# 华东理工大学

# 模式识别课程报告

—— BP 神经网络、卷积神经网络原理介绍

与图像识别应用

**题目** CIFAR-10 十分类

**院系** 信息科学与工程学院

**专业** 控制科学与工程

**组员** 王志强

**指导老师** 赵海涛

目录

一、 选题介绍.....1
1.1 原 kaggle 选题.....1
1.2 CIFAR-10 数据集介绍.....2
二、 实验目的.....2
三、 原理介绍.....3
3.1 BP 神经网络.....3
3.1.1 神经元模型.....3
3.1.2 误差反向传播.....3
3.1.3 更新权重.....4
3.2 卷积神经网络.....4
3.2.1 卷积核与卷积.....4
3.2.2 卷积层.....5
3.2.3 池化层.....6
3.2.4 激活层.....6
3.2.5 全连接层.....6
3.2.6 交叉熵损失.....6
3.2.7 参数更新.....7
3.2.8 Tensorflow.....8
四、 关键程序介绍.....9
4.1 BP 神经网络部分.....9
4.2 卷积神经网络部分.....10
五、 实验结论与分析.....11
5.1 BP 神经网络结果与分析.....11
5.2 卷积神经网络结果与分析.....12
六、 总结与心得.....13
七、 参考文献.....15
附：文件说明.....15
Python 代码附录.....16
1、 BP 神经网络对 kaggle 选题.....16
2、 BP 神经网络对 CIFAR-10.....20
3、 卷积神经网络对 CIFAR-10.....22

# 一、选题介绍

## 1.1 原 kaggle 选题

题目网址：https://www.kaggle.com/c/image-classification2

最初选定的题目是 kaggle 上的一道彩色图像分类题目，即有三类图片：汽车、摩托车和自行车；要求参与者正确识别它们，方法不限。数据集有：训练集包含 231 张自行车图片、518 张汽车图片和 218 张摩托车图片；测试集包含了 914 张三类混合的图片。数据实例如图 1 所示。

图 1　训练集选图

```
<ipython-input-3-4addb82cffb0> in train(self, inputs_list, targets_list)
     46
     47         # 更新输入层至隐藏层的权重
---> 48         self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1
s)), numpy.transpose(inputs))
     49         pass
     50

MemoryError: Unable to allocate array with shape (800, 562500) and data type float64
```

图 2　jupyter notebook 报错

然后在接下来的实验中遇到了内存不足的问题。如图 2 所示，编译器提示“储存错误：无法分配 800×562500 的数组来存储浮点型数字数据”。题目所给的图片规模为：500*375*3，若用神经网络的方法则会产生数量庞大的变量。在我将电脑虚拟内存设置成最大后仍有这个问题。

最终我选择将数据集换成经典的、处理数据规模较小 CIFAR-10 来测试神经网络的性能。

## 1.2 CIFAR-10 数据集介绍

CIFAR-10 数据集由 10 个类的 60000 个 32x32 三通道彩色图像组成，每个类有 6000 个图像。总共有 50000 个训练图像和 10000 个测试图像。十个类别为：飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船和卡车。

数据集分为五个训练批次和一个测试批次，每个批次有 10000 个图像。测试批次包含来自每个类别的恰好 1000 个随机选择的图像。训练批次以随机顺序包含剩余图像，但一些训练批次可能包含来自一个类别的图像比另一个更多。总体来说，所有训练批组成的训练集，每一类都有 5000 张图。这些类完全相互排斥。汽车和卡车之间没有重叠。“汽车”包括轿车，SUV，这类东西。“卡车”只包括大卡车。都不包括皮卡车。

