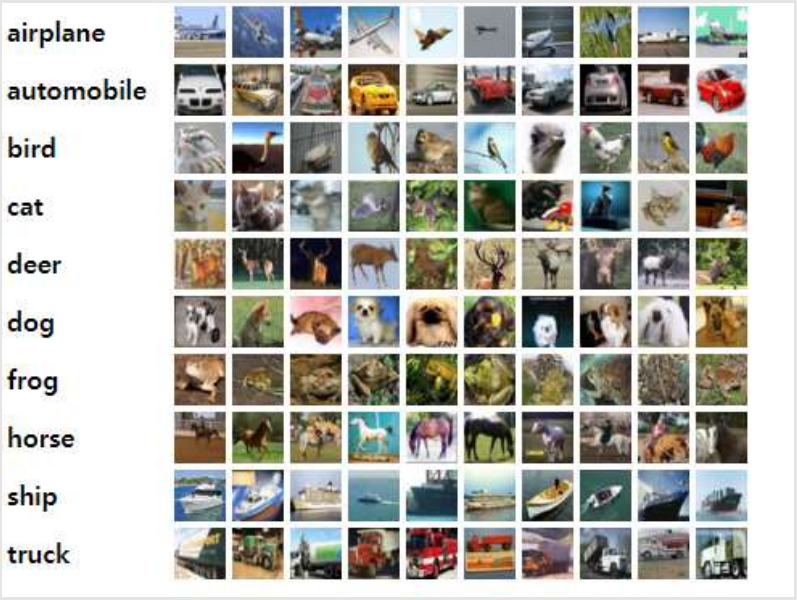
13주. Keras CNN			
학번	32183164	이름	이석현

Q1 (10점) CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class



- CIFAR-10 dataset 에 대해 CNN structure를 설계하고 모델을 개발한 후 테스트 결과를 제시하시오
  (train accuracy 와 test accuracy를 제시)
  * hidden layer 의 수는 3개 이상, layer별 노드수 및 기타 매개변수는 각자 정한다.
  * CIFAR-10 데이터셋을 읽어서 train, test set 을 준비하는 코드

```
from keras.datasets import cifar10

# load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
y_train = np_utils.to_categorical(y_train, nb_classes)
y_test = np_utils.to_categorical(y_test, nb_classes)
```

Source code :

```
// source code 의 폰트는 Courier10 BT Bold으로 하시오
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator

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

X_train = X_train / 255
X_test = X_test / 255

y_cat_train = to_categorical(y_train, 10)
y_cat_test = to_categorical(y_test, 10)

#Create Model
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
```

```
// source code 의 폰트는 Courier10 BT Bold으로 하시오
model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))

METRICS = [
    'accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=METRICS)

#Augment Data
batch_size = 32
data_generator = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
train_generator = data_generator.flow(X_train, y_cat_train, batch_size)
steps_per_epoch = X_train.shape[0] // batch_size

r = model.fit(train_generator,
              epochs=50,
              steps_per_epoch=steps_per_epoch,
              validation_data=(X_test, y_cat_test)
             )

evaluation = model.evaluate(X_test, y_cat_test)
print(f'Test Accuracy : {evaluation[1] * 100:.2f}%')
```

