CIFAR 10 - Functional API Modeling

- Categorical Classification

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

Import Tensorflow & Keras

- import Keras

```
import keras

keras.__version__
```

'2.4.3'

+ 코드 + 텍스트

I. CIFAR 10 Data_Set Load & Review

1) Load CIFAR 10 Data_Set

```
from keras.datasets import cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 12s 0us/step
```

- Train_Data Information

```
print(len(X_train))
print(X_train.shape)

print(len(y_train))
print(y_train[0:5])
```

```
50000
(50000, 32, 32, 3)
50000
[[6]
 [9]
```

```
 [9]
 [4]
 [1]]
```

- Test_Data Information

```
print(len(X_test))
print(X_test.shape)

print(len(y_test))
print(y_test[0:5])
```

```
10000
(10000, 32, 32, 3)
10000
[[3]
 [8]
 [8]
 [0]
 [6]]
```

II. Data Preprocessing

1) Normalization

```
X_train = X_train.astype(float) / 255
X_test = X_test.astype(float) / 255
```

```
print(X_train[0])
```

```
[[[0.23137255 0.24313725 0.24705882]
  [0.16862745 0.18039216 0.17647059]
  [0.19607843 0.18823529 0.16862745]
  ...
  [0.61960784 0.51764706 0.42352941]
  [0.59607843 0.49019608 0.4       ]
  [0.58039216 0.48627451 0.40392157]]

 [[0.0627451  0.07843137 0.07843137]
  [0.         0.         0.        ]
  [0.07058824 0.03137255 0.        ]
  ...
  [0.48235294 0.34509804 0.21568627]
  [0.46666667 0.3254902  0.19607843]
  [0.47843137 0.34117647 0.22352941]]

 [[0.09803922 0.09411765 0.08235294]
  [0.0627451  0.02745098 0.        ]
  [0.19215686 0.10588235 0.03137255]
  ...
  [0.4627451  0.32941176 0.19607843]
```

```
 [0.47058824 0.32941176 0.19607843]
 [0.42745098 0.28627451 0.16470588]]

...

[[0.81568627 0.66666667 0.37647059]
 [0.78823529 0.6        0.13333333]
 [0.77647059 0.63137255 0.10196078]
 ...
 [0.62745098 0.52156863 0.2745098 ]
 [0.21960784 0.12156863 0.02745098]
 [0.20784314 0.13333333 0.07843137]]

[[0.70588235 0.54509804 0.37647059]
 [0.67843137 0.48235294 0.16470588]
 [0.72941176 0.56470588 0.11764706]
 ...
 [0.72156863 0.58039216 0.36862745]
 [0.38039216 0.24313725 0.13333333]
 [0.3254902  0.20784314 0.13333333]]

[[0.69411765 0.56470588 0.45490196]
 [0.65882353 0.50588235 0.36862745]
 [0.70196078 0.55686275 0.34117647]
 ...
 [0.84705882 0.72156863 0.54901961]
 [0.59215686 0.4627451  0.32941176]
 [0.48235294 0.36078431 0.28235294]]]
```

2) One Hot Encoding

```
from keras.utils import to_categorical

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
print(y_train[:5])
```

```
[[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. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

III. Keras Functional API Modeling

1) Model Define

- 모델 신경망 구조 정의

```
%%time

from keras import models
from keras import layers

input_img = layers.Input((32,32,3))

x = layers.Conv2D(filters = 32, kernel_size = (3, 3), strides = (1, 1), padding = 'same')(input_img
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)


x = layers.Conv2D(filters = 32, kernel_size = (3, 3), strides = (2, 2), padding = 'same')(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)


x = layers.Conv2D(filters = 64, kernel_size = (3, 3), strides = (1, 1), padding = 'same')(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)


x = layers.Conv2D(filters = 64, kernel_size = (3, 3), strides = (2, 2), padding = 'same')(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)


x = layers.Flatten()(x)

x = layers.Dense(128)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Dropout(0.5)(x)

x = layers.Dense(10)(x)
output_y_hat = layers.Activation('softmax')(x)

CIFAR = models.Model(input_img, output_y_hat)
```

```
CPU times: user 484 ms, sys: 216 ms, total: 701 ms
Wall time: 5.28 s
```

- 모델 구조 확인

```
CIFAR.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 32, 32, 32)        896
```

```
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 128)               524416
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
_________________________________________________________________
activation (Activation)      (None, 10)                0
=================================================================
Total params: 592,554
Trainable params: 591,914
Non-trainable params: 640
_________________________________________________________________
```

2) Model Compile

- 모델 학습방법 설정

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

3) Model Fit

- 약 6분

```
%%time

Hist_CIFAR = CIFAR.fit(X_train, y_train,
                       epochs = 100,
                       batch_size = 128,
                       validation_split = 0.2)
```