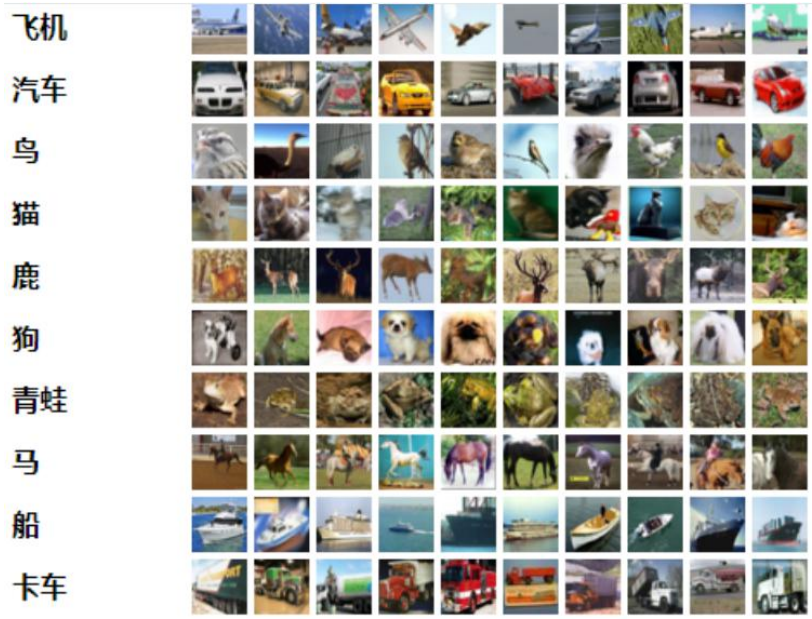


图 3　CIFAR-10 图像随机举例

# 二、实验目的

运用 BP 神经网络和卷积神经网络于 CIFAR-10 数据集进行图像识别的训练与测试，分析两种方法的实验结果。

实验工具：jupyter notebook ， python。

# 三、原理介绍

人工神经网络（Artificial Neutral Network，ANN）是由大量简单的基本元件——神经元相互连接，通过模拟人的大脑神经处理信息的方式，进行信息并行处理和非线性转换的复杂网络系统。

## 3.1 BP 神经网络

### 3.1.1 神经元模型

人造神经元模型如图 4 所示。该神经元的输入输出关系为：

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$

其中 $x_i$ 为神经元的输入，$w_i$ 为各神经元之间的权重，b 为神经元的阈值，y 为神经元的输出，$\sigma(\bullet)$ 为神经元的激活函数，常用的转化函数如图 5 所示。

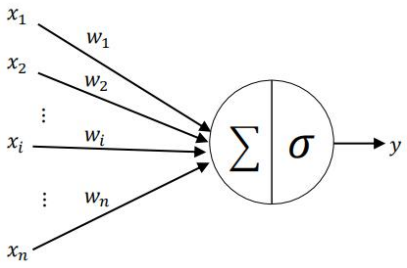


图 4　人造神经元

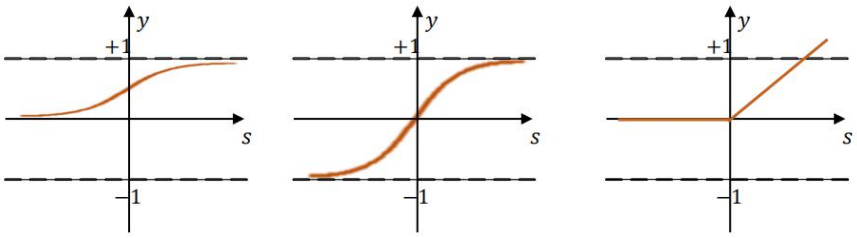


图 5　sigmoid、tanh、ReLU 函数

### 3.1.2 误差反向传播

三层神经网络实例如图 6。BP 神经网络误差的反向传播如图 7 所示。在 BP 神经网络中可以选择不等分误差传播，即为较大链接权重的连接分配更多的误差。因为这些链接对造成误差的贡献较大。如图 7，第一层 1 号节点至下层 1 号

