텐서플로가 제공하는 데이터셋 확인

MNIST 필기 숫자 데이터셋

CIFAR-10 자연영상 데이터셋

```
import tensorflow as tf
import tensorflow.keras.datasets as ds
import matplotlib.pyplot as plt

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

```
plt.figure(figsize=(24,3))
plt.suptitle('MNIST',fontsize=30)
```

```
Text(0.5, 0.98, 'MNIST')
<Figure size 1728x216 with 0 Axes>
```

```
for i in range(10):
    plt.subplot(1,10,i+1)
    plt.imshow(x_train[i],cmap='gray')
    plt.xticks([]); plt.yticks([])
    plt.title(str(y_train[i]),fontsize=30)
```


```
(x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
```

```
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
```

```
class_names=['airplane','car','bird','cat','deer','dog','frog','horse','ship','truck']

plt.figure(figsize=(24,3))
plt.suptitle('CIFAR-10',fontsize=30)
```

```
Text(0.5, 0.98, 'CIFAR-10')
<Figure size 1728x216 with 0 Axes>
```

```
for i in range(10):
    plt.subplot(1,10,i+1)
    plt.imshow(x_train[i])
    plt.xticks([]); plt.yticks([])
    plt.title(class_names[y_train[i,0]],fontsize=12)
```

필기 숫자 인식

다층 퍼셉트론으로 MNIST 인식하기(SGD 옵티마이저)

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()

x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

mlp=Sequential()
mlp.add(Dense(units=512,activation='tanh',input_shape=(784,)))
mlp.add(Dense(units=10,activation='softmax'))

mlp.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])

mlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
```

```
469/469 - 5s - loss: 0.0259 - accuracy: 0.8569 - val_loss: 0.0245 - val_accuracy: 0.8666 - 5s/epoch - 10ms/step
Epoch 23/50
469/469 - 4s - loss: 0.0253 - accuracy: 0.8593 - val_loss: 0.0240 - val_accuracy: 0.8691 - 4s/epoch - 8ms/step
Epoch 24/50
469/469 - 4s - loss: 0.0248 - accuracy: 0.8612 - val_loss: 0.0236 - val_accuracy: 0.8704 - 4s/epoch - 9ms/step
Epoch 25/50
469/469 - 5s - loss: 0.0244 - accuracy: 0.8632 - val_loss: 0.0231 - val_accuracy: 0.8720 - 5s/epoch - 10ms/step
Epoch 26/50
469/469 - 4s - loss: 0.0240 - accuracy: 0.8647 - val_loss: 0.0227 - val_accuracy: 0.8731 - 4s/epoch - 9ms/step
```

```
469/469 - 5s - loss: 0.0190 - accuracy: 0.8868 - val_loss: 0.0179 - val_accuracy: 0.8950 - 5s/epoch - 10ms/step
Epoch 49/50
469/469 - 4s - loss: 0.0189 - accuracy: 0.8873 - val_loss: 0.0178 - val_accuracy: 0.8957 - 4s/epoch - 9ms/step
Epoch 50/50
469/469 - 4s - loss: 0.0187 - accuracy: 0.8881 - val_loss: 0.0177 - val_accuracy: 0.8958 - 4s/epoch - 9ms/step
<keras.callbacks.History at 0x7f39a0c0a1f0>
```

```
res=mlp.evaluate(x_test,y_test,verbose=0)
print('정확률=',res[1]*100)
```

```
정확률= 89.5799994468689
```

Adam 옵티마이저를 사용하여 성능 향상

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()

x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

mlp=Sequential()
mlp.add(Dense(units=512,activation='tanh',input_shape=(784,)))
mlp.add(Dense(units=10,activation='softmax'))

mlp.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])

mlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
```

```
Epoch 39/50
469/469 - 6s - loss: 2.5365e-04 - accuracy: 0.9986 - val_loss: 0.0028 - val_accuracy: 0.9824 - 6s/epoch - 13ms/step
Epoch 40/50
469/469 - 5s - loss: 2.0081e-04 - accuracy: 0.9989 - val_loss: 0.0030 - val_accuracy: 0.9817 - 5s/epoch - 11ms/step
Epoch 41/50
469/469 - 6s - loss: 2.7836e-04 - accuracy: 0.9984 - val_loss: 0.0032 - val_accuracy: 0.9786 - 6s/epoch - 12ms/step
Epoch 42/50
469/469 - 5s - loss: 2.6401e-04 - accuracy: 0.9986 - val_loss: 0.0028 - val_accuracy: 0.9822 - 5s/epoch - 11ms/step
Epoch 43/50
469/469 - 6s - loss: 2.3647e-04 - accuracy: 0.9987 - val_loss: 0.0031 - val_accuracy: 0.9810 - 6s/epoch - 13ms/step
Epoch 44/50
469/469 - 5s - loss: 1.8913e-04 - accuracy: 0.9991 - val_loss: 0.0028 - val_accuracy: 0.9831 - 5s/epoch - 11ms/step
Epoch 45/50
469/469 - 6s - loss: 1.5575e-04 - accuracy: 0.9992 - val_loss: 0.0028 - val_accuracy: 0.9837 - 6s/epoch - 12ms/step
Epoch 46/50
469/469 - 5s - loss: 2.6059e-04 - accuracy: 0.9985 - val_loss: 0.0047 - val_accuracy: 0.9713 - 5s/epoch - 10ms/step
Epoch 47/50
469/469 - 5s - loss: 4.1481e-04 - accuracy: 0.9976 - val_loss: 0.0031 - val_accuracy: 0.9800 - 5s/epoch - 11ms/step
Epoch 48/50
469/469 - 5s - loss: 1.8802e-04 - accuracy: 0.9990 - val_loss: 0.0028 - val_accuracy: 0.9821 - 5s/epoch - 11ms/step
Epoch 49/50
469/469 - 5s - loss: 1.5881e-04 - accuracy: 0.9992 - val_loss: 0.0028 - val_accuracy: 0.9830 - 5s/epoch - 10ms/step
Epoch 50/50
469/469 - 6s - loss: 1.5735e-04 - accuracy: 0.9992 - val_loss: 0.0029 - val_accuracy: 0.9825 - 6s/epoch - 13ms/step
<keras.callbacks.History at 0x7f39a0c24850>
```