```
Epoch 1/100
313/313 [==============================] - 38s 26ms/step - loss: 1.8681 - accuracy: 0.3769 -
Epoch 2/100
313/313 [==============================] - 7s 24ms/step - loss: 1.2065 - accuracy: 0.5705 - v
Epoch 3/100
313/313 [==============================] - 7s 23ms/step - loss: 1.0075 - accuracy: 0.6465 - v
Epoch 4/100
313/313 [==============================] - 8s 24ms/step - loss: 0.9084 - accuracy: 0.6766 - v
Epoch 5/100
313/313 [==============================] - 8s 24ms/step - loss: 0.8243 - accuracy: 0.7091 - v
Epoch 6/100
313/313 [==============================] - 7s 23ms/step - loss: 0.7723 - accuracy: 0.7256 - v
Epoch 7/100
313/313 [==============================] - 7s 24ms/step - loss: 0.7374 - accuracy: 0.7409 - v
Epoch 8/100
313/313 [==============================] - 7s 24ms/step - loss: 0.6857 - accuracy: 0.7606 - v
Epoch 9/100
313/313 [==============================] - 7s 23ms/step - loss: 0.6459 - accuracy: 0.7735 - v
Epoch 10/100
313/313 [==============================] - 7s 24ms/step - loss: 0.5994 - accuracy: 0.7899 - v
Epoch 11/100
313/313 [==============================] - 7s 24ms/step - loss: 0.5638 - accuracy: 0.7995 - v
Epoch 12/100
313/313 [==============================] - 7s 24ms/step - loss: 0.5352 - accuracy: 0.8118 - v
Epoch 13/100
313/313 [==============================] - 7s 24ms/step - loss: 0.4908 - accuracy: 0.8269 - v
Epoch 14/100
313/313 [==============================] - 7s 23ms/step - loss: 0.4808 - accuracy: 0.8317 - v
Epoch 15/100
313/313 [==============================] - 8s 24ms/step - loss: 0.4448 - accuracy: 0.8455 - v
Epoch 16/100
313/313 [==============================] - 8s 25ms/step - loss: 0.4277 - accuracy: 0.8472 - v
Epoch 17/100
313/313 [==============================] - 8s 24ms/step - loss: 0.3881 - accuracy: 0.8616 - v
Epoch 18/100
313/313 [==============================] - 7s 24ms/step - loss: 0.3809 - accuracy: 0.8626 - v
Epoch 19/100
313/313 [==============================] - 7s 24ms/step - loss: 0.3580 - accuracy: 0.8718 - v
Epoch 20/100
313/313 [==============================] - 7s 24ms/step - loss: 0.3462 - accuracy: 0.8768 - v
Epoch 21/100
 25/313 [=>............................] - ETA: 6s - loss: 0.3197 - accuracy: 0.8948
```

4) 학습 결과 시각화

- Loss Visualization

```
import matplotlib.pyplot as plt

epochs = range(1, len(Hist_CIFAR.history['loss']) + 1)

plt.figure(figsize = (9, 6))
plt.plot(epochs, Hist_CIFAR.history['loss'])
plt.plot(epochs, Hist_CIFAR.history['val_loss'])
# plt.ylim(0, 0.25)
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 = CIFAR.evaluate(X_test, y_test)

print('Loss = {:.5f}'.format(loss))
print('Accuracy = {:.5f}'.format(accuracy))
```

IV. 성능평가

1) Label Name 지정

```
import numpy as np

CLASSES = np.array(['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
                    'Dog', 'Frog', 'Horse', 'Ship', 'Truck'])

preds = CIFAR.predict(X_test)
preds_single = CLASSES[np.argmax(preds, axis = -1)]
actual_single = CLASSES[np.argmax(y_test, axis = -1)]
```

2) 비교 시각화

```
import matplotlib.pyplot as plt

n_to_show = 10
```

```
n_to_show = 10
indices = np.random.choice(range(len(X_test)), n_to_show)

fig = plt.figure(figsize = (15, 3))
fig.subplots_adjust(hspace = 0.4, wspace = 0.4)

for i, idx in enumerate(indices):
    img = X_test[idx]
    ax = fig.add_subplot(1, n_to_show, i + 1)
    ax.axis('off')
    ax.text(0.5, -0.35,
            'Pred = ' + str(preds_single[idx]),
            fontsize = 10,
            ha = 'center',
            transform = ax.transAxes)
    ax.text(0.5, -0.7,
            'Act = ' + str(actual_single[idx]),
            fontsize = 10,
            ha = 'center',
            transform = ax.transAxes)
    ax.imshow(img)
```

#

#

#

The End

#

#

#