节点的权重为 3.0，占总权重的 3/4，所以传播给它的误差是 3/4 的输出误差。

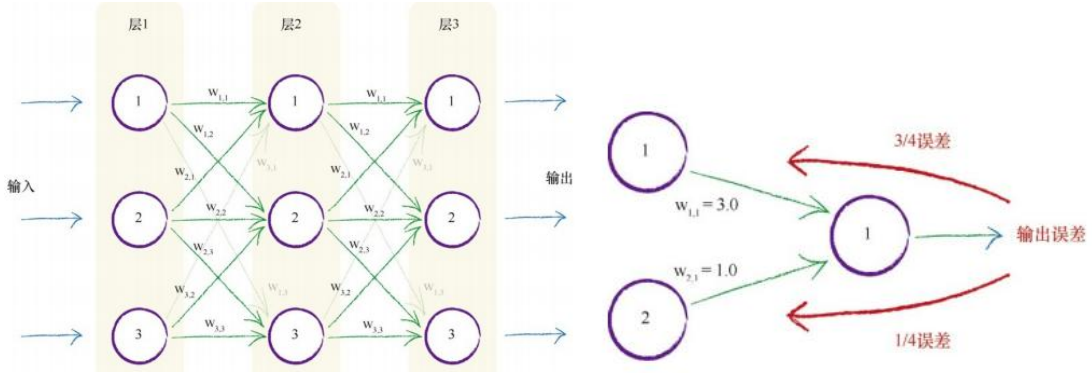


图 6、7　三层神经网络与误差反向传播

### 3.1.3 更新权重

可以运用梯度下降法更新神经网络中的权重参数。

$$w_{j,k}^{new} = w_{j,k}^{old} - \alpha \frac{\partial E}{\partial w_{j,k}}$$

式中，$\alpha$ 称为学习率。这个因子可以调节权重变化的强度，确保不会超调。

## 3.2 卷积神经网络

卷积神经网络（Convolutional Neural Network，CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。 它包括卷积层(convolutional layer)、池化层(pooling layer)和全连接层(fully connected layer)。

在图像识别中，图像的像素数量是网络输入层神经元的数量；卷积后得到的像素个数是卷积层中的神经元个数；池化后得的数据个数是池化层的神经元个数；参数更新是指对卷积核内权重以及全连接层中的权重数据的更新。

### 3.2.1 卷积核与卷积

卷积核又称滤波器（filter）。卷积作用以图 8 为例：输入数据是一个 8×8 的矩阵，卷积核是一个大小为 3×3 的矩阵，将卷积核在输入矩阵上从左向右、至上而下的平移，每次平移后，卷积核与新的部分输入数据重合对应。卷积作用就是将输入数据对应的 9 个数据加权求和得到一个数据。如图 8，设卷积核滑动

步长为 1，则卷积后得到一个 6×6 的输出矩阵。

卷积核中的数据决定了卷积核的作用。如图 8 的卷积核，它就可以用来检测图像边缘。

卷积核中的数字变量是卷积神经网络中需要训练的权重参数之一。

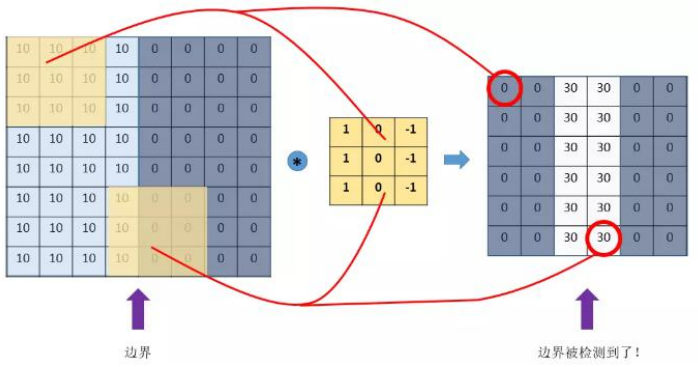


图 8　卷积作用

3.2.2 卷积层

卷积层就是用卷积核对输入图像数据进行卷积的阶段。以 CIFAR-10 数据集为例：每张图是 32*32*3 的三维数据，即长度和宽度为 32 个像素，颜色通道为 RGB 三通道；也就是说一张图的输入数据是三个 32×32 的数字矩阵，相应地，卷积核也是三维矩阵，例如三个 3×3 的矩阵，假设滑动步长为 1，那么卷积后就会得到三个 30×30 的输出矩阵。

