Keras iris Modeling

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

- 실습용 데이터 설정
  - iris.csv

```
import seaborn as sns

iris = sns.load_dataset('iris')
```

- pandas DataFrame

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

I. Data Preprocessing

1) iris.Species 빈도분석

- Species : setosa, virginica, versicolor

```
iris.species.value_counts()
```

```
virginica     50
versicolor    50
setosa        50
Name: species, dtype: int64
```

2) DataFrame to Array & Casting

```
iris_AR = iris.values

iris_AR
```

```
array([[5.1, 3.5, 1.4, 0.2, 'setosa'],
       [4.9, 3.0, 1.4, 0.2, 'setosa'],
```

```
       [4.7, 3.2, 1.3, 0.2, 'setosa'],
       [4.6, 3.1, 1.5, 0.2, 'setosa'],
       [5.0, 3.6, 1.4, 0.2, 'setosa'],
       [5.4, 3.9, 1.7, 0.4, 'setosa'],
       [4.6, 3.4, 1.4, 0.3, 'setosa'],
       [5.0, 3.4, 1.5, 0.2, 'setosa'],
       [4.4, 2.9, 1.4, 0.2, 'setosa'],
       [4.9, 3.1, 1.5, 0.1, 'setosa'],
       [5.4, 3.7, 1.5, 0.2, 'setosa'],
       [4.8, 3.4, 1.6, 0.2, 'setosa'],
       [4.8, 3.0, 1.4, 0.1, 'setosa'],
       [4.3, 3.0, 1.1, 0.1, 'setosa'],
       [5.8, 4.0, 1.2, 0.2, 'setosa'],
       [5.7, 4.4, 1.5, 0.4, 'setosa'],
       [5.4, 3.9, 1.3, 0.4, 'setosa'],
       [5.1, 3.5, 1.4, 0.3, 'setosa'],
       [5.7, 3.8, 1.7, 0.3, 'setosa'],
       [5.1, 3.8, 1.5, 0.3, 'setosa'],
       [5.4, 3.4, 1.7, 0.2, 'setosa'],
       [5.1, 3.7, 1.5, 0.4, 'setosa'],
       [4.6, 3.6, 1.0, 0.2, 'setosa'],
       [5.1, 3.3, 1.7, 0.5, 'setosa'],
       [4.8, 3.4, 1.9, 0.2, 'setosa'],
       [5.0, 3.0, 1.6, 0.2, 'setosa'],
       [5.0, 3.4, 1.6, 0.4, 'setosa'],
       [5.2, 3.5, 1.5, 0.2, 'setosa'],
       [5.2, 3.4, 1.4, 0.2, 'setosa'],
       [4.7, 3.2, 1.6, 0.2, 'setosa'],
       [4.8, 3.1, 1.6, 0.2, 'setosa'],
       [5.4, 3.4, 1.5, 0.4, 'setosa'],
       [5.2, 4.1, 1.5, 0.1, 'setosa'],
       [5.5, 4.2, 1.4, 0.2, 'setosa'],
       [4.9, 3.1, 1.5, 0.2, 'setosa'],
       [5.0, 3.2, 1.2, 0.2, 'setosa'],
       [5.5, 3.5, 1.3, 0.2, 'setosa'],
       [4.9, 3.6, 1.4, 0.1, 'setosa'],
       [4.4, 3.0, 1.3, 0.2, 'setosa'],
       [5.1, 3.4, 1.5, 0.2, 'setosa'],
       [5.0, 3.5, 1.3, 0.3, 'setosa'],
       [4.5, 2.3, 1.3, 0.3, 'setosa'],
       [4.4, 3.2, 1.3, 0.2, 'setosa'],
       [5.0, 3.5, 1.6, 0.6, 'setosa'],
       [5.1, 3.8, 1.9, 0.4, 'setosa'],
       [4.8, 3.0, 1.4, 0.3, 'setosa'],
       [5.1, 3.8, 1.6, 0.2, 'setosa'],
       [4.6, 3.2, 1.4, 0.2, 'setosa'],
       [5.3, 3.7, 1.5, 0.2, 'setosa'],
       [5.0, 3.3, 1.4, 0.2, 'setosa'],
       [7.0, 3.2, 4.7, 1.4, 'versicolor'],
       [6.4, 3.2, 4.5, 1.5, 'versicolor'],
       [6.9, 3.1, 4.9, 1.5, 'versicolor'],
       [5.5, 2.3, 4.0, 1.3, 'versicolor'],
       [6.5, 2.8, 4.6, 1.5, 'versicolor'],
       [5.7, 2.8, 4.5, 1.3, 'versicolor'],
       [6.3, 3.3, 4.7, 1.6, 'versicolor'],
       [4.9, 2.4, 3.3, 1.0, 'versicolor'],
       [6.6, 2.9, 4.6, 1.3, 'versicolor'],
       [5.2, 2.7, 3.9, 1.4, 'versicolor'],
```