```
res=mlp.evaluate(x_test,y_test,verbose=0)
print('정확률=',res[1]*100)
```

```
정확률= 98.25000166893005
```

성능 시각화

SGD와 Adam 성능을 그래프로 비교

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()

x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

mlp_sgd=Sequential()
mlp_sgd.add(Dense(units=512,activation='tanh',input_shape=(784,)))
mlp_sgd.add(Dense(units=10,activation='softmax'))

mlp_sgd.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])

hist_sgd=mlp_sgd.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
```

```
Epoch 27/50
469/469 - 5s - loss: 0.0233 - accuracy: 0.8694 - val_loss: 0.0222 - val_accuracy: 0.8792 - 5s/epoch - 10ms/step
Epoch 28/50
469/469 - 4s - loss: 0.0230 - accuracy: 0.8705 - val_loss: 0.0219 - val_accuracy: 0.8801 - 4s/epoch - 9ms/step
Epoch 29/50
469/469 - 4s - loss: 0.0227 - accuracy: 0.8721 - val_loss: 0.0216 - val_accuracy: 0.8814 - 4s/epoch - 9ms/step
Epoch 30/50
469/469 - 5s - loss: 0.0224 - accuracy: 0.8733 - val_loss: 0.0213 - val_accuracy: 0.8827 - 5s/epoch - 10ms/step
Epoch 31/50
469/469 - 4s - loss: 0.0221 - accuracy: 0.8744 - val_loss: 0.0210 - val_accuracy: 0.8837 - 4s/epoch - 8ms/step
Epoch 32/50
469/469 - 4s - loss: 0.0218 - accuracy: 0.8755 - val_loss: 0.0207 - val_accuracy: 0.8845 - 4s/epoch - 9ms/step
Epoch 33/50
469/469 - 5s - loss: 0.0215 - accuracy: 0.8765 - val_loss: 0.0205 - val_accuracy: 0.8856 - 5s/epoch - 10ms/step
Epoch 34/50
469/469 - 4s - loss: 0.0213 - accuracy: 0.8776 - val_loss: 0.0203 - val_accuracy: 0.8866 - 4s/epoch - 9ms/step
Epoch 35/50
469/469 - 5s - loss: 0.0211 - accuracy: 0.8781 - val_loss: 0.0201 - val_accuracy: 0.8877 - 5s/epoch - 10ms/step
Epoch 36/50
469/469 - 4s - loss: 0.0209 - accuracy: 0.8789 - val_loss: 0.0199 - val_accuracy: 0.8881 - 4s/epoch - 9ms/step
Epoch 37/50
469/469 - 4s - loss: 0.0207 - accuracy: 0.8796 - val_loss: 0.0197 - val_accuracy: 0.8891 - 4s/epoch - 9ms/step
Epoch 38/50
469/469 - 5s - loss: 0.0205 - accuracy: 0.8807 - val_loss: 0.0195 - val_accuracy: 0.8902 - 5s/epoch - 10ms/step
Epoch 39/50
469/469 - 4s - loss: 0.0203 - accuracy: 0.8815 - val_loss: 0.0193 - val_accuracy: 0.8904 - 4s/epoch - 9ms/step
Epoch 40/50
469/469 - 4s - loss: 0.0201 - accuracy: 0.8823 - val_loss: 0.0191 - val_accuracy: 0.8909 - 4s/epoch - 9ms/step
Epoch 41/50
469/469 - 5s - loss: 0.0199 - accuracy: 0.8832 - val_loss: 0.0190 - val_accuracy: 0.8918 - 5s/epoch - 10ms/step
Epoch 42/50
469/469 - 4s - loss: 0.0198 - accuracy: 0.8840 - val_loss: 0.0188 - val_accuracy: 0.8930 - 4s/epoch - 8ms/step
Epoch 43/50
469/469 - 4s - loss: 0.0196 - accuracy: 0.8846 - val_loss: 0.0187 - val_accuracy: 0.8932 - 4s/epoch - 9ms/step
Epoch 44/50
469/469 - 5s - loss: 0.0195 - accuracy: 0.8852 - val_loss: 0.0185 - val_accuracy: 0.8938 - 5s/epoch - 10ms/step
Epoch 45/50
469/469 - 5s - loss: 0.0193 - accuracy: 0.8858 - val_loss: 0.0184 - val_accuracy: 0.8938 - 5s/epoch - 10ms/step
Epoch 46/50
469/469 - 5s - loss: 0.0192 - accuracy: 0.8865 - val_loss: 0.0183 - val_accuracy: 0.8951 - 5s/epoch - 10ms/step
Epoch 47/50
469/469 - 4s - loss: 0.0191 - accuracy: 0.8870 - val_loss: 0.0181 - val_accuracy: 0.8951 - 4s/epoch - 8ms/step
Epoch 48/50
469/469 - 4s - loss: 0.0189 - accuracy: 0.8876 - val_loss: 0.0180 - val_accuracy: 0.8955 - 4s/epoch - 8ms/step
Epoch 49/50
469/469 - 5s - loss: 0.0188 - accuracy: 0.8880 - val_loss: 0.0179 - val_accuracy: 0.8959 - 5s/epoch - 10ms/step
Epoch 50/50
469/469 - 4s - loss: 0.0187 - accuracy: 0.8885 - val_loss: 0.0178 - val_accuracy: 0.8961 - 4s/epoch - 9ms/step
```

```
print('SGD 정확률=',mlp_sgd.evaluate(x_test,y_test,verbose=0)[1]*100)
```

```
SGD 정확률= 89.60999846458435
```

```
mlp_adam=Sequential()
mlp_adam.add(Dense(units=512,activation='tanh',input_shape=(784,)))
mlp_adam.add(Dense(units=10,activation='softmax'))

mlp_adam.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])

hist_adam=mlp_adam.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=
```