一个卷积层常用多个卷积核。如图 9 是一个三通道图片的卷积过程，有四个卷积核，大小为 3×3×3，8×8×3 的输入数据经过卷积后得到 6×6×4（也可以说是 6×6×3×4）的输出。

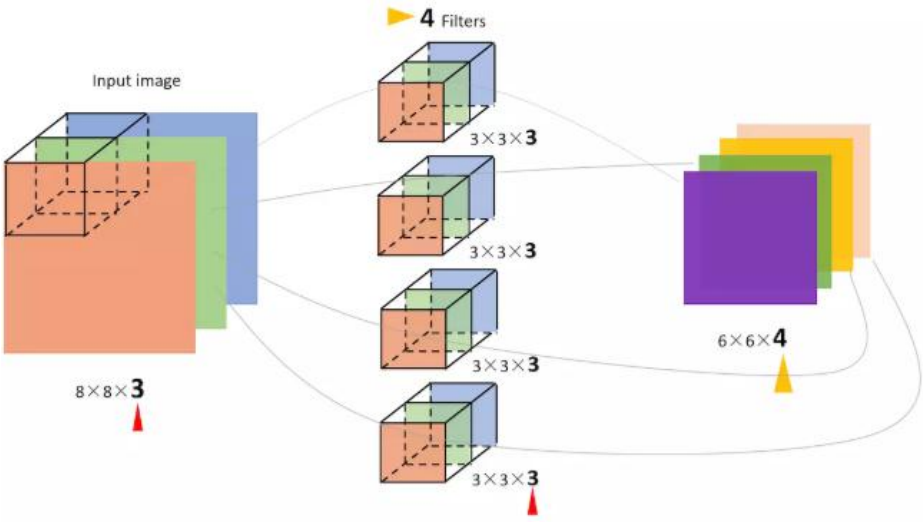


图 9　多通道图片的卷积

3.2.3 池化层

池化作用是为了提取一定区域的主要特征，并减少参数数量，防止模型过拟合。常用的有 Maxpooling，采用一个 2×2 的窗口，并设置滑动步长为 2，如图 10 所示。即取窗口中最大的数据作为输出数据，是一种下采样方式。

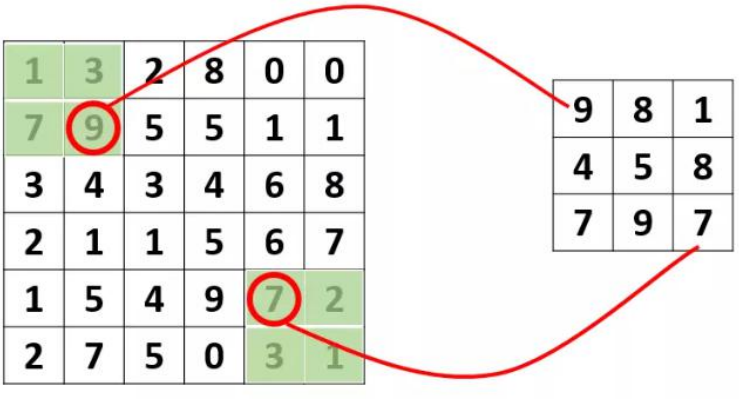


图 10　MAX 池化

3.2.4 激活层

卷积神经网络中的激活概念和 BP 神经网络中的是一致的。本次实验使用 ReLU 函数作为激活函数。

3.2.5 全连接层

全连接层的形式和 BP 神经网络中的隐藏层与输出层的形式一致。因为这一层是每一个单元都和前一层的每一个单元相连接，所以称之为“全连接”。具体实现中就是将数据矩阵依次拉成列向量后连接成为一个超长的列向量，每个数据都作为一个神经元，其输出就是数据数值，其余和 BP 神经网络一致。