- object to float

```
AR_X = iris_AR[:, 0:4].astype(float)
AR_y = iris_AR[:, 4]

AR_X.shape, AR_y.shape
```

```
((150, 4), (150,))
```

3) One Hot Encoding with sklearn & Keras

- LabelEncoder( )
  - ['setosa', 'virginica', 'virsicolor'] to [0, 1, 2]

```
from sklearn.preprocessing import LabelEncoder

encoder =  LabelEncoder()
AR_yLBE = encoder.fit_transform(AR_y)
```

```
AR_yLBE
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

- One-Hot Encoding - to_categorical( )

```
from tensorflow.keras.utils import to_categorical

AR_yOHE = to_categorical(AR_yLBE)

AR_yOHE
```

```
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]], dtype=float32)
```

4) Version check

- tensorFlow Version

```
import tensorflow

tensorflow.__version__
```

```
'2.6.0'
```

- Keras Version

```
import keras

keras.__version__
```

```
'2.6.0'
```

5) train_test_split( )

- 7 : 3

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(AR_X, AR_yOHE,
                                                    test_size = 0.3,
                                                    random_state = 2045)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((105, 4), (45, 4), (105, 3), (45, 3))
```

II. Keras Modeling

1) Keras models & layers Import

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

2) Model Define

- 모델 신경망 구조 정의

```
Model_iris = models.Sequential()

Model_iris.add(layers.Dense(16, activation = 'relu', input_shape = (4,)))
Model_iris.add(layers.Dense(8, activation = 'relu'))
Model_iris.add(layers.Dense(3, activation = 'softmax'))
```

- 모델 구조 확인
  - Layers & Parameters

```
Model_iris.summary()
```

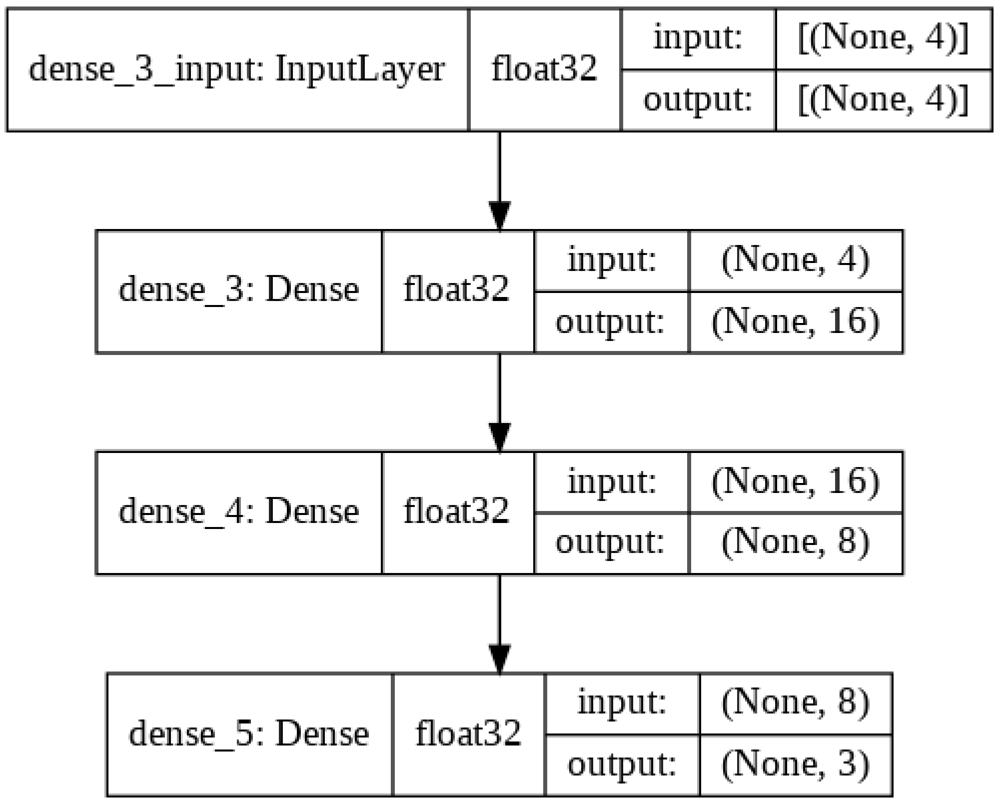
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 16)                80
_________________________________________________________________
dense_19 (Dense)             (None, 8)                 136
```

