```
"""
プログラム１：CNNモデルを構築し、画像データを学習させます。
"""
%tensorflow_version 1.x

import os
import matplotlib.pyplot as plt
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization
from keras import regularizers
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
import random

os.chdir('/content/drive/My Drive/Python/CNN/')

num_classes = 10
im_rows = 32
im_cols = 32
in_shape = (im_rows, im_cols, 3)

#  データを読み込む
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

#  データを正規化
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
#  ラベルデータをOne-Hot形式に変換
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
#  検証データとテストデータに分ける
mask = list(range(10000))
random.shuffle(mask)
x_val = x_test[mask[:5000]]
```

```
y_val = y_test[mask[:5000]]
x_test = x_test[mask[5000:]]
y_test = y_test[mask[5000:]]
#  データ拡張
train_datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    fill_mode="nearest",
    horizontal_flip=True
)
train_generator = train_datagen.flow(x_train, y_train, batch_size=512)


#  モデルを定義
model = Sequential()
model.add(Conv2D(128, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05),
    input_shape = in_shape))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Conv2D(128, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(256, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Conv2D(256, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Conv2D(256, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2, 2)))
```

```
model.add(Dropout(0.3))

model.add(Conv2D(512, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Conv2D(512, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Conv2D(512, (3, 3), padding="same", kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(512, kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.3))
model.add(Dense(num_classes, kernel_regularizer= regularizers.l1(1e-05)))
model.add(BatchNormalization())
model.add(Activation("softmax"))
model.summary()

#  モデルをコンパイル
model.compile(
    loss = "categorical_crossentropy",
    optimizer = "adadelta",
    metrics = ["accuracy"])

#  学習を実行
reduce_lr = ReduceLROnPlateau(monitor='accuracy', factor=0.8, patience=5, ¥
    verbose=1, mode='auto', epsilon=0.01, cooldown=0, min_lr=0.0001)
filepath = 'weights_max.hdf5'
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, ¥
    save_best_only=True, mode='auto')
hist = model.fit_generator(train_generator,
    steps_per_epoch=300
```

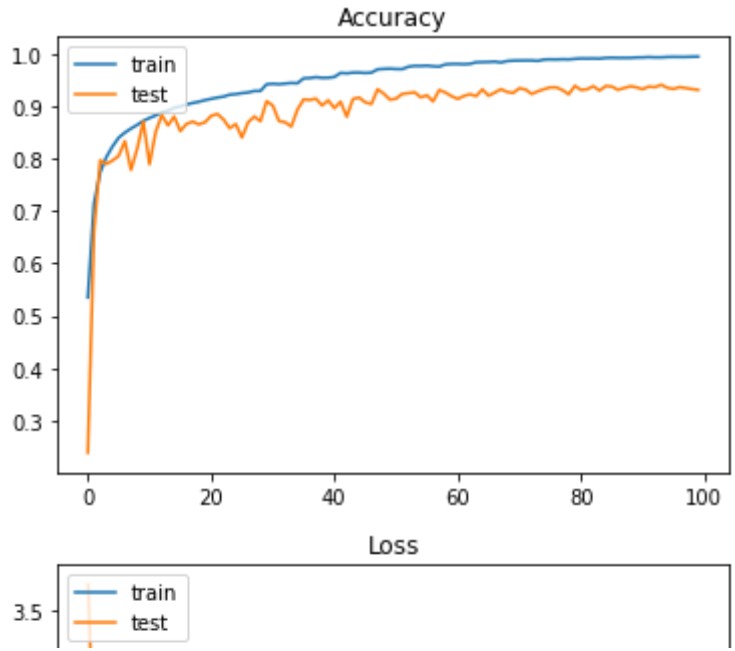
```
    steps_per_epoch=100,
    epochs=100,
    verbose=1,
    validation_data=(x_val, y_val),
    callbacks=[reduce_lr, checkpoint])

#  モデルを評価
score = model.evaluate(x_test, y_test, verbose = 1)
print("正確率=", score[1], "loss=", score[0])

#  学習の様子をグラフへ描画
plt.plot(hist.history["accuracy"])
plt.plot(hist.history["val_accuracy"])
plt.title("Accuracy")
plt.legend(["train", "test"], loc = "upper left")
plt.show()

plt.plot(hist.history["loss"])
plt.plot(hist.history["val_loss"])
plt.title("Loss")
plt.legend(["train", "test"], loc = "upper left")
plt.show()

#  学習結果を保存
model.save("cifar10-cnn-weight.h5")
```

```
TensorFlow 1.x selected.
