```
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
```

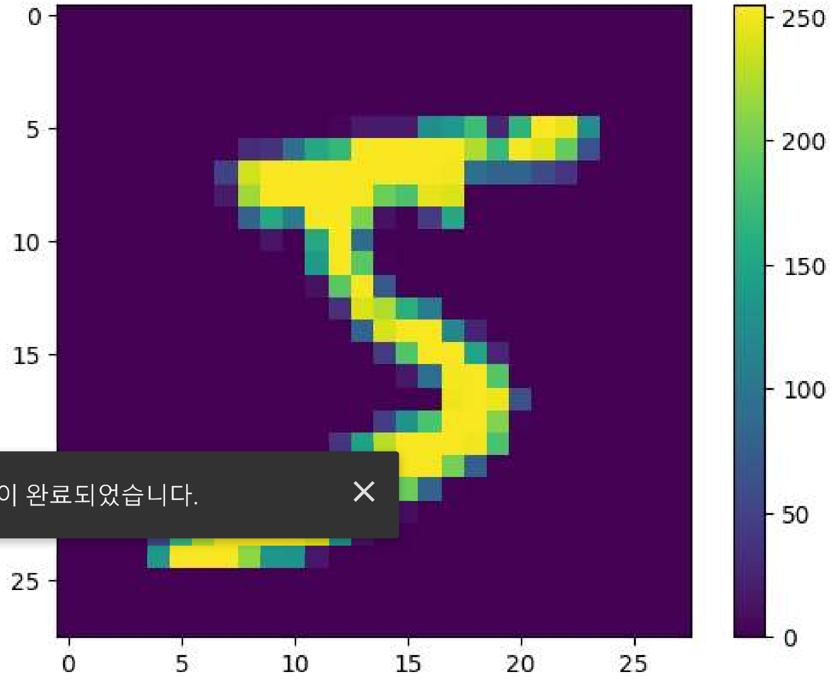
```
# mnist의 data인 (x_train, y_train), (x_test, y_test)를 입력 받는다
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

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

```
# 다음의 type 정보들을 확인해 보자
type(x_train), type(y_train), type(x_test), type(y_test), x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
(numpy.ndarray,
 numpy.ndarray,
 numpy.ndarray,
 numpy.ndarray,
 (60000, 28, 28),
 (60000,),
 (10000, 28, 28),
 (10000,))
```

```
plt.figure()
plt.imshow(x_train[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



저장이 완료되었습니다. ✕

```
# input : Feature, Atrribute => Flatten, Scailing
x_train = x_train.reshape((60000, 28 * 28))
x_test = x_test.reshape((10000, 28 * 28))
x_train = x_train/255
x_test = x_test/255
```

```
# x_train과 x_test의 모양을 살펴보고, x_train과 x_test의 값들이 정규화 [0 ~ 1] 사이의 값이 되었는지 확인하는 셀
x_train.shape, x_test.shape, x_train.max(), x_test.max()
```

```
((60000, 784), (10000, 784), 1.0, 1.0)
```

```
y_train[0 : 10], y_train.shape # 기본 정보는 다음과 같이 되어 있다. => one-hot encoding으로 변경해준다
```

```
(array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8), (60000,))
```

```
# Label : one hot encoding
y_train = np.eye(y_train.max() + 1)[y_train] # 1. numpy의 eye를 사용해서 one-hot encoding : np.eye(최대값 + 1)[기본 l
```

```
from tensorflow.keras.utils import to_categorical
y_test = to_categorical(y=y_test, num_classes = 10) # 2. tensorflow.keras.utils를 사용해서 one-hot encoding: 진행
```

```
y_train[0 : 10], y_train.shape, y_test[0:10], y_test.shape
```

```
(array([[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.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]]),
 (60000, 10),
 array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0., 0., 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., 1., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32),
 (10000, 10))
```

모델 설계

```
from tensorflow.keras import models, layers
# 3 Layer :
# input layer feature count : 784(28*28)
# hidden layer perceptron : 512, 'relu'
                                                  x'
                                                 ls")
model.add(layers.Input(28 * 28, ))
model.add(layers.Dense(units=512, activation='relu', name='hidden'))
model.add(layers.Dense(units=10, activation='softmax', name='output'))
```