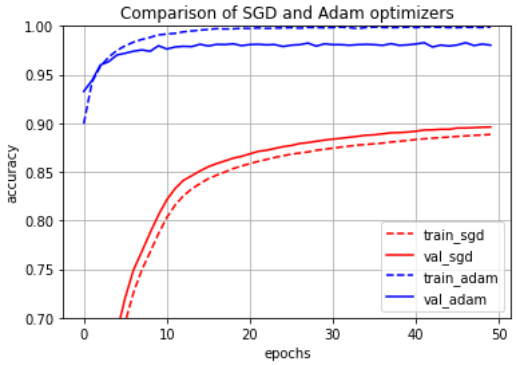
```
Epoch 34/50
469/469 - 5s - loss: 4.5890e-04 - accuracy: 0.9973 - val_loss: 0.0030 - val_accuracy: 0.9808 - 5s/epoch - 11ms/step
Epoch 35/50
469/469 - 6s - loss: 2.9185e-04 - accuracy: 0.9984 - val_loss: 0.0030 - val_accuracy: 0.9811 - 6s/epoch - 12ms/step
Epoch 36/50
469/469 - 5s - loss: 2.1683e-04 - accuracy: 0.9989 - val_loss: 0.0031 - val_accuracy: 0.9808 - 5s/epoch - 11ms/step
Epoch 37/50
469/469 - 6s - loss: 2.7260e-04 - accuracy: 0.9985 - val_loss: 0.0032 - val_accuracy: 0.9801 - 6s/epoch - 12ms/step
Epoch 38/50
469/469 - 5s - loss: 3.1379e-04 - accuracy: 0.9984 - val_loss: 0.0030 - val_accuracy: 0.9818 - 5s/epoch - 10ms/step
Epoch 39/50
469/469 - 5s - loss: 2.8668e-04 - accuracy: 0.9985 - val_loss: 0.0031 - val_accuracy: 0.9801 - 5s/epoch - 11ms/step
Epoch 40/50
469/469 - 6s - loss: 2.1850e-04 - accuracy: 0.9988 - val_loss: 0.0032 - val_accuracy: 0.9806 - 6s/epoch - 12ms/step
Epoch 41/50
469/469 - 5s - loss: 2.9311e-04 - accuracy: 0.9984 - val_loss: 0.0031 - val_accuracy: 0.9815 - 5s/epoch - 11ms/step
Epoch 42/50
469/469 - 6s - loss: 2.1058e-04 - accuracy: 0.9989 - val_loss: 0.0029 - val_accuracy: 0.9828 - 6s/epoch - 12ms/step
Epoch 43/50
469/469 - 6s - loss: 2.1498e-04 - accuracy: 0.9989 - val_loss: 0.0034 - val_accuracy: 0.9784 - 6s/epoch - 12ms/step
Epoch 44/50
469/469 - 6s - loss: 2.6037e-04 - accuracy: 0.9986 - val_loss: 0.0032 - val_accuracy: 0.9804 - 6s/epoch - 12ms/step
Epoch 45/50
469/469 - 5s - loss: 2.9674e-04 - accuracy: 0.9984 - val_loss: 0.0033 - val_accuracy: 0.9794 - 5s/epoch - 11ms/step
Epoch 46/50
469/469 - 6s - loss: 2.9257e-04 - accuracy: 0.9984 - val_loss: 0.0032 - val_accuracy: 0.9805 - 6s/epoch - 12ms/step
Epoch 47/50
469/469 - 5s - loss: 2.4312e-04 - accuracy: 0.9987 - val_loss: 0.0028 - val_accuracy: 0.9828 - 5s/epoch - 11ms/step
Epoch 48/50
469/469 - 6s - loss: 2.6606e-04 - accuracy: 0.9986 - val_loss: 0.0033 - val_accuracy: 0.9799 - 6s/epoch - 13ms/step
Epoch 49/50
469/469 - 5s - loss: 2.7457e-04 - accuracy: 0.9984 - val_loss: 0.0031 - val_accuracy: 0.9814 - 5s/epoch - 10ms/step
Epoch 50/50
469/469 - 5s - loss: 1.9454e-04 - accuracy: 0.9990 - val_loss: 0.0031 - val_accuracy: 0.9803 - 5s/epoch - 11ms/step
```

```
print('Adam 정확률=',mlp_adam.evaluate(x_test,y_test,verbose=0)[1]*100)
```

```
Adam 정확률= 98.03000092506409
```

```
import matplotlib.pyplot as plt

plt.plot(hist_sgd.history['accuracy'],'r--')
plt.plot(hist_sgd.history['val_accuracy'],'r')
plt.plot(hist_adam.history['accuracy'],'b--')
plt.plot(hist_adam.history['val_accuracy'],'b')
plt.title('Comparison of SGD and Adam optimizers')
plt.ylim((0.7,1.0))
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['train_sgd','val_sgd','train_adam','val_adam'])
plt.grid()
plt.show()
```



