```
import tensorflow as tf
import numpy as np
import pandas as pd
```

```
import mnist # 같은 디렉토리 내에 있ㄴ느 파일들 불러오는 코드
from tensorflow.keras.utils import to_categorical # 원 핫 인코딩 하는 함수
```

```
# 학습용과 테스트용 데이터 나눠서 받아옴
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# 28*28 사이즈의 이미지 6만개
print(x_train.shape)
```

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

```
# normalization 하지 않았으므로 0~255 사이의 값을 가짐
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]]
```

```
# 0~255 사이의 값을 0~1 사이의 값으로 Normalization
def convertData(x):
    x = np.array(x/255.0, dtype=np.float32)
    return x


# Normalization
x_train = convertData(x_train)
x_test = convertData(x_test)

# 원 핫 인코딩
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)


# 데이터 확인, 0~1값으로 변경된 것을 알 수 있음
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.
  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
```

```
# 데이터 확인, 원 핫 인코딩 된 것을 알 수 있음
print(y_train[0])
```

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

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, Input, Flatten, Dropout, Dense, Activation, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler

IMAGE_SIZE = len(x_train[0])

# input으로 28*28*1(흑백)의 크기를 가짐
input = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1))

# Conv2D 함수로 데이터의 특징 추출
# 필터 32개 사용해 특징맵 32개 생성, 커널 사이즈 (3,3), padding=same으로 설정하여 이미지 사이즈 유지
output = Conv2D(filters=32, kernel_size=(3,3), padding='same')(input)

#배치 정규화
# 각 배치의 데이터를 평균 0, 분산 1로 정규화
output = BatchNormalization()(output)
output = Activation('relu')(output)


output = Conv2D(filters=32, kernel_size=(3,3), padding='same')(output)
output = BatchNormalization()(output)
output = Activation('relu')(output)

# 커진 특징맵 크기를 줄여 계샨량 감소, 과적합 방지
output = MaxPooling2D(pool_size=(2,2))(output)


output = Conv2D(filters=64, kernel_size=(3,3), padding='same')(output)
output = BatchNormalization()(output)
output = Activation('relu')(output)


output = Conv2D(filters=64, kernel_size=(3,3), padding='same')(output)
output = BatchNormalization()(output)
output = Activation('relu')(output)
output = MaxPooling2D(pool_size=2)(output)


output = Conv2D(filters=128, kernel_size=3, padding='same')(output)
output = BatchNormalization()(output)
output = Activation('relu')(output)


output = Conv2D(filters=128, kernel_size=3, padding='same')(output)
output = BatchNormalization()(output)
output = Activation('relu')(output)
output = MaxPooling2D(pool_size=2)(output)

# 은닉층 쌓기 전에 데이터 1차원으로 변경
output = Flatten(name='flatten')(output)

# 과적합 방지
# 30%의 뉴런 무작위로 선택해 비활성화
output = Dropout(rate=0.3)(output)
# 은닉층 쌓음
```

```
output = Dense(300, activation='relu', name='fc1')(output)
output = Dropout(rate=0.3)(output)

# 10개 중에 하나의 답을 고르는 것이므로 마지막 레이어의 활성화 함수는 softmax
output = Dense(10, activation='softmax', name='output')(output)

model = Model(inputs = input, outputs = output)


from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping


np.random.seed(2020)
tf.random.set_seed(2020)

# patience동안 val_loss의 값이 변화가 없으면 학습률을 factor만큼 줄임
# val_loss가 더 이상 작아지지 않으면 학습률을 줄여서 과적합 방지
learning_rate_cb = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience= 3, mode='min', verbose=1)

# patience 동안 val_loss의 값이 변화가 없으면 학습 중단
earlystop_cb = EarlyStopping(monitor='val_loss', patience=5, mode='min', verbose=1)

# optimizer로 Adam사용
# 원 핫 인코딩을 했으므로 categorical crossentropy 사용
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# 검증 데이터를 훈련 데이터에서 0.2만큼 추출하여 사용
# callback 함수로 학습률 조정 함수와, 미리 멈추는 함수 사용
history = model.fit(x=x_train, y=y_train, batch_size=32, epochs=50, shuffle=True, validation_split=0.2, callbacks=[learning_
```