Using TensorFlow backend.
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 14s 0us/step
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable._
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4070: The name tf.nn.max_pool is deprecate

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 128)       3584
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 256)       295168
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 256)       590080
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 256)       590080
```

batch_normalization_5 (Batch	(None, 16, 16, 256)	1024
activation_5 (Activation)	(None, 16, 16, 256)	0
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 256)	0
dropout_2 (Dropout)	(None, 8, 8, 256)	0
conv2d_6 (Conv2D)	(None, 8, 8, 512)	1180160
batch_normalization_6 (Batch	(None, 8, 8, 512)	2048
activation_6 (Activation)	(None, 8, 8, 512)	0
conv2d_7 (Conv2D)	(None, 8, 8, 512)	2359808
batch_normalization_7 (Batch	(None, 8, 8, 512)	2048
activation_7 (Activation)	(None, 8, 8, 512)	0
conv2d_8 (Conv2D)	(None, 8, 8, 512)	2359808
batch_normalization_8 (Batch	(None, 8, 8, 512)	2048
activation_8 (Activation)	(None, 8, 8, 512)	0
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 512)	0
dropout_3 (Dropout)	(None, 4, 4, 512)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 512)	4194816
batch_normalization_9 (Batch	(None, 512)	2048
activation_9 (Activation)	(None, 512)	0
dropout_4 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

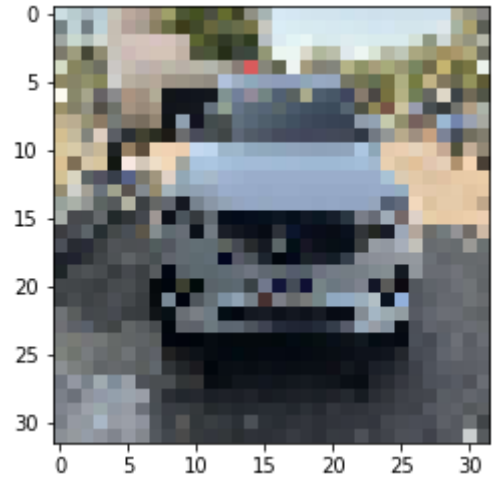
```
_________________________________________________________________
batch_normalization_10 (Batc (None, 10)                40
_________________________________________________________________
activation_10 (Activation)   (None, 10)                0
=================================================================
Total params: 11,738,546
Trainable params: 11,732,382
Non-trainable params: 6,164
_________________________________________________________________
/usr/local/lib/python3.6/dist-packages/keras/callbacks/callbacks.py:998: UserWarning: `epsilon` argument is deprecated and will be removed, u
  warnings.warn('`epsilon` argument is deprecated and '
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is depre

Epoch 1/100
300/300 [==============================] - 184s 613ms/step - loss: 2.8823 - accuracy: 0.5348 - val_loss: 3.7329 - val_accuracy: 0.2384

Epoch 00001: val_accuracy improved from -inf to 0.23840, saving model to weights_max.hdf5
Epoch 2/100
300/300 [==============================] - 172s 572ms/step - loss: 2.1503 - accuracy: 0.7140 - val_loss: 2.1939 - val_accuracy: 0.6610

Epoch 00002: val_accuracy improved from 0.23840 to 0.66100, saving model to weights_max.hdf5
Epoch 3/100
300/300 [==============================] - 172s 573ms/step - loss: 1.7511 - accuracy: 0.7733 - val_loss: 1.5888 - val_accuracy: 0.7966

Epoch 00003: val_accuracy improved from 0.66100 to 0.79660, saving model to weights_max.hdf5
Epoch 4/100