실행화면 캡쳐:

```
10000/10000 [==============================] - 18s 2ms/sample - loss: 0.4804 - acc: 0.8391 - precision: 0.8866 - recall: 0.7975
Test Accuracy : 83.91%
```

```
Epoch 10/50
10000/10000 [==============================] - 18s 2ms/sample - loss: 0.6057 - acc: 0.7962 - precision: 0.8555 - recall: 0.7459
1563/1563 [==============================] - 698s 447ms/step - loss: 0.6476 - acc: 0.7826 - precision: 0.8540 - recall: 0.7162 - val_loss:
0.6060 - val_acc: 0.7962 - val_precision: 0.8555 - val_recall: 0.7459
Epoch 11/50
10000/10000 [==============================] - 19s 2ms/sample - loss: 0.5966 - acc: 0.8007 - precision: 0.8607 - recall: 0.7502
1563/1563 [==============================] - 679s 434ms/step - loss: 0.6259 - acc: 0.7863 - precision: 0.8566 - recall: 0.7236 - val_loss:
0.5964 - val_acc: 0.8007 - val_precision: 0.8607 - val_recall: 0.7502
Epoch 12/50
10000/10000 [==============================] - 19s 2ms/sample - loss: 0.5359 - acc: 0.8217 - precision: 0.8778 - recall: 0.7697
1563/1563 [==============================] - 670s 429ms/step - loss: 0.6078 - acc: 0.7944 - precision: 0.8615 - recall: 0.7333 - val_loss:
0.5359 - val_acc: 0.8217 - val_precision: 0.8778 - val_recall: 0.7697
Epoch 13/50
10000/10000 [==============================] - 19s 2ms/sample - loss: 0.5914 - acc: 0.8046 - precision: 0.8548 - recall: 0.7589
1563/1563 [==============================] - 664s 425ms/step - loss: 0.5916 - acc: 0.7997 - precision: 0.8654 - recall: 0.7415 - val_loss:
0.5910 - val_acc: 0.8046 - val_precision: 0.8548 - val_recall: 0.7589
Epoch 14/50
10000/10000 [==============================] - 19s 2ms/sample - loss: 0.5874 - acc: 0.8071 - precision: 0.8565 - recall: 0.7637
1563/1563 [==============================] - 664s 425ms/step - loss: 0.5753 - acc: 0.8049 - precision: 0.8669 - recall: 0.7475 - val_loss:
0.5874 - val_acc: 0.8071 - val_precision: 0.8565 - val_recall: 0.7637
Epoch 15/50
10000/10000 [==============================] - 19s 2ms/sample - loss: 0.5270 - acc: 0.8289 - precision: 0.8740 - recall: 0.7878s - loss:
0.5252 - acc: 0.8289 - precisio
1563/1563 [==============================] - 663s 424ms/step - loss: 0.5605 - acc: 0.8104 - precision: 0.8686 - recall: 0.7562 - val_loss:
0.5267 - val_acc: 0.8289 - val_precision: 0.8740 - val_recall: 0.7878
Epoch 16/50
10000/10000 [==============================] - 19s 2ms/sample - loss: 0.4854 - acc: 0.8407 - precision: 0.8804 - recall: 0.8059
1563/1563 [==============================] - 671s 429ms/step - loss: 0.5422 - acc: 0.8162 - precision: 0.8756 - recall: 0.7648 - val_loss:
0.4853 - val_acc: 0.8407 - val_precision: 0.8804 - val_recall: 0.8059
Epoch 17/50
10000/10000 [==============================] - 18s 2ms/sample - loss: 0.5302 - acc: 0.8251 - precision: 0.8746 - recall: 0.7803
1563/1563 [==============================] - 666s 426ms/step - loss: 0.5402 - acc: 0.8179 - precision: 0.8754 - recall: 0.7651 - val_loss:
0.5304 - val_acc: 0.8251 - val_precision: 0.8746 - val_recall: 0.7803
Epoch 18/50
10000/10000 [==============================] - 18s 2ms/sample - loss: 0.4637 - acc: 0.8471 - precision: 0.8834 - recall: 0.8111
1563/1563 [==============================] - 665s 426ms/step - loss: 0.5146 - acc: 0.8232 - precision: 0.8799 - recall: 0.7761 - val_loss:
0.4635 - val_acc: 0.8471 - val_precision: 0.8834 - val_recall: 0.8111
Epoch 19/50
10000/10000 [==============================] - 18s 2ms/sample - loss: 0.4589 - acc: 0.8483 - precision: 0.8864 - recall: 0.8149
1563/1563 [==============================] - 670s 429ms/step - loss: 0.5112 - acc: 0.8255 - precision: 0.8790 - recall: 0.7768 - val_loss:
0.4589 - val_acc: 0.8483 - val_precision: 0.8864 - val_recall: 0.8149
```

① 다음과 같은 모델 training 실행화면 (맨 뒷부분 10줄 정도만)

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/2
50000/50000 [==============================] - 1566s 31ms/step - loss: 0.3772 -
accuracy: 0.8973 - val_loss: 0.3251 - val_accuracy: 0.9000
Epoch 2/2
50000/50000 [==============================] - 1568s 31ms/step - loss: 0.3251 -
accuracy: 0.9000 - val_loss: 0.3251 - val_accuracy: 0.9000
```

② train accuracy 와 test accuracy 출력 부분

평가기준 :
프로그램의 정상적으로 실행여부 : 5점
test accuracy : 5점