모델의 성능 검증하기

1. 데이터의 확인과 예측 실행

```
import pandas as pd

# 광물 데이터를 불러옵니다.
df = pd.read_csv('./data/sonar3.csv', header=None)

# 첫 5줄을 봅니다.
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...

5 rows × 61 columns

```
# 일반 암석(0)과 광석(1)이 몇 개 있는지 확인합니다.
df[60].value_counts()
```

```
1    111
0     97
```

저장이 완료되었습니다.

```
# 음파 관련 속성을 X로, 광물의 종류를 y로 저장합니다.
X = df.iloc[:,0:60]
y = df.iloc[:,60]

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

# 모델을 설정합니다.
model = Sequential()
model.add(Dense(24,  input_dim=60, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델을 실행합니다.
history=model.fit(X, y, epochs=200, batch_size=10)
```

```
Epoch 177/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0304 - accuracy: 0.9952
Epoch 178/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0204 - accuracy: 1.0000
Epoch 179/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0190 - accuracy: 1.0000
Epoch 180/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0182 - accuracy: 1.0000
Epoch 181/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0194 - accuracy: 1.0000
Epoch 182/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0191 - accuracy: 1.0000
Epoch 183/200
21/21 [==============================] - 0s 3ms/step - loss: 0.0167 - accuracy: 1.0000
Epoch 184/200
21/21 [==============================] - 0s 3ms/step - loss: 0.0170 - accuracy: 1.0000
Epoch 185/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0149 - accuracy: 1.0000
Epoch 186/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0150 - accuracy: 1.0000
Epoch 187/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0156 - accuracy: 1.0000
Epoch 188/200
21/21 [==============================] - 0s 3ms/step - loss: 0.0141 - accuracy: 1.0000
Epoch 189/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0140 - accuracy: 1.0000
Epoch 190/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0140 - accuracy: 1.0000
Epoch 191/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0138 - accuracy: 1.0000
Epoch 192/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0161 - accuracy: 1.0000
Epoch 193/200
21/21 [==============================] - 0s 3ms/step - loss: 0.0152 - accuracy: 1.0000
Epoch 194/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0142 - accuracy: 1.0000
Epoch 195/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0133 - accuracy: 1.0000
Epoch 196/200
21/21 [==============================] - 0s 3ms/step - loss: 0.0163 - accuracy: 1.0000
Epoch 197/200
21/21 [==============================] - 0s 3ms/step - loss: 0.0128 - accuracy: 1.0000
Epoch 198/200
21/21 [==============================] - 0s 2ms/step - loss: 0.0144 - accuracy: 1.0000
Epoch 199/200
21/21 [==============================] - 0s 4ms/step - loss: 0.0151 - accuracy: 1.0000
```

저장이 완료되었습니다.

```
==============] - 0s 4ms/step - loss: 0.0137 - accuracy: 1.0000
```

3. 학습셋과 테스트셋

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

import pandas as pd

# 광물 데이터를 불러옵니다.
df = pd.read_csv('./data/sonar3.csv', header=None)

# 음파 관련 속성을 X로, 광물의 종류를 y로 저장합니다.
X = df.iloc[:,0:60]
y = df.iloc[:,60]

# 학습셋과 테스트셋을 구분합니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)

 # 모델을 설정합니다.
 model = Sequential()
 model.add(Dense(24,  input_dim=60, activation='relu'))
 model.add(Dense(10, activation='relu'))
 model.add(Dense(1, activation='sigmoid'))
```

```
# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델을 실행합니다.
history=model.fit(X_train, y_train, epochs=200, batch_size=10)
```

```
Epoch 172/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0286 - accuracy: 1.0000
Epoch 173/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0296 - accuracy: 1.0000
Epoch 174/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0285 - accuracy: 1.0000
Epoch 175/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0262 - accuracy: 1.0000
Epoch 176/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0307 - accuracy: 1.0000
Epoch 177/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0349 - accuracy: 1.0000
Epoch 178/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0260 - accuracy: 1.0000
Epoch 179/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0248 - accuracy: 1.0000
Epoch 180/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0243 - accuracy: 1.0000
Epoch 181/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0249 - accuracy: 1.0000
Epoch 182/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0265 - accuracy: 1.0000
Epoch 183/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0233 - accuracy: 1.0000
Epoch 184/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0223 - accuracy: 1.0000
Epoch 185/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0236 - accuracy: 1.0000
Epoch 186/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0222 - accuracy: 1.0000
Epoch 187/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0203 - accuracy: 1.0000
Epoch 188/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0201 - accuracy: 1.0000
Epoch 189/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0203 - accuracy: 1.0000
Epoch 190/200
                                        ] - 0s 3ms/step - loss: 0.0193 - accuracy: 1.0000
                                        ] - 0s 3ms/step - loss: 0.0195 - accuracy: 1.0000
Epoch 192/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0195 - accuracy: 1.0000
Epoch 193/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0188 - accuracy: 1.0000
Epoch 194/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0182 - accuracy: 1.0000
Epoch 195/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0175 - accuracy: 1.0000
Epoch 196/200
15/15 [==============================] - 0s 4ms/step - loss: 0.0175 - accuracy: 1.0000
Epoch 197/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0172 - accuracy: 1.0000
Epoch 198/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0167 - accuracy: 1.0000
Epoch 199/200
15/15 [==============================] - 0s 3ms/step - loss: 0.0167 - accuracy: 1.0000
Epoch 200/200
15/15 [==============================] - 0s 2ms/step - loss: 0.0169 - accuracy: 1.0000
```