300/300 [==============================] - 172s 574ms/step - loss: 1.4855 - accuracy: 0.8026 - val_loss: 1.4691 - val_accuracy: 0.7894

Epoch 00004: val_accuracy did not improve from 0.79660
Epoch 5/100
300/300 [==============================] - 173s 575ms/step - loss: 1.2938 - accuracy: 0.8230 - val_loss: 1.3414 - val_accuracy: 0.7960

Epoch 00005: val_accuracy did not improve from 0.79660
Epoch 6/100
300/300 [==============================] - 173s 577ms/step - loss: 1.1546 - accuracy: 0.8396 - val_loss: 1.2223 - val_accuracy: 0.8048

Epoch 00006: val_accuracy improved from 0.79660 to 0.80480, saving model to weights_max.hdf5
Epoch 7/100
300/300 [==============================] - 172s 574ms/step - loss: 1.0665 - accuracy: 0.8493 - val_loss: 1.0964 - val_accuracy: 0.8330

Epoch 00007: val_accuracy improved from 0.80480 to 0.83300, saving model to weights_max.hdf5
Epoch 8/100
```

```
Epoch 00096: val_accuracy did not improve from 0.94060
Epoch 97/100
300/300 [==============================] - 171s 570ms/step - loss: 0.2322 - accuracy: 0.9938 - val_loss: 0.4669 - val_accuracy: 0.9362

Epoch 00097: val_accuracy did not improve from 0.94060
Epoch 98/100
300/300 [==============================] - 171s 570ms/step - loss: 0.2309 - accuracy: 0.9938 - val_loss: 0.4880 - val_accuracy: 0.9342

Epoch 00098: val_accuracy did not improve from 0.94060
Epoch 99/100
300/300 [==============================] - 171s 571ms/step - loss: 0.2284 - accuracy: 0.9942 - val_loss: 0.4799 - val_accuracy: 0.9324

Epoch 00099: ReduceLROnPlateau reducing learning rate to 0.04398046135902405.

Epoch 00099: val_accuracy did not improve from 0.94060
Epoch 100/100
300/300 [==============================] - 171s 571ms/step - loss: 0.2258 - accuracy: 0.9944 - val_loss: 0.4950 - val_accuracy: 0.9308

Epoch 00100: val_accuracy did not improve from 0.94060
5000/5000 [==============================] - 3s 651us/step
正確率= 0.925599992275238 loss= 0.5341672481536865
```



"""

```
"""
プログラム２：プログラム２で学習済みのCNNモデルを使って、画像を判定させます。
"""
import os
import cv2
import numpy as np
from keras.models import load_model
import matplotlib.pyplot as plt
labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]
im_shape = (32, 32, 3)

os.chdir('/content/drive/My Drive/Python/CNN/')

#  モデルデータを読み込み
model = load_model("cifar10-cnn-weight.h5")

#  OpenCVを使って画像を読み込み
im = cv2.imread("test-car.jpg")
#  色空間を変換して、リサイズ
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im = cv2.resize(im, (32, 32))
plt.imshow(im)
plt.show()

#  MLPで学習した画像データに合わせる
im = im.reshape(im_shape).astype("float32") / 255
#  予測する
r = model.predict(np.array([im]), batch_size = 32, verbose = 1)
res = r[0]
#  結果を表示する
for i, acc in enumerate(res):
    print(labels[i], "=", int(acc * 100))
print("---")
print("予測した結果:", labels[res.argmax()])
```




```
1/1 [==============================] - 1s 736ms/step
airplane = 0
automobile = 99
bird = 0
cat = 0
deer = 0
dog = 0
frog = 0
horse = 0
ship = 0
truck = 0
---
予測した結果: automobile
```