如果我们对灵活性要求很高，并希望能够用自己定制的层来构建神经网络，直接使用TensorFlow，但很多时候，可以通过诸如Keras这种友好的接口就可以直接实现目标；而且它的效率更高，界面也更简洁。

[Keras](https://faroit.github.io/keras-docs/1.2.1/) makes coding deep neural networks simpler. To demonstrate just how easy it is, you're going to build a simple fully-connected network in a few dozen lines of code.

The **[keras.models.Sequential](https://keras.io/models/sequential/" \t "_blank)** class is a wrapper for the neural network model. It provides common functions like fit(), evaluate(), and compile(). We'll cover these functions as we get to them. Let's start looking at the layers of the model.

import pickle

import numpy as np

import tensorflow as tf

# Load pickled data

with open('small\_train\_traffic.p', mode='rb') as f:

data = pickle.load(f)

# Setup Keras

from keras.models import Sequential

from keras.layers.core import Dense, Activation, Flatten, Dropout

from keras.layers.convolutional import Conv2D

from keras.layers.pooling import MaxPool2D

# Build the Fully Connected Neural Network in Keras Here

model = Sequential()

model.add(Conv2D(32,(3,3),input\_shape=(32,32,3)))

model.add(MaxPool2D((2,2)))

model.add(Dropout(0.5))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(128))

model.add(Activation('relu'))

model.add(Dense(5))

model.add(Activation('softmax'))

# preprocess data

X\_normalized = np.array(X\_train / 255.0 - 0.5 )

from sklearn.preprocessing import LabelBinarizer

label\_binarizer = LabelBinarizer()

y\_one\_hot = label\_binarizer.fit\_transform(y\_train)

model.compile('adam', 'categorical\_crossentropy', ['accuracy'])

history = model.fit(X\_normalized, y\_one\_hot, epochs=15, validation\_split=0.2)

# evaluate model against the test data

with open('small\_test\_traffic.p', 'rb') as f:

data\_test = pickle.load(f)

X\_test = data\_test['features']

y\_test = data\_test['labels']

# preprocess data

X\_normalized\_test = np.array(X\_test / 255.0 - 0.5 )

y\_one\_hot\_test = label\_binarizer.fit\_transform(y\_test)

print("Testing")

metrics = model.evaluate(X\_normalized\_test, y\_one\_hot\_test)

for metric\_i in range(len(model.metrics\_names)):

metric\_name = model.metrics\_names[metric\_i]

metric\_value = metrics[metric\_i]

print('{}: {}'.format(metric\_name, metric\_value))