MNIST - Categorical Classification

> ## Convolutional Neural Network

```
import warnings
warnings.filterwarnings('ignore')
```

- import Tensorflow

```
import tensorflow

tensorflow.__version__
```

```
'2.5.0'
```

I. MNIST Data_Set Load

```
from tensorflow.keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

II. Data Preprocessing

> ## 1) Reshape and Normalization

- reshape

```
X_train = X_train.reshape((60000,  28, 28, 1))
X_test = X_test.reshape((10000,  28, 28, 1))
```

- Normalization

```
X_train = X_train.astype(float) / 255
X_test = X_test.astype(float) / 255
```

> ## 2) One Hot Encoding

```
from tensorflow.keras.utils import to_categorical

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

III. MNIST Keras Modeling

> ## 1) Model Define

- Feature Extraction Layer

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import layers

model = models.Sequential()
model.add(layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPool2D(pool_size=(2,2)))
model.add(layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
model.add(layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 64)          36928     
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

- Classification Layer

```
model.add(layers.Flatten())
model.add(layers.Dense(units=64, activation='relu'))
model.add(layers.Dense(units=10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 64)          36928     
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                36928     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650       
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

2) Model Compile

- 모델 학습방법 설정

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

3) Model Fit

- 약 5분

```
%%time

Hist_mnist = model.fit(X_train, y_train,
                       epochs = 100,
                       batch_size = 128,
                       validation_split = 0.2)
```

```
Epoch 73/100
375/375 [==============================] - 2s 6ms/step - loss: 1.2659e-04 - accuracy: 1.0000 - val_loss: 0.1817 - val_accuracy: 0.9912
Epoch 74/100
375/375 [==============================] - 2s 6ms/step - loss: 0.0016 - accuracy: 0.9997 - val_loss: 0.1379 - val_accuracy: 0.9912
Epoch 75/100
375/375 [==============================] - 2s 6ms/step - loss: 0.0019 - accuracy: 0.9997 - val_loss: 0.1601 - val_accuracy: 0.9923
Epoch 76/100
375/375 [==============================] - 2s 6ms/step - loss: 3.5105e-04 - accuracy: 0.9999 - val_loss: 0.1681 - val_accuracy: 0.9914
Epoch 77/100
375/375 [==============================] - 2s 6ms/step - loss: 9.3833e-04 - accuracy: 0.9999 - val_loss: 0.1567 - val_accuracy: 0.9923
Epoch 78/100
375/375 [==============================] - 2s 6ms/step - loss: 5.7310e-04 - accuracy: 0.9999 - val_loss: 0.2105 - val_accuracy: 0.9897
Epoch 79/100
375/375 [==============================] - 2s 6ms/step - loss: 7.4071e-04 - accuracy: 0.9999 - val_loss: 0.1531 - val_accuracy: 0.9918
Epoch 80/100
375/375 [==============================] - 2s 6ms/step - loss: 5.4836e-04 - accuracy: 0.9999 - val_loss: 0.1481 - val_accuracy: 0.9916
Epoch 81/100
375/375 [==============================] - 2s 6ms/step - loss: 1.9263e-04 - accuracy: 0.9999 - val_loss: 0.1624 - val_accuracy: 0.9915
Epoch 82/100

