实验二：卷积神经网络

一、实验简介

本实验的目的是搭建卷积神经网络(CNN)来解决图像分类问题，将搭建一个简易的卷积神经网络对一个简单的实物数据集进行识别和分类。

在开始实验前需要安装好 python 以及相应的集成开发环境 IDE，并安装好需要用到的 python 库。

本实验的示例中用到的 python 库主要有 Keras、Numpy、matplotlib、tensorflow。

二、图像分类：多分类问题

2.1 CIFAR-10 数据集

CIFAR-10 是一个更接近普适物体的彩色图像数据集。CIFAR-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型数据集。一共包含 10 个类别的彩色图片：飞机(airplane)、汽车(automobile)、鸟类(bird)、猫(cat)、鹿(deer)、狗(dog)、蛙类(frog)、马(horse)、船(ship)和卡车(truck)。

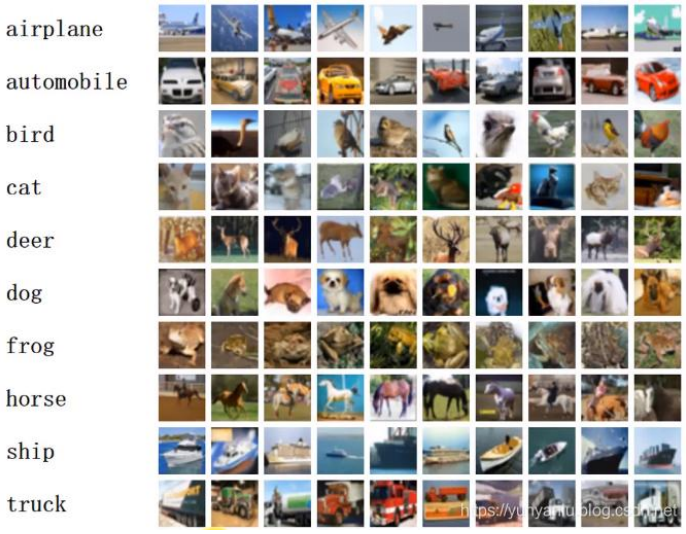

图 2-1　CIFAR-10 数据集

每个图片的尺寸为 32×32 ，每个类别有 6000 个图像，数据集中一共有 50000 张训练图片和 10000 张测试图片。与 MNIST 数据集中手写数字的灰度图不同，

CIFAR-10 数据集中为 3 通道 RGB 彩色图片，包含训练数据、训练标签、测试数据、测试标签，大小分别为(50000, 32, 32, 3)、(50000, 1)、(10000, 32, 32, 3)、(10000, 1)。

不同于手写字符， CIFAR-10 含有的是现实世界中真实的物体，不仅噪声很大，而且物体的比例、特征都不尽相同，这为识别带来很大困难，因此相比于全连接网络，卷积神经网络(CNN)将会有比较明显的优势。不过由于图片本身复杂度高和分辨率低等原因，使用一个简单的网络进行分类准确率不会太高，在 60%~80%左右均属正常。

数据集可以从官方网址 https://www.cs.toronto.edu/~kriz/cifar.html 下载，也可以通过 Keras 库中的 datasets.cifar10.load_data()直接加载。

2.2 加载 CIFAR-10 数据集

```
import tensorflow as tf
from tensorflow import keras
from keras.datasets import cifar10

(train_data, train_label), (test_data, test_label) = cifar10.load_data()
```

train_data 和 test_data 是分别是 50000 张和 10000 张 32×32 的彩色图像数据，train_label 和 test_label 都是 0~9 的数字，分别代表 10 个类别。

2.3 数据预处理

将数据输入神经网络之前，应该将数据格式化为经过预处理的浮点数张量。在本例中，图像数据被读取为 Numpy 的 N 维数组对象 ndarray，它是一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引。需要将 RGB 像素值（0~255 范围内）转换为浮点数张量，并缩放到[0, 1] 区间（神经网络适合处理较小的输入值）。

```
x_data = train_data.astype('float32') / 255.
y_data = test_data.astype('float32') / 255.
```

2.4 标签预处理

对于多分类任务，通常会对标签进行 one-hot 编码，将其转换为 0 和 1 组成的向量。在本例中，总共有 10 类不同的物体，标签值在 0~9 范围内，因此 one-hot 编

码后的向量长度为 10。举个例子，原本的标签值[2]会被转化为向量[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]，只有索引为 2 的元素是 1，其余元素都是 0。

```
import numpy as np

def one_hot(label, num_classes):
    # your code here# 补充 one-hot 编码函数

    return label_one_hot

num_classes = 10
train_label = train_label.astype('int32')
train_label = np.squeeze(train_label)
x_label = one_hot(train_label, num_classes)
test_label = test_label.astype('int32')
y_label = np.squeeze(test_label)
```