하이퍼 매개변수 다루기

깊은 다층 퍼셉트론으로 MNIST 인식

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()

x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

dmlp=Sequential()
dmlp.add(Dense(units=1024,activation='relu',input_shape=(784,)))
dmlp.add(Dense(units=512,activation='relu'))
dmlp.add(Dense(units=512,activation='relu'))
dmlp.add(Dense(units=10,activation='softmax'))

dmlp.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.0001),metrics=['accuracy'])

hist=dmlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
```

```
Epoch 22/50
469/469 - 19s - loss: 2.3229e-04 - accuracy: 1.0000 - val_loss: 0.0751 - val_accuracy: 0.9827 - 19s/epoch - 41ms/step
Epoch 23/50
469/469 - 18s - loss: 0.0016 - accuracy: 0.9996 - val_loss: 0.1144 - val_accuracy: 0.9756 - 18s/epoch - 37ms/step
Epoch 24/50
469/469 - 19s - loss: 0.0122 - accuracy: 0.9959 - val_loss: 0.0726 - val_accuracy: 0.9820 - 19s/epoch - 40ms/step
Epoch 25/50
```

```
469/469 - 19s - loss: 1.2692e-05 - accuracy: 1.0000 - val_loss: 0.0921 - val_accuracy: 0.9857 - 19s/epoch - 40ms/step
Epoch 47/50
469/469 - 18s - loss: 1.0229e-05 - accuracy: 1.0000 - val_loss: 0.0944 - val_accuracy: 0.9832 - 18s/epoch - 38ms/step
Epoch 48/50
469/469 - 19s - loss: 8.4136e-06 - accuracy: 1.0000 - val_loss: 0.0947 - val_accuracy: 0.9838 - 19s/epoch - 41ms/step
Epoch 49/50
469/469 - 18s - loss: 6.9131e-06 - accuracy: 1.0000 - val_loss: 0.0966 - val_accuracy: 0.9836 - 18s/epoch - 39ms/step
Epoch 50/50
469/469 - 19s - loss: 5.4257e-06 - accuracy: 1.0000 - val_loss: 0.0985 - val_accuracy: 0.9840 - 19s/epoch - 40ms/step
```

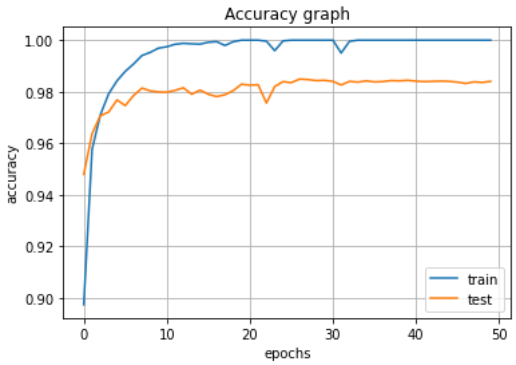
```
print('정확률=', dmlp.evaluate(x_test,y_test,verbose=0)[1]*100)
```

```
정확률= 98.4000027179718
```

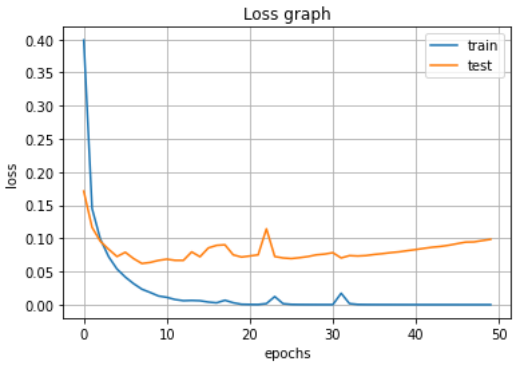
```
dmlp.save('dmlp_trained.h5')
```

