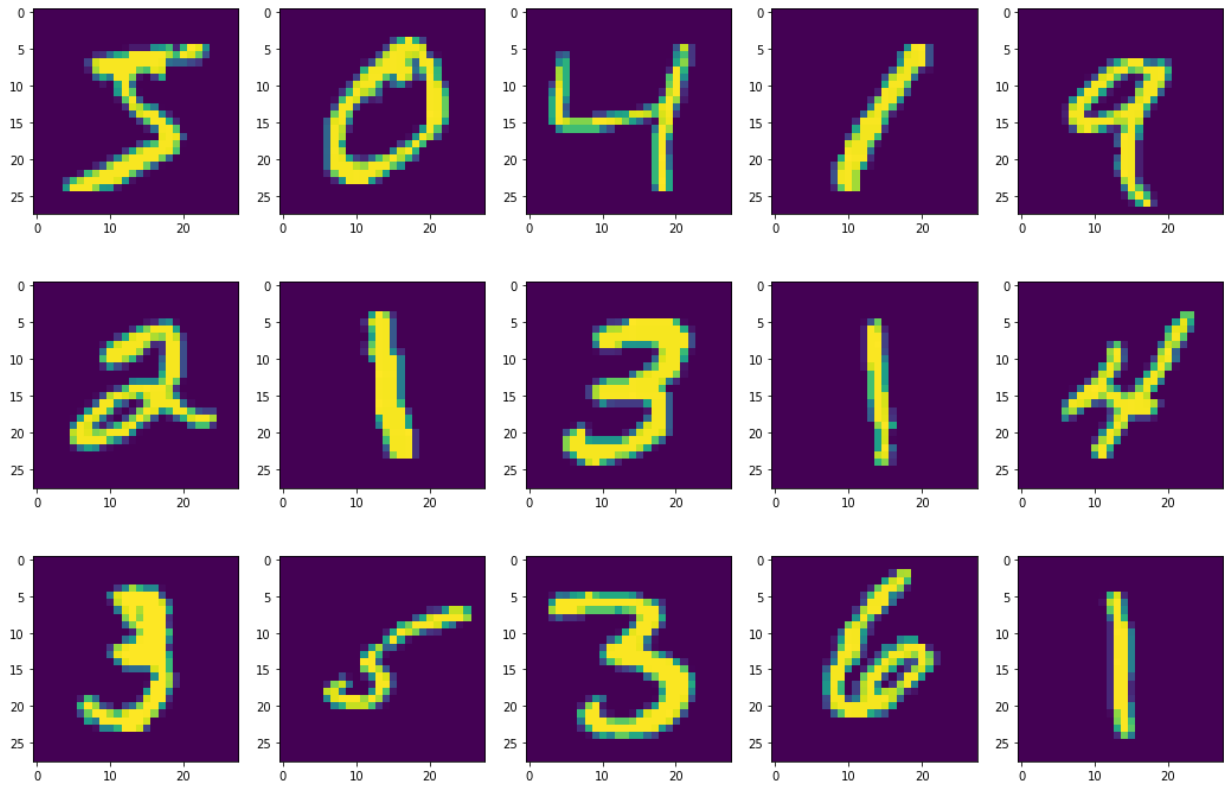


10개의 y_train 데이터 셋 확인

In [ ]:
```
print("label={}".format(y_train[0:10]))    # y 레이블 데이터 0~10개 확인
```

```
label=[5 0 4 1 9 2 1 3 1 4]
```

In [ ]:
```
fig, axes = plt.subplots(3, 5, figsize=(18,12) )
for image, ax in zip( X_train, axes.ravel() ):
    ax.imshow(image) # 이미지 표시
```



In [ ]:
```
## 데이터 셋의 변경 60000, 28, 28  -> 60000, 784 (28*28)
## 데이터 셋의 변경 10000, 28, 28  -> 10000, 784 (28*28)
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

## 데이터 셋의 변경 60000,    -> 60000, 10 (28*28)
## 데이터 셋의 변경 10000,    -> 10000, 10 (28*28)
print(y_train.shape, y_test.shape)
print(y_train[0:5])
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

```
print(y_train.shape, y_test.shape)
print(y_train[0:5])
```

```
(60000,) (10000,)
[5 0 4 1 9]
(60000, 10) (10000, 10)
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 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. 1.]]
```

In [ ]:

```
## 데이터 자료형 변경
## 01. 실수형 변경.
## 02. 값의 범위를 정규화(0~255) -> 0~1로 변경
print(X_train[0])
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   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  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   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
0.49411765 0.53333336 0.6862745  0.10196079 0.6509804  1.
0.96862745 0.49803922 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.11764706 0.14117648 0.36862746 0.6039216
0.6666667  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.88235295 0.6745098  0.99215686 0.9490196  0.7647059  0.2509804
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.19215687
0.93333334 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.99215686 0.99215686 0.99215686 0.9843137  0.3647059  0.32156864
0.32156864 0.21960784 0.15294118 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.07058824 0.85882354 0.99215686
0.99215686 0.99215686 0.99215686 0.99215686 0.7764706  0.7137255
0.96862745 0.94509804 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.3137255  0.6117647  0.41960785 0.99215686
0.99215686 0.8039216  0.04313726 0.         0.16862746 0.6039216
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.05490196 0.00392157 0.6039216  0.99215686 0.3529412
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.54509807 0.99215686 0.74509805 0.00784314 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313726
0.74509805 0.99215686 0.27450982 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.13725491 0.94509804
0.88235295 0.627451   0.42352942 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764707 0.9411765  0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.1764706  0.7294118  0.99215686 0.99215686
0.5882353  0.10588235 0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.3647059  0.9882353  0.99215686 0.73333335
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.9764706  0.99215686 0.9764706  0.2509804  0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980395 0.7176471  0.99215686
0.99215686 0.8117647  0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.5803922
0.8980392  0.99215686 0.99215686 0.99215686 0.98039216 0.7137255
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705883 0.8666667  0.99215686 0.99215686 0.99215686
0.99215686 0.7882353  0.30588236 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882354 0.8352941  0.99215686
0.99215686 0.99215686 0.99215686 0.7764706  0.31764707 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058825
0.85882354 0.99215686 0.99215686 0.99215686 0.99215686 0.7647059
0.3137255  0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.21568628 0.6745098  0.8862745  0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156866 0.04313726 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333336 0.99215686
0.99215686 0.99215686 0.83137256 0.5294118  0.5176471  0.0627451
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.        ]
```

위의 내용을 이렇게 줄일수도 있음.

```
X_train = X_train.reshape(60000, 784).astype('float32') / 255.0
X_test = X_test.reshape(10000, 784).astype('float32') / 255.0
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

02. 모델 구성 및 학습, 평가하기

In [ ]:
```
model = Sequential()
model.add( Dense(units=64, input_dim=28*28)) #입력층(28*28=784노드) - 은닉층(64개
model.add( Activation('tanh') )
model.add( Dense(units=32))
model.add( Activation('tanh') )
model.add( Dense(units=32))
model.add( Activation('tanh') )

# 한줄로 한다면
# model.add(Dense(32, activation='tanh'))

model.add(Dense(10))                    # 출력층(10개 노드)
model.add(Activation('softmax'))
```

다음과 같이 은닉층이 하나 추가되면 다음과 같다.

```
model = Sequential()
model.add(Dense(units=64, input_dim=28*28, activation='relu'))
# 입력층(28*28=784노드) - 은닉층(8개노드)
model.add(Dense(units=32, activation='relu'))                    #
두번째 은닉층(8개)
model.add(Dense(units=10, activation='softmax'))                 #
출력층(10개 노드)
```

손실함수(loss function) or 목적함수(objective function)

- 신경망의 출력을 제어하기 위해 출력이 기대하는 것보다 얼마나 벗어났는지에 대해 측정하기
- 비용함수는 모든 훈련 데이터에 대한 손실 함수의 합을 나타내고 목적함수는 더 일반적인 용어로 최적화하기 위한 대상 함수를 의미한다.

03. 모델 학습과정 설정하기

- model.compile
  - 손실 함수(loss function) : 훈련 데이터에서 신경망의 성능을 측정하는 방법. 네트워크가 옳은 방향으로 학습될 수 있도록 도와준다.
  - 옵티마이저(optimizer) : 입력된 데이터와 손실 함수를 기반으로 네트워크를 업데이트하는 메커니즘.
  - metrics : 훈련과 테스트 과정을 모니터링할 지표 : 정확도(정확히 분류된 이미지의 비율)만 고려
- categorical_crossentropy : 여러가지 오차를 구하는 함수 중의 하나
- sgd(Stochastic Gradient Descent) : 데이터 셋을 미니배치만큼 돌려서 찾아가는 방법
- accuracy : 정확도로 성능을 측정

In [ ]:
```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

04. 모델 학습시키기

- [모델명].fit(, , ,)
- hist = model.fit(x_train, y_train, epochs=5, batch_size=32)
  - x_train : 입력 데이터
  - y_train : 출력(예측) 데이터

- epochs : 전체 데이터 몇번 돌리것인가?
- batch_size : 몇개씩 데이터를 돌려볼 것인가?

In [ ]:
```
# hist = model.fit(X_train, y_train, epochs=5, batch_size=32)
hist = model.fit(X_train, y_train,
                 validation_data=(X_test, y_test),
                 epochs=10,
                 batch_size=100,
                 verbose=1)
```