预处理过后的标签：

```
>>> train_label[0:5]
>>> x_label[0:5]
[6 9 9 4 1]
[[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.]]
```

三、卷积神经网络(CNN)

3.1 构建网络

在本实验中，将复用相同的总体结构，即卷积神经网络由 Conv2D 层（使用 relu 激活）和 MaxPooling2D 层交替堆叠构成。初始输入的尺寸为(32, 32, 3)，经过两个卷积单元后变为(8, 8, 64)，经过 Flatten 层打平为向量后送入全连接层。

如果处理更大的图像和更复杂的问题，需要相应地增大网络，即再增加一些 Conv2D+MaxPooling2D 的组合。这既可以增大网络容量，也可以进一步减小特征图的尺寸，使其在连接 Flatten 层时尺寸不会太大。

该问题是一个多分类问题，所以网络最后一层是使用 softmax 激活函数（大小为 10 的 Dense 层），这一层将对某个类别的概率进行编码。此外，还设置了一些 dropout 层，在训练过程中随机将该层的一些输出特征舍弃（设置为 0），来避免过拟合。

```
from keras import Sequential
from keras.layers import Convolution2D, MaxPooling2D, Dense, Flatten, Dropout

cnn = Sequential()
#unit1
cnn.add(Convolution2D(32, kernel_size=[3, 3], input_shape=(32, 32, 3), activation='relu', padding='same'))
cnn.add(Convolution2D(32, kernel_size=[3, 3], activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=[2, 2], padding='same'))
cnn.add(Dropout(0.5))

#unit2
# 编写网络的第二部分，可自行尝试增加更多的卷积层，改变通道数、激活函数等，以下设置仅供参考：(两个 2D 卷积层，均为 64 个通道，卷积核为(3, 3)，激活函数为 relu，padding 为 same；一个 2D 池化层，pool_size 为(2, 2)，padding 为 same，最后是 Dropout 层，保留概率为 0.5)
# your code here#



cnn.add(Flatten())

cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(128, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(10, activation='softmax'))
```

看一下特征图的维度如何随着每层变化。

```
>>> cnn.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
=================================================================
Total params: 65,568
Trainable params: 65,568
Non-trainable params: 0
_________________________________________________________________
```

3.2 编译模型

在编译这一步，将使用 Adam 优化器，学习率为 0.001。因为网络最后一层是 softmax 激活函数，所以使用交叉熵(categorical_crossentropy)作为损失函数。

```
# 采用 model.compile 模型编译，设置优化器、学习率、损失函数和准确率观测
# your code here
```

3.3 训练模型

使用 32 个样本组成的小批量，将模型训练 50 个轮次（即对 train_data 和 y_train 两个张量中的所有样本进行 50 次迭代）。同时，分离出 1/10 的训练数据作为验证集，不对其进行训练，并且将在每个时期结束时评估此数据的损失和任何模型度量。

```
history_cnn = cnn.fit(x_data, x_label, epochs=50, batch_size=32, shuffle=True, verbose=1, validation_split=0.1)
```

在 Nvidia RTX2060 GPU 上运行，每轮的时间在 12 秒左右，训练过程将在 10 分钟左右结束。每轮结束时会有短暂的停顿，因为模型要计算在验证集的样本上的损失和精度。

注意，调用 model.fit() 返回了一个 History 对象。这个对象有一个成员 history，它是一个字典，包含训练过程中的所有数据。

```
>>> history_dict = history.history
>>> history_dict.keys()
dict_keys(['val_acc', 'acc', 'val_loss', 'loss'])
```

字典中包含 4 个条目，对应训练过程和验证过程中监控的指标。在下面两个代码清单中，可以使用 Matplotlib 在同一张图上绘制训练损失和验证损失，以及训练精度和验证精度。请注意，由于网络的随机初始化不同，得到的结果可能会略有不同。

3.4 绘制损失和精度图

```
import matplotlib.pyplot as plt

plt.figure(1)
plt.plot(np.array(history_cnn.history['loss']))
plt.plot(np.array(history_cnn.history['val_loss']))
plt.xlabel('Epoch')
plt.ylabel('Train loss')
plt.legend(['loss', 'val_loss'])
plt.show()
plt.figure(2)
plt.plot(np.array(history_cnn.history['acc']))
plt.plot(np.array(history_cnn.history['val_acc']))
plt.xlabel('Epoch')
plt.ylabel('Train acc')
plt.legend(['acc', 'val_acc'])
plt.show()
```