```
dense_20 (Dense)             (None, 3)                 27
=================================================================
Total params: 243
Trainable params: 243
Non-trainable params: 0
_________________________________________________________________
```

- 모델 레이어 시각화

```
from tensorflow.keras import utils

utils.plot_model(Model_iris,
                 show_shapes = True,
                 show_dtype = True)
```



3) Model Compile

- 모델 학습방법 설정

```
Model_iris.compile(loss = 'categorical_crossentropy',
                   optimizer = 'adam',
                   metrics = ['accuracy'])
```

4) Model Fit

- 모델 학습 수행

```
History_iris = Model_iris.fit(X_train, y_train,
                              epochs = 500,
                              batch_size = 7,
                              validation_data = (X_test, y_test))
```

```
15/15 [==============================] - 0s 3ms/step - loss: 0.0604 - accuracy: 0.9810 - val_loss: 0.0337 - val_accuracy: 0.9778
Epoch 429/500
15/15 [==============================] - 0s 5ms/step - loss: 0.0666 - accuracy: 0.9810 - val_loss: 0.0336 - val_accuracy: 0.9778
Epoch 430/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0728 - accuracy: 0.9714 - val_loss: 0.0240 - val_accuracy: 0.9778
Epoch 431/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0568 - accuracy: 0.9810 - val_loss: 0.0384 - val_accuracy: 0.9778
Epoch 432/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0635 - accuracy: 0.9810 - val_loss: 0.0344 - val_accuracy: 0.9778
Epoch 433/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0613 - accuracy: 0.9810 - val_loss: 0.0317 - val_accuracy: 0.9778
Epoch 434/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0649 - accuracy: 0.9810 - val_loss: 0.0277 - val_accuracy: 0.9778
Epoch 435/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0615 - accuracy: 0.9905 - val_loss: 0.0347 - val_accuracy: 0.9778
Epoch 436/500
```

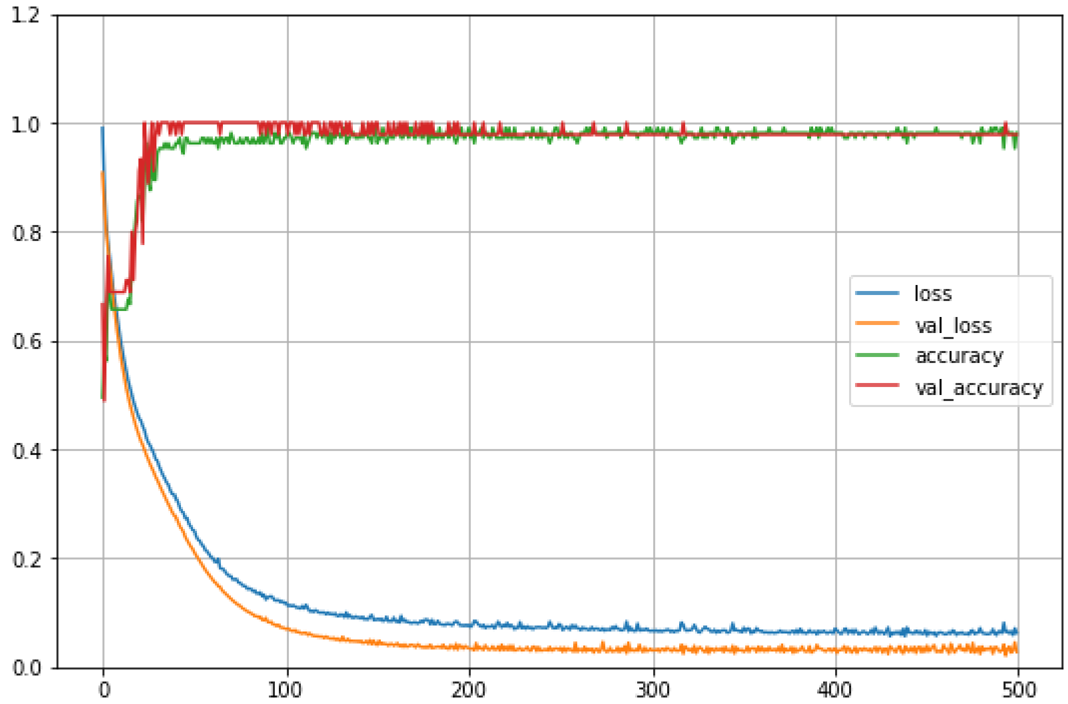
```
15/15 [==============================] - 0s 3ms/step - loss: 0.0655 - accuracy: 0.9810 - val_loss: 0.0385 - val_accuracy: 0.9778
Epoch 437/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0590 - accuracy: 0.9810 - val_loss: 0.0284 - val_accuracy: 0.9778
Epoch 438/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0703 - accuracy: 0.9714 - val_loss: 0.0253 - val_accuracy: 0.9778
Epoch 439/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0568 - accuracy: 0.9905 - val_loss: 0.0384 - val_accuracy: 0.9778
Epoch 440/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0727 - accuracy: 0.9619 - val_loss: 0.0320 - val_accuracy: 0.9778
Epoch 441/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0638 - accuracy: 0.9810 - val_loss: 0.0285 - val_accuracy: 0.9778
Epoch 442/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0642 - accuracy: 0.9810 - val_loss: 0.0359 - val_accuracy: 0.9778
Epoch 443/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0621 - accuracy: 0.9810 - val_loss: 0.0293 - val_accuracy: 0.9778
Epoch 444/500
15/15 [==============================] - 0s 5ms/step - loss: 0.0625 - accuracy: 0.9714 - val_loss: 0.0371 - val_accuracy: 0.9778
Epoch 445/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0669 - accuracy: 0.9810 - val_loss: 0.0260 - val_accuracy: 0.9778
Epoch 446/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0622 - accuracy: 0.9810 - val_loss: 0.0358 - val_accuracy: 0.9778
Epoch 447/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0657 - accuracy: 0.9810 - val_loss: 0.0374 - val_accuracy: 0.9778