저장이 완료되었습니다. ✕

```
model.summary() # 모델을 확인할 수 있게 도와주는 summary 함수
```

```
Model: "mnist_cls"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden (Dense)              (None, 512)               401920

 output (Dense)              (None, 10)                5130

=================================================================
Total params: 407050 (1.55 MB)
Trainable params: 407050 (1.55 MB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

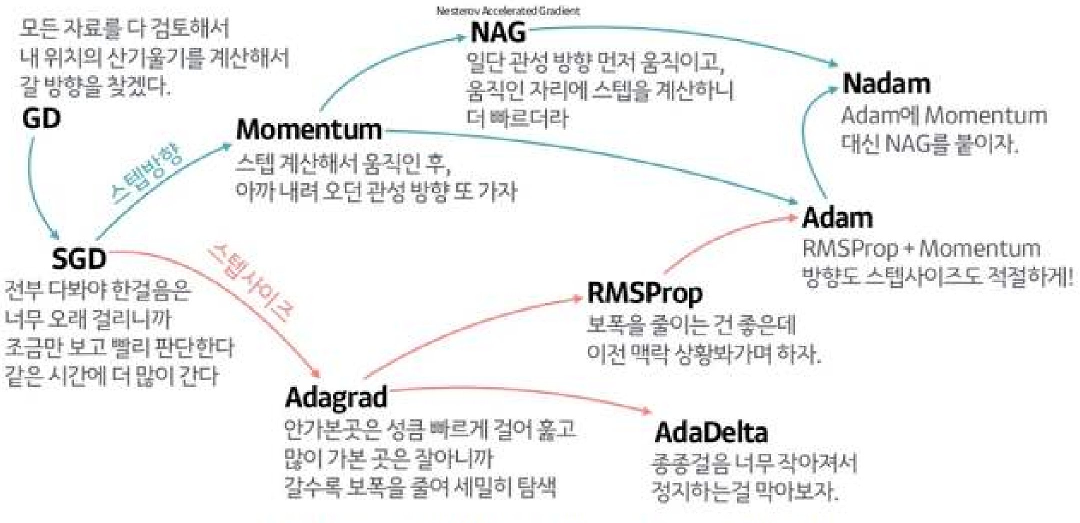
```
# Compile model
# Setting optimizer, loss function, metrics
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Optimizer

2. Output 값이 목표로 하는 타겟값과 가까워지도록 Error_Function(= loss function)을 설정하고 Error 값이 줄어드는 방향으로 학습시키기 위해 각 Weight와 gradient 값을 계산
3. weight들을 다시 설정
4. 최적의 weight 모델을 찾는다

이 (2~4) 과정이 Optimization 과정, 최적화 과정

Optimizer 종류



출처 : https://www.slideshare.net/yongho/ss-79607172

Loss Function(손실함수)