```
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy graph')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['train','test'])
plt.grid()
plt.show()
```




```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss graph')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['train','test'])
plt.grid()
plt.show()
```



자연 영상 인식

깊은 다층 퍼셉트론으로 CIFAR-10 인식하기

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

(x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()

x_train=x_train.reshape(50000,3072)
x_test=x_test.reshape(10000,3072)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

dmlp=Sequential()
dmlp.add(Dense(units=1024,activation='relu',input_shape=(3072,)))
dmlp.add(Dense(units=512,activation='relu'))
dmlp.add(Dense(units=512,activation='relu'))
dmlp.add(Dense(units=10,activation='softmax'))

dmlp.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.0001),metrics=['accuracy'])

hist=dmlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
```

```
Epoch 1/50
391/391 - 37s - loss: 1.8238 - accuracy: 0.3490 - val_loss: 1.6463 - val_accuracy: 0.4195 - 37s/epoch - 95ms/step
Epoch 2/50
391/391 - 33s - loss: 1.6242 - accuracy: 0.4223 - val_loss: 1.6044 - val_accuracy: 0.4213 - 33s/epoch - 83ms/step
Epoch 3/50
391/391 - 34s - loss: 1.5373 - accuracy: 0.4568 - val_loss: 1.5097 - val_accuracy: 0.4582 - 34s/epoch - 88ms/step
Epoch 4/50
391/391 - 33s - loss: 1.4788 - accuracy: 0.4754 - val_loss: 1.4783 - val_accuracy: 0.4701 - 33s/epoch - 84ms/step
Epoch 5/50
391/391 - 33s - loss: 1.4282 - accuracy: 0.4937 - val_loss: 1.4577 - val_accuracy: 0.4801 - 33s/epoch - 86ms/step
Epoch 6/50
391/391 - 33s - loss: 1.3820 - accuracy: 0.5101 - val_loss: 1.4157 - val_accuracy: 0.4980 - 33s/epoch - 84ms/step
Epoch 7/50
391/391 - 33s - loss: 1.3435 - accuracy: 0.5238 - val_loss: 1.4164 - val_accuracy: 0.4972 - 33s/epoch - 84ms/step
Epoch 8/50
391/391 - 33s - loss: 1.3136 - accuracy: 0.5374 - val_loss: 1.3759 - val_accuracy: 0.5068 - 33s/epoch - 84ms/step
Epoch 9/50
391/391 - 33s - loss: 1.2742 - accuracy: 0.5507 - val_loss: 1.3689 - val_accuracy: 0.5162 - 33s/epoch - 85ms/step
Epoch 10/50
391/391 - 32s - loss: 1.2455 - accuracy: 0.5618 - val_loss: 1.3624 - val_accuracy: 0.5151 - 32s/epoch - 82ms/step
Epoch 11/50
391/391 - 33s - loss: 1.2152 - accuracy: 0.5732 - val_loss: 1.3326 - val_accuracy: 0.5294 - 33s/epoch - 84ms/step
Epoch 12/50
391/391 - 33s - loss: 1.1821 - accuracy: 0.5864 - val_loss: 1.3144 - val_accuracy: 0.5329 - 33s/epoch - 84ms/step
Epoch 13/50
391/391 - 36s - loss: 1.1492 - accuracy: 0.5966 - val_loss: 1.3378 - val_accuracy: 0.5219 - 36s/epoch - 93ms/step
Epoch 14/50
391/391 - 33s - loss: 1.1220 - accuracy: 0.6063 - val_loss: 1.3013 - val_accuracy: 0.5418 - 33s/epoch - 84ms/step
Epoch 15/50
391/391 - 32s - loss: 1.0948 - accuracy: 0.6164 - val_loss: 1.3421 - val_accuracy: 0.5304 - 32s/epoch - 82ms/step
Epoch 16/50
391/391 - 33s - loss: 1.0733 - accuracy: 0.6247 - val_loss: 1.3514 - val_accuracy: 0.5363 - 33s/epoch - 84ms/step
Epoch 17/50
391/391 - 33s - loss: 1.0408 - accuracy: 0.6369 - val_loss: 1.3124 - val_accuracy: 0.5437 - 33s/epoch - 84ms/step
Epoch 18/50
391/391 - 33s - loss: 1.0156 - accuracy: 0.6440 - val_loss: 1.3350 - val_accuracy: 0.5310 - 33s/epoch - 84ms/step
Epoch 19/50
391/391 - 33s - loss: 0.9851 - accuracy: 0.6556 - val_loss: 1.3107 - val_accuracy: 0.5485 - 33s/epoch - 84ms/step
Epoch 20/50
391/391 - 34s - loss: 0.9663 - accuracy: 0.6620 - val_loss: 1.3367 - val_accuracy: 0.5421 - 34s/epoch - 86ms/step
Epoch 21/50
391/391 - 33s - loss: 0.9341 - accuracy: 0.6746 - val_loss: 1.3436 - val_accuracy: 0.5367 - 33s/epoch - 84ms/step
Epoch 22/50
391/391 - 33s - loss: 0.9060 - accuracy: 0.6855 - val_loss: 1.3161 - val_accuracy: 0.5494 - 33s/epoch - 85ms/step
Epoch 23/50
391/391 - 32s - loss: 0.8842 - accuracy: 0.6926 - val_loss: 1.3101 - val_accuracy: 0.5522 - 32s/epoch - 83ms/step
Epoch 24/50
```

```
391/391 - 33s - loss: 0.8570 - accuracy: 0.7037 - val_loss: 1.3286 - val_accuracy: 0.5497 - 33s/epoch - 85ms/step
Epoch 25/50
391/391 - 33s - loss: 0.8275 - accuracy: 0.7117 - val_loss: 1.3345 - val_accuracy: 0.5498 - 33s/epoch - 84ms/step
Epoch 26/50
391/391 - 33s - loss: 0.8099 - accuracy: 0.7184 - val_loss: 1.3433 - val_accuracy: 0.5525 - 33s/epoch - 83ms/step
Epoch 27/50
391/391 - 32s - loss: 0.7835 - accuracy: 0.7263 - val_loss: 1.3438 - val_accuracy: 0.5506 - 32s/epoch - 83ms/step
Epoch 28/50
391/391 - 34s - loss: 0.7528 - accuracy: 0.7388 - val_loss: 1.3693 - val_accuracy: 0.5528 - 34s/epoch - 88ms/step
Epoch 29/50
```