3.2.6 交叉熵损失

举个例子。P 用表示样本的真实分布，比如[1，0，0]表示当前样本属于第一类。Q 用来表示神经网络预测的分布，比如[0.7，0.2，0.1]。直观的理解就是如果用 P 来描述样本就十分完美。若用 Q 来描述样本，信息量不足，需要额外的一些“信息增量”才能达到如 P 一样的描述。可以通过训练 Q，使其像 P 一样描述样本。用 KL 散度来描述 $p$ 、 $q$ 的联系：

$$D_{KL}(p\|q)=\sum_{i=1}^{n}p(x_i)\log(p(x_i))-\sum_{i=1}^{n}p(x_i)\log(q(x_i))$$
$$=-H(p(x))+[-\sum_{i=1}^{n}p(x_i)\log(q(x_i))]$$

式中，$p=[x_1,x_2,\cdots x_i]$，$p(x_i)$表示类为第 i 类的概率，对数据规范化后也就是$x_i$的数值，$q$形式一致。等式的前一部分是$p$的熵，后一部分就是交叉损失熵：

$$H(p,q)=-\sum_{i=1}^{n}p(x_i)\log(q(x_i))$$

3.2.7 参数更新

卷积神经网络的参数更新也可以使用梯度下降法，且在理论上的推导是和 BP 神经网络相一致的。

**全连接层梯度反向传播：**

全连接层之间传播可与 BP 神经网络一样，使用不等分误差传播。

假设$S=XW+b$，$X$是输入，$W$是权重，$b$是偏置，$S$是输出分值矩阵。若$\frac{\partial L}{\partial S}$已知，$L$是损失函数，则：$\frac{\partial L}{\partial X}=\frac{\partial L}{\partial S}\cdot W^T$，$\frac{\partial L}{\partial W}=X^T\cdot\frac{\partial L}{\partial S}$，$\frac{\partial L}{\partial b}$中每个元素是$\frac{\partial L}{\partial S}$每列之和。

**激活层梯度反向传播：**

假设采用 ReLU 作为激活函数。公式描述为：$H_{out}=\mathrm{Re}LU(H_{in})$，其中激活层输入和输出$H_{out}$、$H_{in}$。该公式是逐元计算的。已知$\frac{\partial L}{\partial H_{out}}$，求$\frac{\partial L}{\partial H_{in}}$。因为当输入小于 0 是，梯度为 0，当输入大于 0 时，梯度为 1。所以$H_{in}$的每个元素只需与 0 比较，若大于 0，则输出梯度等于输入梯度，否则为 0。

**卷积层梯度的反向传播：**

先简述利用矩阵乘法实现卷积层正向计算的过程：

（1）将输入特征图变成大矩阵。

（2）进行矩阵相乘和非线性激活后得到输出数据。

（3）将输出数据变换为输出特征图。

卷积层梯度传播就是已知输出特征图的梯度，求输入特征图的梯度及卷积核的梯度，其过程如下。

$$\frac{\partial L}{\partial out}$$

（1）把输出特征图的梯度 $\frac{\partial L}{\partial out}$ 变换为矩阵形式（正向计算第三步的逆过程）。

（2）将全连接层和激活层的梯度进行反向传播。

（3）把第（2）步得到的矩阵梯度变换为特征图形状的梯度，即得到输入特征图的梯度。

**卷积层梯度的反向传播：**

最大池化层是求 $f=\max(a,b,c,d)$ 的偏导数，其中 a,b,c,d 是池化窗口中的数据。该函数的偏导数：a,b,c,d 中的最大值的梯度为 1，其余为 0。 推到如下：