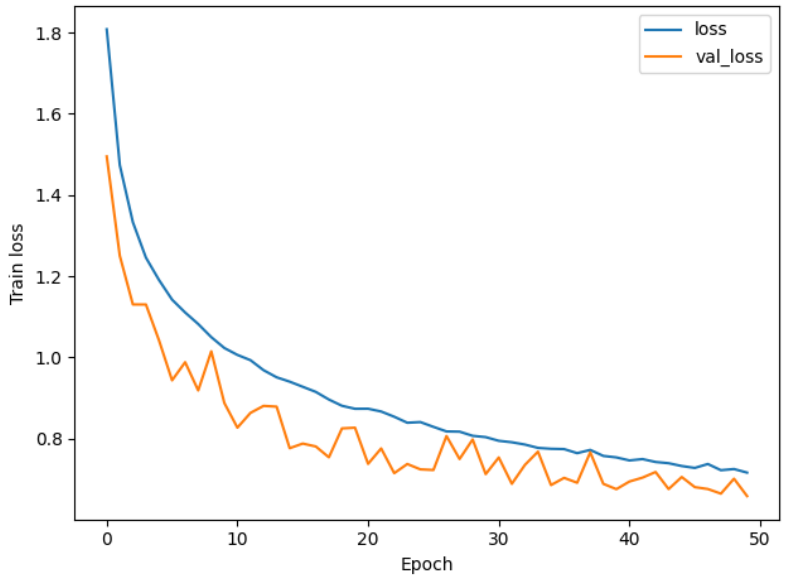

图 3-1　训练损失和验证损失

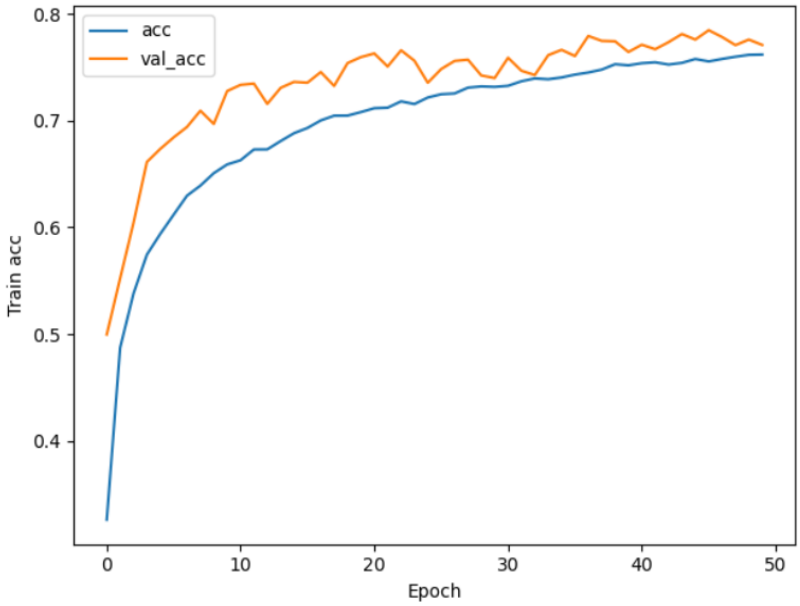

图 3-2　训练精度和验证精度

如图所示，训练损失整体呈下降趋势，训练精度整体呈上升趋势。

3.5 保存模型

```
cnn.save('model/cnn.h5')
```

3.6 在新数据上生成预测结果

加载保存的模型并输入测试数据得到预测结果。

```
cnn = keras.models.load_model('model/cnn.h5')
test_out = cnn.predict(y_data)
```

```
>>> y_label[0:3]
[3 8 8]

>>> cnn.pridect(y_data[0:3])
[[1.3347306e-04 3.0984458e-05 3.6098459e-03 8.6117876e-01 1.0844456e-03 8.5025288e-02
4.8267592e-02 4.7753484e-04 1.1310063e-04 7.9002835e-05]
 [2.6698842e-06 2.4105470e-06 1.4300645e-12 6.7049567e-13 1.9550720e-13 4.4489105e-15
2.7995191e-16 1.9613197e-14 9.9999487e-01 2.7010516e-08]
 [1.0718202e-01 1.3392895e-01 5.4476517e-03 6.0969568e-03 4.1209986e-03 1.1701657e-03
4.0965888e-04 2.0788626e-03 6.4716345e-01 9.2401281e-02]]
```

输出长度为 10 的向量中每个值代表对应类别的概率，其最大值对应的类别即模型的预测结果。

3.7 测试模型准确率

将 10000 个测试数据输入模型中，得到预测结果并计算准确率，最终得到模型的预测准确率为 76.93%。

```
cnn = keras.models.load_model('model/cnn.h5')
test_out = cnn.predict(y_data)

num = 0
total_num = y_data.shape[0]

for i in range(total_num):
    predict = np.argmax(test_out[i])
    if predict == y_label[i]:
        num += 1
accuracy = num / total_num
```

```
>>> accuracy
0.7693
```