375/375 [==============================] - 2s 5ms/step - loss: 0.0014 - accuracy: 0.9998 - val_loss: 0.1712 - val_accuracy: 0.9918
Epoch 83/100
375/375 [==============================] - 2s 6ms/step - loss: 1.5336e-04 - accuracy: 0.9999 - val_loss: 0.1516 - val_accuracy: 0.9917
Epoch 84/100
375/375 [==============================] - 2s 6ms/step - loss: 5.7159e-04 - accuracy: 0.9999 - val_loss: 0.1784 - val_accuracy: 0.9917
Epoch 85/100
375/375 [==============================] - 2s 5ms/step - loss: 9.2548e-04 - accuracy: 0.9998 - val_loss: 0.1663 - val_accuracy: 0.9920
Epoch 86/100
375/375 [==============================] - 2s 6ms/step - loss: 3.2150e-04 - accuracy: 0.9999 - val_loss: 0.1579 - val_accuracy: 0.9921
Epoch 87/100
375/375 [==============================] - 2s 5ms/step - loss: 1.8501e-04 - accuracy: 1.0000 - val_loss: 0.1659 - val_accuracy: 0.9922
Epoch 88/100
375/375 [==============================] - 2s 5ms/step - loss: 0.0013 - accuracy: 0.9999 - val_loss: 0.1795 - val_accuracy: 0.9919
Epoch 89/100
375/375 [==============================] - 2s 5ms/step - loss: 5.3789e-04 - accuracy: 0.9999 - val_loss: 0.1779 - val_accuracy: 0.9916
Epoch 90/100
375/375 [==============================] - 2s 5ms/step - loss: 0.0012 - accuracy: 0.9998 - val_loss: 0.2049 - val_accuracy: 0.9923
Epoch 91/100
375/375 [==============================] - 2s 5ms/step - loss: 0.0013 - accuracy: 0.9997 - val_loss: 0.1701 - val_accuracy: 0.9924
Epoch 92/100
375/375 [==============================] - 2s 5ms/step - loss: 6.8966e-04 - accuracy: 0.9999 - val_loss: 0.1729 - val_accuracy: 0.9923
Epoch 93/100
375/375 [==============================] - 2s 5ms/step - loss: 3.8062e-04 - accuracy: 0.9999 - val_loss: 0.1753 - val_accuracy: 0.9924
Epoch 94/100
375/375 [==============================] - 2s 6ms/step - loss: 0.0011 - accuracy: 0.9998 - val_loss: 0.1992 - val_accuracy: 0.9918
Epoch 95/100
375/375 [==============================] - 2s 6ms/step - loss: 0.0010 - accuracy: 0.9998 - val_loss: 0.1869 - val_accuracy: 0.9923
Epoch 96/100
375/375 [==============================] - 2s 5ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.2045 - val_accuracy: 0.9907
Epoch 97/100
375/375 [==============================] - 2s 6ms/step - loss: 0.0014 - accuracy: 0.9998 - val_loss: 0.1837 - val_accuracy: 0.9916
Epoch 98/100
375/375 [==============================] - 2s 5ms/step - loss: 0.0012 - accuracy: 0.9998 - val_loss: 0.1819 - val_accuracy: 0.9918
Epoch 99/100
375/375 [==============================] - 2s 5ms/step - loss: 0.0011 - accuracy: 0.9997 - val_loss: 0.1960 - val_accuracy: 0.9908
Epoch 100/100
375/375 [==============================] - 2s 5ms/step - loss: 2.4916e-04 - accuracy: 0.9999 - val_loss: 0.1807 - val_accuracy: 0.9920
CPU times: user 3min 38s, sys: 18.1 s, total: 3min 56s
Wall time: 4min 22s
```

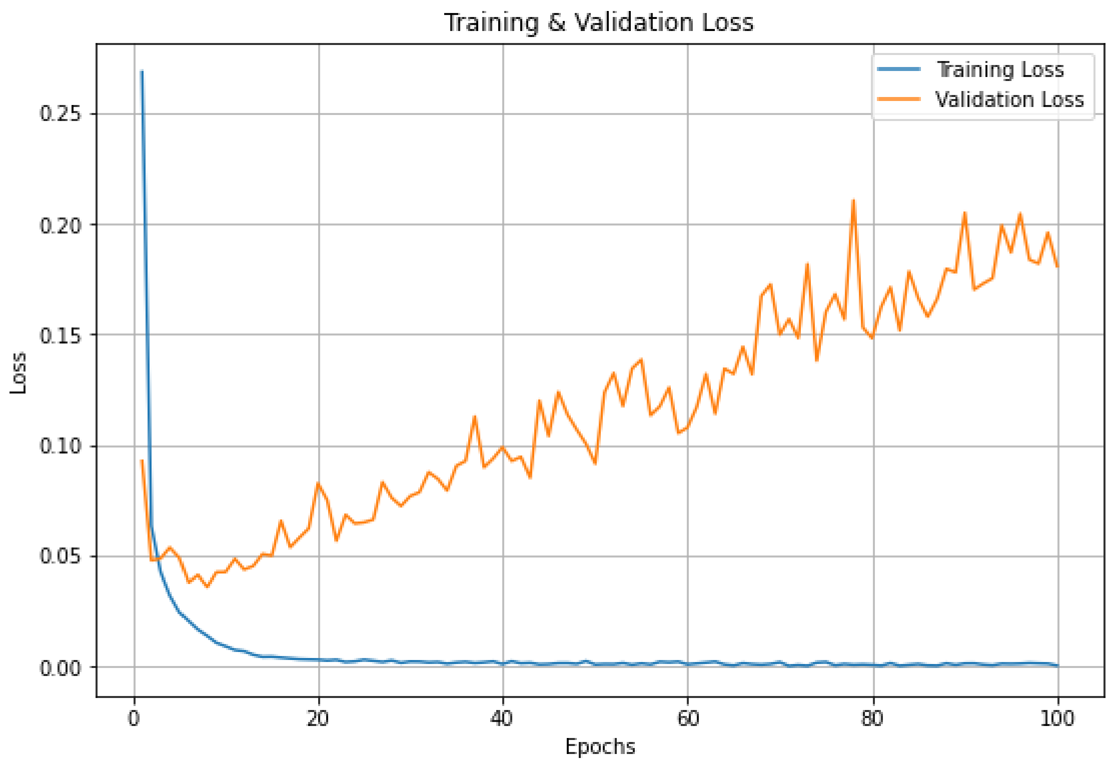
4) 학습 결과 시각화

- Loss Visualization

```
import matplotlib.pyplot as plt

epochs = range(1, len(Hist_mnist.history['loss']) + 1)

plt.figure(figsize = (9, 6))
plt.plot(epochs, Hist_mnist.history['loss'])
plt.plot(epochs, Hist_mnist.history['val_loss'])
# plt.ylim(0, 0.4)
plt.title('Training & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.grid()
plt.show()
```



5) Model Evaluate

- Loss & Accuracy

```
loss, accuracy = model.evaluate(X_test, y_test)

print('Loss = {:.5f}'.format(loss))
print('Accuracy = {:.5f}'.format(accuracy))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1758 - accuracy: 0.9918
Loss = 0.17580
Accuracy = 0.99180
```

#

#

#

The End

#

#

#

✓ 1초 오전 8:56에 완료됨 ● ✕