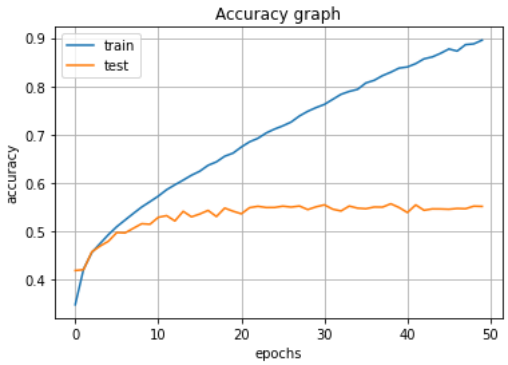
```
print('정확률=', dmlp.evaluate(x_test,y_test,verbose=0)[1]*100)
```

```
정확률= 55.19999861717224
```

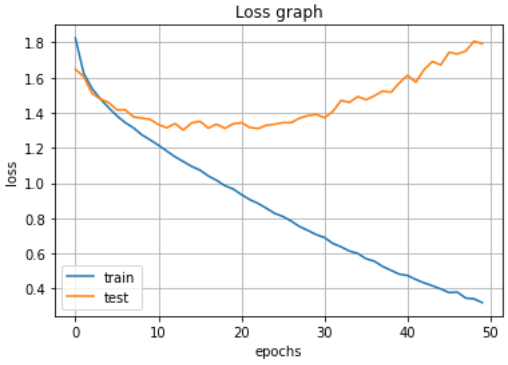
```
# dmlp.save('dmlp_trained.h5')
```

```
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy graph')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['train','test'])
plt.grid()
plt.show()
```




```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss graph')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['train','test'])
plt.grid()
plt.show()
```



✓ 0초 오전 7:54에 완료됨