```
Epoch 1/10
600/600 [==============================] - 3s 4ms/step - loss: 1.1078 - accura
cy: 0.7370 - val_loss: 0.6466 - val_accuracy: 0.8557
Epoch 2/10
600/600 [==============================] - 2s 4ms/step - loss: 0.5436 - accura
cy: 0.8696 - val_loss: 0.4517 - val_accuracy: 0.8883
Epoch 3/10
600/600 [==============================] - 2s 4ms/step - loss: 0.4222 - accura
cy: 0.8901 - val_loss: 0.3772 - val_accuracy: 0.9027
Epoch 4/10
600/600 [==============================] - 2s 4ms/step - loss: 0.3652 - accura
cy: 0.9015 - val_loss: 0.3349 - val_accuracy: 0.9105
Epoch 5/10
600/600 [==============================] - 2s 4ms/step - loss: 0.3295 - accura
cy: 0.9092 - val_loss: 0.3081 - val_accuracy: 0.9160
Epoch 6/10
600/600 [==============================] - 2s 4ms/step - loss: 0.3039 - accura
cy: 0.9156 - val_loss: 0.2866 - val_accuracy: 0.9199
Epoch 7/10
600/600 [==============================] - 3s 4ms/step - loss: 0.2841 - accura
cy: 0.9202 - val_loss: 0.2706 - val_accuracy: 0.9241
Epoch 8/10
600/600 [==============================] - 3s 4ms/step - loss: 0.2678 - accura
cy: 0.9244 - val_loss: 0.2575 - val_accuracy: 0.9261
Epoch 9/10
600/600 [==============================] - 3s 4ms/step - loss: 0.2540 - accura
cy: 0.9279 - val_loss: 0.2463 - val_accuracy: 0.9307
Epoch 10/10
600/600 [==============================] - 3s 4ms/step - loss: 0.2419 - accura
cy: 0.9308 - val_loss: 0.2363 - val_accuracy: 0.9342
```

05. 학습과정 살펴보기

- 각 epoch 별 loss값과 acc 값을 확인해 보기
- 그래프로 확인해 보기

In [ ]:
```
# 값 확인
hist.history.keys()
```

Out[ ]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

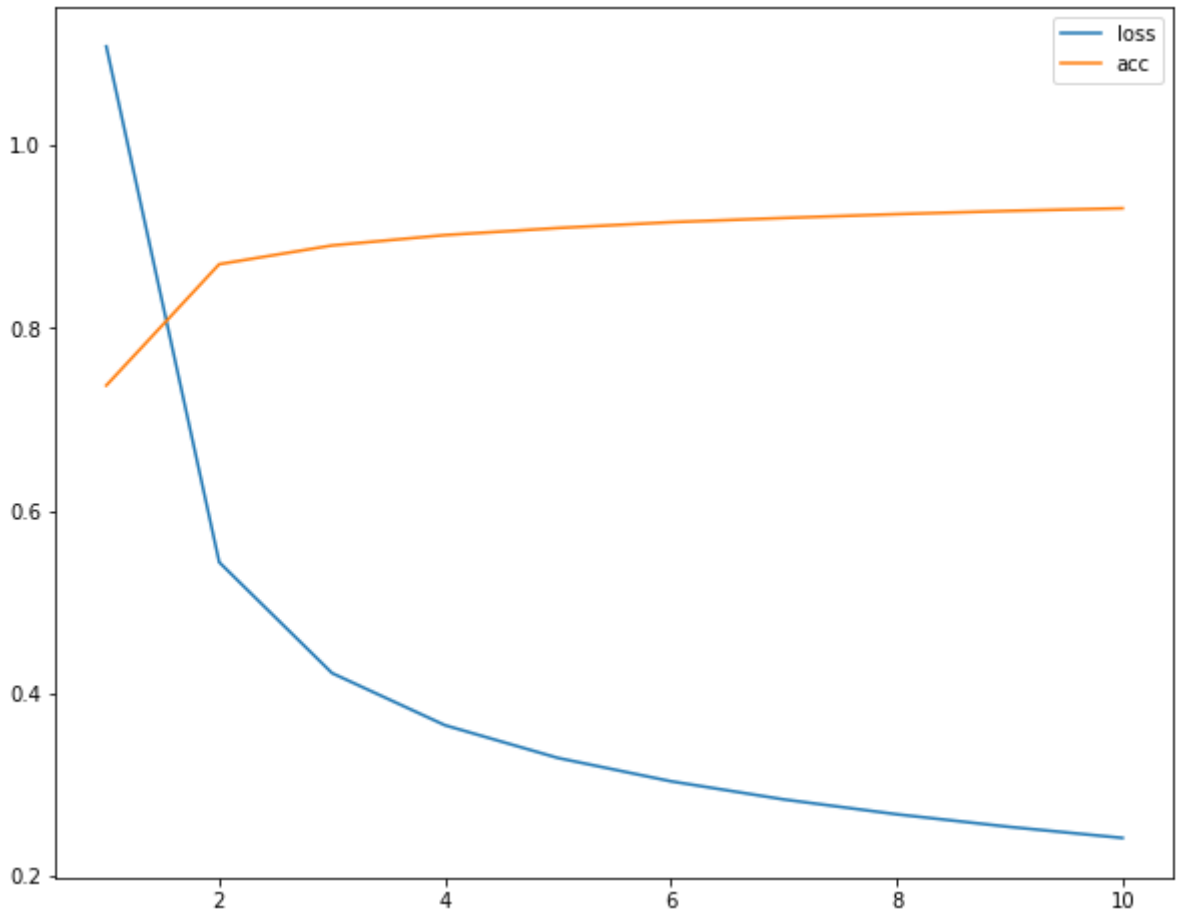
In [ ]:
```
# 10번의 epoch마다의 loss(손실)과 accuracy(정확도)의 값.
print('## training loss and acc ##')
print(hist.history['loss'])
print(hist.history['accuracy'])
```