Epoch 448/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0614 - accuracy: 0.9810 - val_loss: 0.0270 - val_accuracy: 0.9778
Epoch 449/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0614 - accuracy: 0.9810 - val_loss: 0.0359 - val_accuracy: 0.9778
Epoch 450/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0680 - accuracy: 0.9810 - val_loss: 0.0283 - val_accuracy: 0.9778
Epoch 451/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0582 - accuracy: 0.9810 - val_loss: 0.0395 - val_accuracy: 0.9778
Epoch 452/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0650 - accuracy: 0.9810 - val_loss: 0.0293 - val_accuracy: 0.9778
Epoch 453/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0626 - accuracy: 0.9810 - val_loss: 0.0317 - val_accuracy: 0.9778
Epoch 454/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0618 - accuracy: 0.9810 - val_loss: 0.0381 - val_accuracy: 0.9778
Epoch 455/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0637 - accuracy: 0.9810 - val_loss: 0.0401 - val_accuracy: 0.9778
Epoch 456/500
15/15 [==============================] - 0s 4ms/step - loss: 0.0575 - accuracy: 0.9905 - val_loss: 0.0282 - val_accuracy: 0.9778
Epoch 457/500
15/15 [==============================] - 0s 3ms/step - loss: 0.0628 - accuracy: 0.9810 - val_loss: 0.0287 - val_accuracy: 0.9778
```

5) 학습 결과 시각화

```
import matplotlib.pyplot as plt

plt.figure(figsize = (9, 6))
plt.ylim(0, 1.2)
plt.plot(History_iris.history['loss'])
plt.plot(History_iris.history['val_loss'])
plt.plot(History_iris.history['accuracy'])
plt.plot(History_iris.history['val_accuracy'])
plt.legend(['loss', 'val_loss', 'accuracy', 'val_accuracy'])
plt.grid()
plt.show()
```



6) Model Evaluate

- Loss & Accuracy

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

print('Loss = {:.2f}'.format(loss))
print('Accuracy = {:.2f}'.format(accuracy))
```

```
2/2 [==============================] - 0s 6ms/step - loss: 0.0281 - accuracy: 0.9778
Loss = 0.03
Accuracy = 0.98
```