사용자가 원하는 출력값의 오차를 의미

1. MSE(Mean Squared Error)

저장이 완료되었습니다. ✕

3. Binary Crossentropy
4. Categorical Crossentropy(레이블 클래스가 2개 초과일 경우)
5. Focal loss

Metrics

1. Classification Metrics(분류 메트릭) 1.1. Accuracy(정확도) 1.2. Logloss 1.3. AUC ROC(Area Under Curve)
2. Regression Metric(회귀 메트릭) 2.1 MSE(Mean Squared Error) 2.2 RMSE(Root Mean Squared Error) 2.3 R-squared 2.4 MAE(Mean Absolute Error)

```
history = model.fit(x=x_train, y=y_train,
                    batch_size=100, epochs=30,
                    validation_split=0.2)
                    validation_split=0.2)
```

```
=====================] - 5s 11ms/step - loss: 0.1146 - accuracy: 0.9671 - val_loss: 0.1087 - val_accuracy: 0
=====================] - 5s 11ms/step - loss: 0.0745 - accuracy: 0.9784 - val_loss: 0.0914 - val_accuracy: 0
=====================] - 6s 13ms/step - loss: 0.0521 - accuracy: 0.9849 - val_loss: 0.0781 - val_accuracy: 0
=====================] - 5s 10ms/step - loss: 0.0374 - accuracy: 0.9894 - val_loss: 0.0814 - val_accuracy: 0
=====================] - 6s 13ms/step - loss: 0.0267 - accuracy: 0.9929 - val_loss: 0.0792 - val_accuracy: 0
=====================] - 5s 11ms/step - loss: 0.0205 - accuracy: 0.9944 - val_loss: 0.0794 - val_accuracy: 0
=====================] - 6s 13ms/step - loss: 0.0160 - accuracy: 0.9957 - val_loss: 0.0770 - val_accuracy: 0
=====================] - 5s 11ms/step - loss: 0.0105 - accuracy: 0.9977 - val_loss: 0.0824 - val_accuracy: 0
=====================] - 5s 10ms/step - loss: 0.0086 - accuracy: 0.9981 - val_loss: 0.0799 - val_accuracy: 0
=====================] - 6s 13ms/step - loss: 0.0094 - accuracy: 0.9978 - val_loss: 0.0818 - val_accuracy: 0
=====================] - 5s 10ms/step - loss: 0.0068 - accuracy: 0.9984 - val_loss: 0.0889 - val_accuracy: 0
=====================] - 6s 12ms/step - loss: 0.0080 - accuracy: 0.9979 - val_loss: 0.0852 - val_accuracy: 0
=====================] - 5s 10ms/step - loss: 0.0071 - accuracy: 0.9980 - val_loss: 0.0909 - val_accuracy: 0
=====================] - 8s 16ms/step - loss: 0.0050 - accuracy: 0.9986 - val_loss: 0.0856 - val_accuracy: 0
=====================] - 5s 10ms/step - loss: 0.0019 - accuracy: 0.9997 - val_loss: 0.0874 - val_accuracy: 0
=====================] - 5s 10ms/step - loss: 0.0015 - accuracy: 0.9998 - val_loss: 0.0896 - val_accuracy: 0
=====================] - 6s 13ms/step - loss: 0.0082 - accuracy: 0.9971 - val_loss: 0.0912 - val_accuracy: 0
=====================] - 5s 10ms/step - loss: 0.0090 - accuracy: 0.9969 - val_loss: 0.0973 - val_accuracy: 0
=====================] - 6s 12ms/step - loss: 0.0030 - accuracy: 0.9992 - val_loss: 0.0875 - val_accuracy: 0
=====================] - 5s 11ms/step - loss: 6.0187e-04 - accuracy: 1.0000 - val_loss: 0.0914 - val_accurac
=====================] - 5s 11ms/step - loss: 2.3088e-04 - accuracy: 1.0000 - val_loss: 0.0904 - val_accurac
=====================] - 6s 12ms/step - loss: 1.6892e-04 - accuracy: 1.0000 - val_loss: 0.0907 - val_accurac
=====================] - 5s 11ms/step - loss: 1.3664e-04 - accuracy: 1.0000 - val_loss: 0.0921 - val_accurac
=====================] - 6s 13ms/step - loss: 1.1458e-04 - accuracy: 1.0000 - val_loss: 0.0928 - val_accurac
                                    tep - loss: 0.0178 - accuracy: 0.9945 - val_loss: 0.1048 - val_accuracy: 0
                                    tep - loss: 0.0044 - accuracy: 0.9986 - val_loss: 0.0976 - val_accuracy: 0
=====================] - 5s 11ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 0.0990 - val_accuracy: 0
=====================] - 5s 11ms/step - loss: 3.9722e-04 - accuracy: 1.0000 - val_loss: 0.0941 - val_accurac
=====================] - 6s 12ms/step - loss: 1.5759e-04 - accuracy: 1.0000 - val_loss: 0.0950 - val_accurac
```

