```
%tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, BatchNormalization, Input

import time
```

이전 CNN 코드

```
(raw_train_x, raw_train_y), (raw_test_x, raw_test_y) = tf.keras.datasets.mnist.load_data()

train_x = raw_train_x/255
test_x = raw_test_x/255

train_x = train_x.reshape((60000, 28, 28, 1))
test_x = test_x.reshape((10000, 28, 28, 1))

train_y = raw_train_y
test_y = raw_test_y
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

```
# 실습 시에는 시간 관계로 일부만 사용한다.
train_x = train_x[:10000] # ADDED
train_y = train_y[:10000] # ADDED


model = keras.Sequential()
model.add(Input((28,28,1)))
model.add(Conv2D(32, (3, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0821 01:28:29.673068 139727351523200 deprecation.py:506] From /usr/local/lib/python3.6/dist-
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
_________________________________________________________________
dense_1 (Dense)              (None, 10)                110
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
=================================================================
Total params: 35,046
Trainable params: 35,046
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 8s 757us/sample - loss: 1.6464 - acc: 0.3698
Epoch 2/5
10000/10000 [==============================] - 7s 698us/sample - loss: 0.7883 - acc: 0.7686
Epoch 3/5
10000/10000 [==============================] - 7s 683us/sample - loss: 0.5142 - acc: 0.8504
Epoch 4/5
10000/10000 [==============================] - 7s 689us/sample - loss: 0.3651 - acc: 0.8965
Epoch 5/5
10000/10000 [==============================] - 7s 684us/sample - loss: 0.2845 - acc: 0.9191
10000/10000 [==============================] - 3s 284us/sample - loss: 0.2775 - acc: 0.9220
loss= 0.2775210696309805
acc= 0.922
[7 2 1 ... 4 5 6]
```

CIFAR10 적용

refered https://www.cs.toronto.edu/~kriz/cifar.html

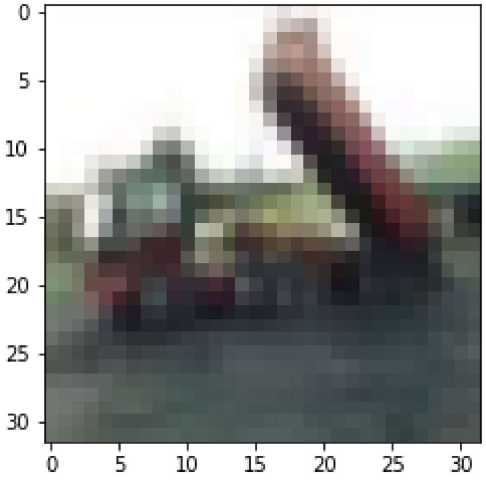
```
# (raw_train_x, raw_train_y), (raw_test_x, raw_test_y) = tf.keras.datasets.mnist.load_data()
(raw_train_x, raw_train_y), (raw_test_x, raw_test_y) = tf.keras.datasets.cifar10.load_data()

print(raw_train_x.shape)
print(raw_train_y.shape)
print(raw_test_x.shape)
print(raw_test_y.shape)
```

```
train_x = raw_train_x/255
test_x = raw_test_x/255

# train_x = train_x.reshape((60000, 28, 28, 1)) # COMMENT OUT
# test_x = test_x.reshape((10000, 28, 28, 1)) # COMMENT OUT

train_y = raw_train_y
test_y = raw_test_y
```

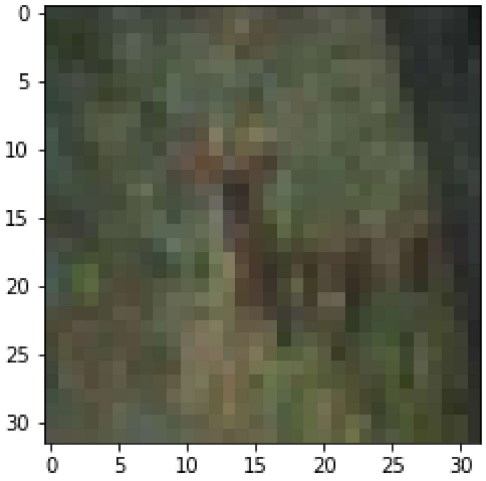
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 11s 0us/step
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
```

```
labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

def show_sample(i):
  print(raw_train_y[i][0], labels[raw_train_y[i][0]])
  plt.imshow(raw_train_x[i])
  plt.show()

for i in [2, 10, 12, 14]:
  show_sample(i)
```