7) Model Predict

- Probability

```
import numpy as np
np.set_printoptions(suppress = True, precision = 5)

Model_iris.predict(X_test)
```

```
array([[0.99992, 0.00008, 0.     ],
       [0.99914, 0.00086, 0.     ],
       [0.00133, 0.99866, 0.00001],
       [0.     , 0.00009, 0.99991],
       [0.99999, 0.00001, 0.     ],
       [0.00015, 0.99984, 0.00001],
       [0.     , 0.00301, 0.99699],
       [1.     , 0.     , 0.     ],
       [0.     , 0.00036, 0.99964],
       [0.99992, 0.00008, 0.     ],
       [0.00004, 0.88859, 0.11137],
       [0.     , 0.00135, 0.99865],
       [0.00003, 0.99808, 0.00189],
       [1.     , 0.     , 0.     ],
       [0.99999, 0.00001, 0.     ],
       [0.00006, 0.99736, 0.00258],
       [0.00002, 0.98803, 0.01194],
       [0.99996, 0.00004, 0.     ],
       [0.00002, 0.9998 , 0.00018],
       [1.     , 0.     , 0.     ],
       [0.99994, 0.00006, 0.     ],
       [0.99994, 0.00006, 0.     ],
       [0.     , 0.00045, 0.99955],
       [0.99998, 0.00002, 0.     ],
       [0.     , 0.03228, 0.96772],
       [0.     , 0.00468, 0.99532],
       [0.99999, 0.00001, 0.     ],
       [0.00001, 0.99842, 0.00157],
       [0.00001, 0.99814, 0.00185],
       [0.     , 0.02365, 0.97635],
       [1.     , 0.     , 0.     ],
       [0.00001, 0.99911, 0.00088],
       [0.99999, 0.00001, 0.     ],
       [0.00001, 0.64521, 0.35478],
       [0.00003, 0.99667, 0.0033 ],
       [0.00008, 0.9999 , 0.00001],
       [0.     , 0.00009, 0.99991],
       [0.0001 , 0.99982, 0.00008],
       [0.99995, 0.00005, 0.     ],
       [0.00005, 0.99855, 0.0014 ],
       [0.     , 0.00014, 0.99986],
       [0.99977, 0.00023, 0.     ],
       [0.     , 0.00112, 0.99888],
       [0.     , 0.00054, 0.99946],
       [0.     , 0.01126, 0.98874]], dtype=float32)
```

- Probability to Class

```
y_hat = np.argmax(Model_iris.predict(X_test), axis = 1)

y_hat
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

- One-Hot Encoding to Array
    - np.argmax( ) : 다차원 배열의 차원에 따라 가장 큰 값의 인덱스를 반환
    - axis = 1 : 열기준

```
y = np.argmax(y_test, axis = 1)

y
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 2, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

- Confusion Matrix & Claasification Report

```
from sklearn.metrics import confusion_matrix, classification_report

confusion_matrix(y, y_hat)
```

```
array([[17,  0,  0],
       [ 0, 14,  0],
       [ 0,  1, 13]])
```

```
print(classification_report(y, y_hat,
                            target_names = ['setosa',
                                            'virginica',
                                            'versicolor']))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        17
   virginica       0.93      1.00      0.97        14
  versicolor       1.00      0.93      0.96        14

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

III. Model Save & Load

1) File System

- Save to Colab File System

```
!ls -l
```

```
total 36
-rw-r--r-- 1 root root 30037 Sep 28 06:49 model.png
drwxr-xr-x 1 root root  4096 Sep 16 13:40 sample_data
```

```
Model_iris.save('Model_iris.h5')

!ls -l
```

```
total 72
-rw-r--r-- 1 root root 34600 Sep 28 07:33 Model_iris.h5
-rw-r--r-- 1 root root 30037 Sep 28 06:49 model.png
drwxr-xr-x 1 root root  4096 Sep 16 13:40 sample_data
```

- Download Colab File System to Local File System

```
from google.colab import files

files.download('Model_iris.h5')
```

- Load from Colab File System

```
from tensorflow.keras.models import load_model

Model_local = load_model('Model_iris.h5')
```

```
np.argmax(Model_local.predict(X_test), axis = 1)
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

2) Google Drive

- Mount Google Drive

```
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

- Check Mounted_Drive

```
!ls -l '/content/drive/My Drive/Colab Notebooks/models'
```

```
total 0
```

- Save to Mounted Google Drive Directory

```
Model_iris.save('/content/drive/My Drive/Colab Notebooks/models/001_Model_iris.h5')
```

```
!ls -l '/content/drive/My Drive/Colab Notebooks/models'
```

```
total 34
-rw------- 1 root root 34600 Sep 28 07:37 001_Model_iris.h5
```

- Load from Mounted Google Drive Directory

```
from tensorflow.keras.models import load_model

Model_google = load_model('/content/drive/My Drive/Colab Notebooks/models/001_Model_iris.h5')
```

```
np.argmax(Model_google.predict(X_test), axis = 1)
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

#

#

#

The End

#

#

#

0초 오후 4:38에 완료됨