MNIST - Categorical Classification

Convolutional Neural Network

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

- import Tensorflow

```
import tensorflow

tensorflow.__version__
```

```
'2.6.0'
```

I. MNIST Data_Set Load

```
from tensorflow.keras.datasets import mnist

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

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

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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (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 1/100
375/375 [==============================] - 35s 16ms/step - loss: 0.2763 - accuracy: 0.9141 - val_loss: 0.0753 - val_accuracy: 0.9780
Epoch 2/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0664 - accuracy: 0.9800 - val_loss: 0.0509 - val_accuracy: 0.9858
Epoch 3/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0428 - accuracy: 0.9868 - val_loss: 0.0444 - val_accuracy: 0.9869
Epoch 4/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0318 - accuracy: 0.9896 - val_loss: 0.0376 - val_accuracy: 0.9889
Epoch 5/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0239 - accuracy: 0.9924 - val_loss: 0.0393 - val_accuracy: 0.9887
Epoch 6/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0189 - accuracy: 0.9939 - val_loss: 0.0443 - val_accuracy: 0.9873
Epoch 7/100
375/375 [==============================] - 6s 15ms/step - loss: 0.0156 - accuracy: 0.9945 - val_loss: 0.0483 - val_accuracy: 0.9873
Epoch 8/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0130 - accuracy: 0.9956 - val_loss: 0.0399 - val_accuracy: 0.9872
Epoch 9/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0104 - accuracy: 0.9968 - val_loss: 0.0352 - val_accuracy: 0.9909
Epoch 10/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0092 - accuracy: 0.9971 - val_loss: 0.0377 - val_accuracy: 0.9912
Epoch 11/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0072 - accuracy: 0.9976 - val_loss: 0.0488 - val_accuracy: 0.9893
Epoch 12/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0062 - accuracy: 0.9977 - val_loss: 0.0455 - val_accuracy: 0.9902
Epoch 13/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0058 - accuracy: 0.9981 - val_loss: 0.0657 - val_accuracy: 0.9882
Epoch 14/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0050 - accuracy: 0.9984 - val_loss: 0.0490 - val_accuracy: 0.9904
Epoch 15/100
375/375 [==============================] - 5s 15ms/step - loss: 0.0042 - accuracy: 0.9985 - val_loss: 0.0458 - val_accuracy: 0.9923
Epoch 16/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0031 - accuracy: 0.9989 - val_loss: 0.0613 - val_accuracy: 0.9910
Epoch 17/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0038 - accuracy: 0.9987 - val_loss: 0.0505 - val_accuracy: 0.9918
Epoch 18/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0024 - accuracy: 0.9992 - val_loss: 0.0504 - val_accuracy: 0.9921
Epoch 19/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0028 - accuracy: 0.9990 - val_loss: 0.0607 - val_accuracy: 0.9908
Epoch 20/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0032 - accuracy: 0.9990 - val_loss: 0.0572 - val_accuracy: 0.9916
Epoch 21/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0025 - accuracy: 0.9993 - val_loss: 0.0636 - val_accuracy: 0.9909
Epoch 22/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0020 - accuracy: 0.9994 - val_loss: 0.0672 - val_accuracy: 0.9910
Epoch 23/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0022 - accuracy: 0.9992 - val_loss: 0.0616 - val_accuracy: 0.9923
Epoch 24/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0022 - accuracy: 0.9994 - val_loss: 0.0785 - val_accuracy: 0.9897
Epoch 25/100
375/375 [==============================] - 5s 13ms/step - loss: 0.0023 - accuracy: 0.9993 - val_loss: 0.0798 - val_accuracy: 0.9908
Epoch 26/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0019 - accuracy: 0.9994 - val_loss: 0.0817 - val_accuracy: 0.9913
Epoch 27/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0015 - accuracy: 0.9997 - val_loss: 0.0833 - val_accuracy: 0.9908
Epoch 28/100
375/375 [==============================] - 5s 13ms/step - loss: 0.0014 - accuracy: 0.9997 - val_loss: 0.0904 - val_accuracy: 0.9912
Epoch 29/100
375/375 [==============================] - 5s 14ms/step - loss: 0.0017 - accuracy: 0.9995 - val_loss: 0.0927 - val_accuracy: 0.9903
Epoch 30/100
```

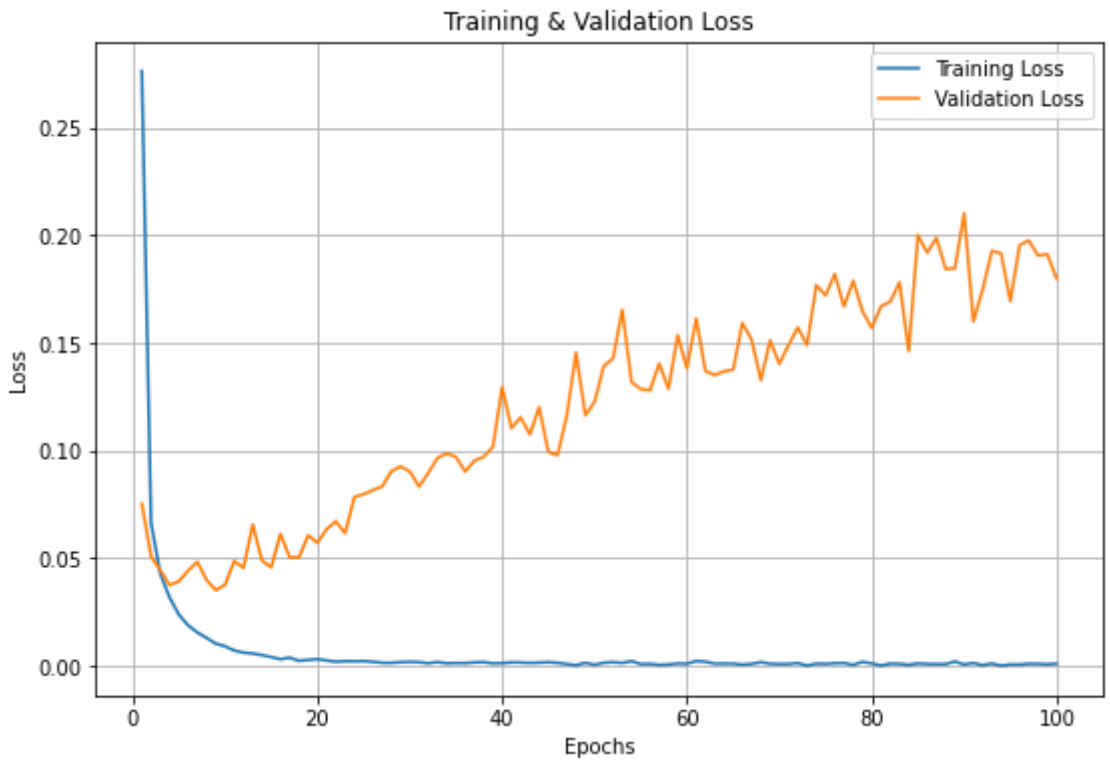
4) 학습 결과 시각화

- 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 4ms/step - loss: 0.1330 - accuracy: 0.9919
Loss = 0.13302
Accuracy = 0.99190
```

#

#

#

The End

#

#

#