```
## training loss and acc ##
[1.107835054397583, 0.5435747504234314, 0.42223066091537476, 0.365223675966262
8, 0.32945185899734497, 0.3039432764053345, 0.2840723395347595, 0.267840772867
20276, 0.2540345788002014, 0.24194549024105072]
[0.7369833588600159, 0.869616687297821, 0.8901333212852478, 0.901466667652130
1, 0.9092000126838684, 0.9155666828155518, 0.9201666712760925, 0.9243666529655
457, 0.9279333353042603, 0.9307666420936584]
```

In [ ]:
```
plt.figure(figsize=(10,8),facecolor='white')
```

```
x_lim = range(1,11)
plt.plot(x_lim,  hist.history['loss'])
plt.plot(x_lim,  hist.history['accuracy'])
plt.legend(['loss','acc'])
```

Out[ ]: <matplotlib.legend.Legend at 0x285e5e4dbe0>



6. 모델 평가하기

- test 데이터 셋을 활용하여 만들어진 모델을 평가해보기

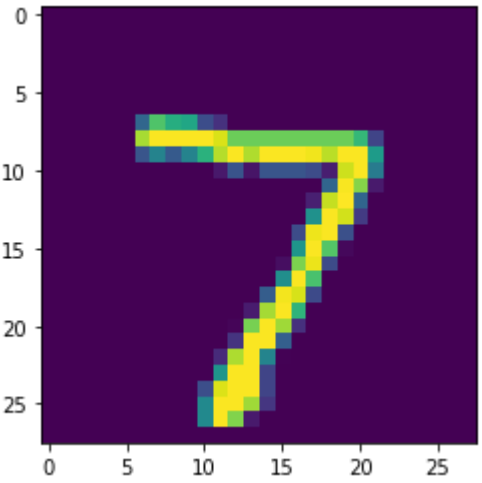
In [ ]:
```
loss_and_metrics = model.evaluate(X_test, y_test, batch_size=32)
print('## evaluation loss and_metrics ##')
print(loss_and_metrics)  # 최종 데이터 loss와 정확도(accuracy)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.2363 - accura
cy: 0.9342
## evaluation loss and_metrics ##
[0.23633261024951935, 0.9341999888420105]
```

7. 모델 사용하여 예측해 보기

In [ ]:
```
### x_train 의 하나의 데이터 확인
plt.imshow(X_test_n[0])
```

Out[ ]: <matplotlib.image.AxesImage at 0x285e60497f0>



In [ ]:
```
import numpy as np
# np.set_printoptions(precision=3)
# 좀 더 확인하기 쉽게 표시
np.set_printoptions(formatter={'float_kind': lambda x: "{0:0.6f}".format(x)})
```

In [ ]:
```
Xhat = X_test[0:1]
yhat = model.predict(Xhat)
print('## yhat ##')
print(yhat) # 각 값의 확률을 표시
print(yhat.argmax(axis=1))
```

```
## yhat ##
[[0.000698 0.000145 0.000659 0.006960 0.000064 0.000173 0.000007 0.986645
  0.000066 0.004583]]
[7]
```

실습과제

- (1) epoch를 30으로 늘려서 확인해 보자.
  - model.evaluate()의 값 확인
- (2) epoch를 30과 Activation을 Relu로 변경후, 해보기
  - model.evaluate()의 값 확인
- (3) epoch를 30과 은닉층을 3개로 변경 후, 해보기
  - model.evaluate()의 값 확인
- (4) epoch를 30과 은닉층 2개, 노드수를 64로 늘려서 해보기
  - model.evaluate()의 값 확인

REF

참고 동영상 : https://www.youtube.com/watch?v=aircAruvnKk