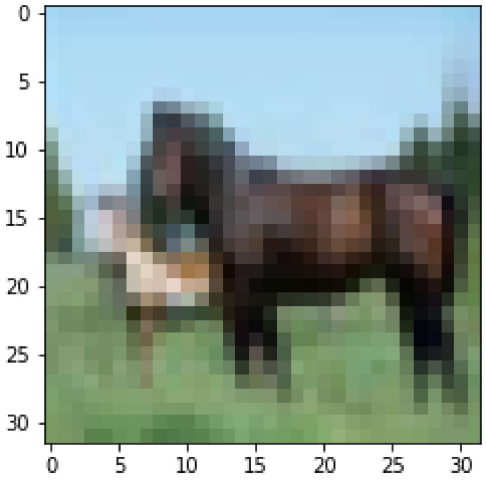
```
9 truck
```



```
4 deer
```



```
7 horse
```



```
9 truck
```



```
# 실습 시에는 시간 관계로 일부만 사용한다.
train_x = train_x[:10000] # ADDED
train_y = train_y[:10000] # ADDED
```

```
model = keras.Sequential()
# model.add(Input((28,28,1)))
model.add(Input((32,32,3))) # 이 부분이 달라진 부분
model.add(Conv2D(32, (3, 3)))
model.add(MaxPooling2D((2, 2)))
```

```
model.add(Conv2D(64, (3, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()


model.fit(train_x, train_y, epochs=100, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
79/79 [==============================] - 1s 8ms/step - loss: 0.2142 - accuracy: 0.9341
Epoch 69/100
79/79 [==============================] - 1s 8ms/step - loss: 0.2099 - accuracy: 0.9356
Epoch 70/100
79/79 [==============================] - 1s 8ms/step - loss: 0.2195 - accuracy: 0.9298
Epoch 71/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1859 - accuracy: 0.9447
Epoch 72/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1819 - accuracy: 0.9458
Epoch 73/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1848 - accuracy: 0.9465
Epoch 74/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1762 - accuracy: 0.9465
Epoch 75/100
79/79 [==============================] - 1s 8ms/step - loss: 0.2022 - accuracy: 0.9350
Epoch 76/100
79/79 [==============================] - 1s 8ms/step - loss: 0.2424 - accuracy: 0.9188
Epoch 77/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1795 - accuracy: 0.9431
Epoch 78/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1680 - accuracy: 0.9488
Epoch 79/100
79/79 [==============================] - 1s 7ms/step - loss: 0.1295 - accuracy: 0.9664
Epoch 80/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1218 - accuracy: 0.9685
Epoch 81/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1084 - accuracy: 0.9757
Epoch 82/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1104 - accuracy: 0.9733
Epoch 83/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1007 - accuracy: 0.9765
Epoch 84/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1088 - accuracy: 0.9745
Epoch 85/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1083 - accuracy: 0.9728
Epoch 86/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0971 - accuracy: 0.9773
Epoch 87/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0868 - accuracy: 0.9794
Epoch 88/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0949 - accuracy: 0.9767
Epoch 89/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1566 - accuracy: 0.9507
Epoch 90/100
79/79 [==============================] - 1s 8ms/step - loss: 0.2663 - accuracy: 0.9075
Epoch 91/100
79/79 [==============================] - 1s 8ms/step - loss: 0.2454 - accuracy: 0.9166
Epoch 92/100
79/79 [==============================] - 1s 8ms/step - loss: 0.1427 - accuracy: 0.9524
Epoch 93/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0856 - accuracy: 0.9792
Epoch 94/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0723 - accuracy: 0.9842
Epoch 95/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0651 - accuracy: 0.9863
Epoch 96/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0612 - accuracy: 0.9878
Epoch 97/100
```

```
79/79 [==============================] - 1s 8ms/step - loss: 0.0584 - accuracy: 0.9893
Epoch 98/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0503 - accuracy: 0.9915
Epoch 99/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0504 - accuracy: 0.9914
Epoch 100/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0438 - accuracy: 0.9925
313/313 [==============================] - 1s 3ms/step - loss: 4.3761 - accuracy: 0.5017
loss= 4.376052379608154
acc= 0.5016999840736389
[6 8 8 ... 5 1 7]
```