저장이 완료되었습니다.

```
# 모델을 테스트셋에 적용해 정확도를 구합니다.
score=model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

```
2/2 [==============================] - 0s 9ms/step - loss: 0.5739 - accuracy: 0.8254
Test accuracy: 0.8253968358039856
```

4. 모델 저장과 재사용

```
# 모델 이름과 저장할 위치를 함께 지정합니다.
model.save('./data/model/my_model.hdf5')
```

```
from tensorflow.keras.models import Sequential, load_model

# 테스트를 위해 조금 전 사용한 모델을 메모리에서 삭제합니다.
del model

# 모델을 새로 불러옵니다.
model = load_model('./data/model/my_model.hdf5')

# 불러온 모델을 테스트셋에 적용해 정확도를 구합니다.
score=model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.5739 - accuracy: 0.8254
Test accuracy: 0.8253968358039856
```

5. k겹 교차 검증

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

import pandas as pd

# 광물 데이터를 불러옵니다.
df = pd.read_csv('./data/sonar3.csv', header=None)

# 음파 관련 속성을 X로, 광물의 종류를 y로 저장합니다.
X = df.iloc[:,0:60]
y = df.iloc[:,60]
```

저장이 완료되었습니다.

정합니다.

```
k=5

# KFold 함수를 불러옵니다. 분할하기 전에 샘플이 치우치지 않도록 섞어 줍니다.
kfold = KFold(n_splits=k, shuffle=True)

# 정확도가 채워질 빈 리스트를 준비합니다.
acc_score = []

def model_fn():
    model = Sequential() # 딥러닝 모델의 구조를 시작합니다.
    model.add(Dense(24, input_dim=60, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    return model

# K겹 교차 검증을 이용해 k번의 학습을 실행합니다.
for train_index , test_index in kfold.split(X):  # for 문에 의해서 k번 반복합니다. spilt()에 의해 k
    X_train , X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train , y_test = y.iloc[train_index], y.iloc[test_index]

    model = model_fn()
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    history=model.fit(X_train, y_train, epochs=200, batch_size=10, verbose=0)

    accuracy = model.evaluate(X_test, y_test)[1]  # 정확도를 구합니다.
    acc_score.append(accuracy)  # 정확도 리스트에 저장합니다.
```

```
# k번 실시된 정확도의 평균을 구합니다.
avg_acc_score = sum(acc_score)/k

# 결과를 출력합니다.
print('정확도:', acc_score)
print('정확도 평균:', avg_acc_score)
```

```
2/2 [==============================] - 0s 11ms/step - loss: 0.5545 - accuracy: 0.7381
2/2 [==============================] - 0s 7ms/step - loss: 0.4469 - accuracy: 0.8810
WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_test_function.<locals>.test_function at 0x7f75feb54ca0> triggered tf
2/2 [==============================] - 0s 12ms/step - loss: 1.0392 - accuracy: 0.6905
WARNING:tensorflow:6 out of the last 11 calls to <function Model.make_test_function.<locals>.test_function at 0x7f75fea20ee0> triggered t
2/2 [==============================] - 0s 11ms/step - loss: 0.6882 - accuracy: 0.7805
2/2 [==============================] - 0s 10ms/step - loss: 0.5391 - accuracy: 0.8293
정확도: [0.738095223903656, 0.8809523582458496, 0.6904761791229248, 0.7804877758026123, 0.8292682766914368]
정확도 평균: 0.7838559627532959
```