저장이 완료되었습니다. ✕

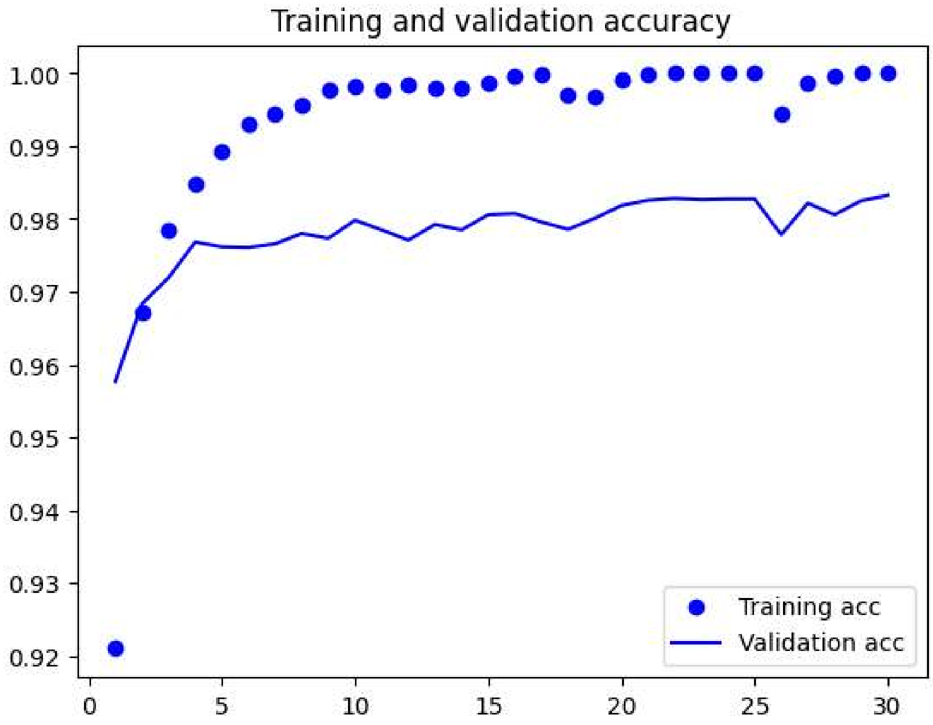
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc)+1)
```

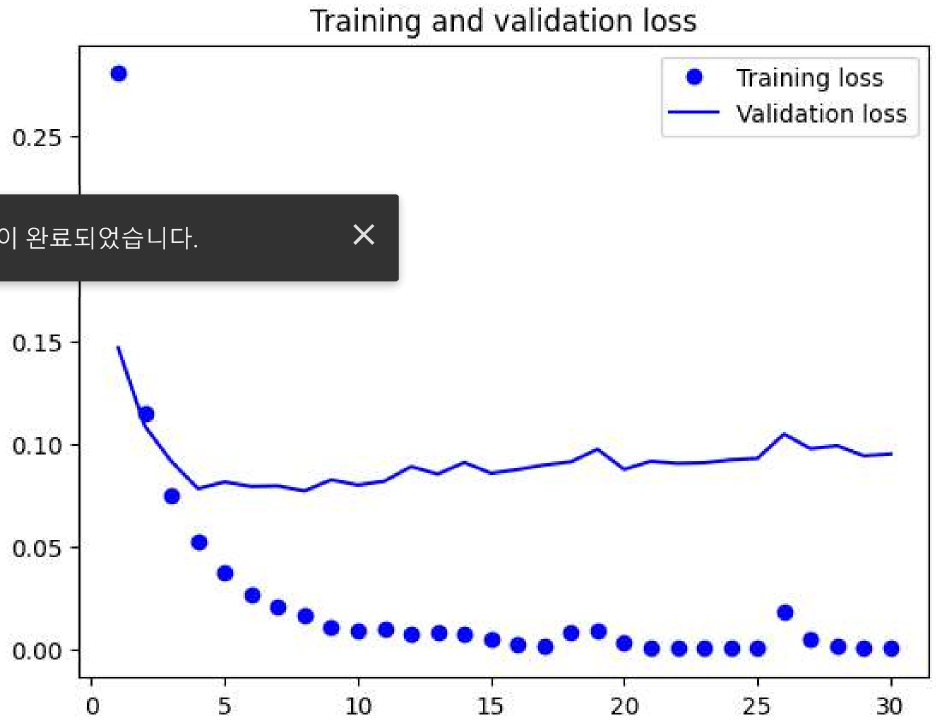
```
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.show()
```



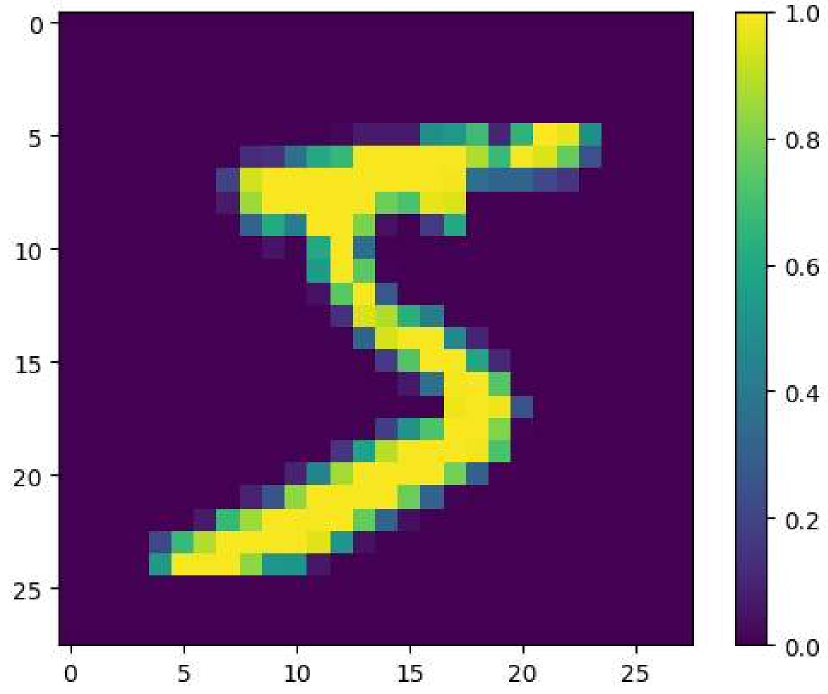
```
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



저장이 완료되었습니다. ✕

```
plt.figure()
plt.imshow(x_train[0].reshape(28, 28))
plt.imshow(x_train[0].reshape(28, 28))
plt.colorbar()
plt.grid(False)
plt.show()
```



저장이 완료되었습니다. ×

Colab 유료 제품 - 여기에서 계약 취소

✓ 1초 오후 6:11에 완료됨 ● ×