```
Epoch 1/50
1500/1500 [==============================] - 149s 100ms/step - loss: 0.1615 - accuracy: 0.9506 - val_loss: 0.0551 - val
Epoch 2/50
1500/1500 [==============================] - 154s 103ms/step - loss: 0.0583 - accuracy: 0.9829 - val_loss: 0.0493 - val
Epoch 3/50
1500/1500 [==============================] - 154s 103ms/step - loss: 0.0458 - accuracy: 0.9860 - val_loss: 0.0317 - val
Epoch 4/50
1500/1500 [==============================] - 156s 104ms/step - loss: 0.0392 - accuracy: 0.9883 - val_loss: 0.0510 - val
Epoch 5/50
1500/1500 [==============================] - 154s 103ms/step - loss: 0.0336 - accuracy: 0.9898 - val_loss: 0.0393 - val
Epoch 6/50
1500/1500 [==============================] - 155s 103ms/step - loss: 0.0283 - accuracy: 0.9918 - val_loss: 0.0274 - val
Epoch 7/50
1500/1500 [==============================] - 159s 106ms/step - loss: 0.0237 - accuracy: 0.9927 - val_loss: 0.0277 - val
Epoch 8/50
1500/1500 [==============================] - 155s 103ms/step - loss: 0.0212 - accuracy: 0.9939 - val_loss: 0.0266 - val
Epoch 9/50
1500/1500 [==============================] - 156s 104ms/step - loss: 0.0188 - accuracy: 0.9943 - val_loss: 0.0322 - val
Epoch 10/50
1500/1500 [==============================] - 157s 105ms/step - loss: 0.0177 - accuracy: 0.9944 - val_loss: 0.0289 - val
Epoch 11/50
1500/1500 [==============================] - ETA: 0s - loss: 0.0151 - accuracy: 0.9954
Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
1500/1500 [==============================] - 154s 103ms/step - loss: 0.0151 - accuracy: 0.9954 - val_loss: 0.0347 - val
Epoch 12/50
1500/1500 [==============================] - 154s 103ms/step - loss: 0.0066 - accuracy: 0.9977 - val_loss: 0.0252 - val
Epoch 13/50
1500/1500 [==============================] - 154s 103ms/step - loss: 0.0032 - accuracy: 0.9991 - val_loss: 0.0268 - val
Epoch 14/50
1500/1500 [==============================] - 155s 103ms/step - loss: 0.0025 - accuracy: 0.9991 - val_loss: 0.0288 - val
Epoch 15/50
1500/1500 [==============================] - ETA: 0s - loss: 0.0018 - accuracy: 0.9993
Epoch 00015: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
1500/1500 [==============================] - 154s 103ms/step - loss: 0.0018 - accuracy: 0.9993 - val_loss: 0.0284 - val
Epoch 16/50
1500/1500 [==============================] - 156s 104ms/step - loss: 0.0011 - accuracy: 0.9997 - val_loss: 0.0289 - val
Epoch 17/50
1500/1500 [==============================] - 154s 103ms/step - loss: 9.2389e-04 - accuracy: 0.9997 - val_loss: 0.0293 -
Epoch 00017: early stopping
```

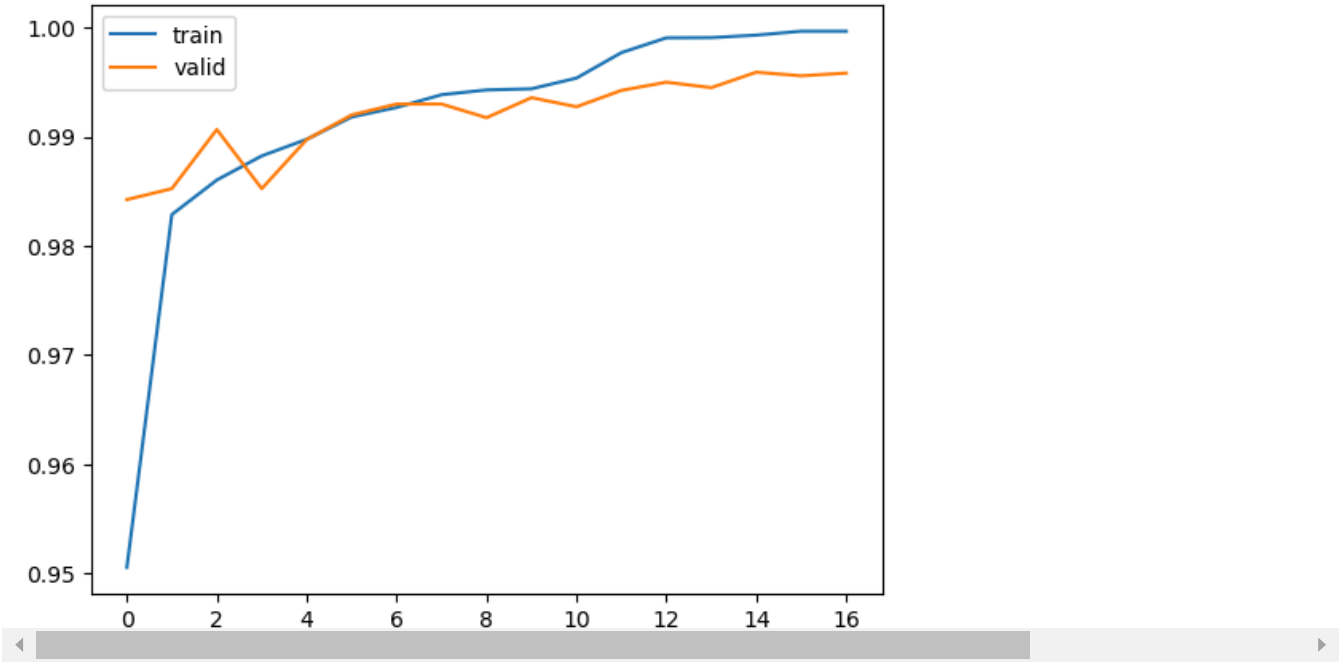
```
import matplotlib.pyplot as plt
%matplotlib inline

# 검증 데이터와 학습 데이터의 학습 과정 그래프
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='valid')
plt.legend()

# 성능 평가 테스트
test_loss, test_acc = model.evaluate(x_test, y_test)

print('Test loss : ', test_loss)
print('Test accuracy : ', test_acc)
```

```
313/313 [==============================] - 8s 25ms/step - loss: 0.0199 - accuracy: 0.9957
[0.019850514829158783, 0.9957000017166138]
```



코딩을 시작하거나 AI로 코드를 생성하세요.