$$\frac{\max(a+da,b,c,d)-\max(a,b,c,d)}{da},da\to 0,$$

当 a 为最大值时，则上式变为：$(a+da-a)/da=1$；当 a 不为最大值，不妨设 b 为最大值，则上式变为 $(b-b)/da=0$。

所以，池化层梯度反向传播时，每个局部窗口中，若该元素是最大值，则该位置处的输出梯度等于输入梯度，否则为 0。公式原理如下：

$$\frac{\partial L}{\partial a}=\frac{\partial L}{\partial f}\cdot\frac{\partial f}{\partial a}=\frac{\partial L}{\partial f}\cdot 1=\frac{\partial L}{\partial f}\ ,a\text{最大}$$

$$\frac{\partial L}{\partial b}=\frac{\partial L}{\partial f}\cdot\frac{\partial f}{\partial b}=\frac{\partial L}{\partial f}\cdot 0=0\ ,b\text{ 最大}$$

3.2.8 Tensorflow

TensorFlow 是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用于各类机器学习（machine learning）算法的编程实现，其前身是谷歌的神经网络算法库 DistBelief。Tensorflow 拥有多层级结构，可部署于各类服务器、PC 终端和网页并支持 GPU 和 TPU 高性能数值计算，被广泛应用于谷歌内部的产品开发和各领域的科学研究。TensorFlow 由谷歌人工智能团队谷歌大脑（Google Brain）开发和维护，拥有包括 TensorFlow Hub、TensorFlow

Lite、TensorFlow Research Cloud 在内的多个项目以及各类应用程序接口（Application Programming Interface, API）。

本次实验卷积神经网络的实现使用了库 tensorflow。

# 四、关键程序介绍

## 4.1 BP 神经网络部分

在 BP 神经网络的输入层中，神经元个数是图像的每个像素。例如一个 375*500*3 的彩色图像，把它依次拉成列向量作为输入，也就是共有 375*500*3=562500 个神经元。拉成列向量的 python 代码实现如下。

```
# 将三维数据矩阵拉成列向量
for i in range(len(d_bicycle)):
    t1=d_bicycle[i].reshape(375*500*3,)
    t1=t1.reshape(-1,1)
    if i==0:
        da_bicycle=t1
    if i!=0:
        da_bicycle=np.append(da_bicycle,t1,axis=1)
```

为了便利，可以先定义一个 BP 神经网络类。类中包含网络各层神经元个数、对应于各层间的权重矩阵、学习率与激活函数的设置、网络的训练与运行功能函数。具体可见附录。

权重矩阵设置的代码例如下。wih 是输入层至隐藏层间的权重矩阵，随机初始化每个元素数值，可设置方差、均值。

```
self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))
```

训练或测试网络的输入输出数据需要归一化，即使数据值处于 0%至 100%之间。但是 0.0 和 1.0 这两个数有驱动产生过大的权重的风险，因此尝使用范围 0.01~0.99 替代。代码如下。

```
for record in da_bicycle:
        all_values=record
        inputs = all_values / 255.0 * 0.99 + 0.01     #所有数据归一化
```

```
        targets = numpy.zeros(output_nodes) + 0.01
        targets[0] = 0.99
        n.train(inputs, targets)
        pass
```

神经网络的训练代码如下。具体就是由之前原理公式推导指示的矩阵乘法运算。

```
def train(self, inputs_list, targets_list):
        # 输入行向量特征，处理为列向量
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T
        # 计算输入隐藏层的信号
        hidden_inputs = numpy.dot(self.wih, inputs)
        # 计算隐藏层输出信号
        hidden_outputs = self.activation_function(hidden_inputs)
        # 计算输出层输入信号
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # 计算最终输出信号
        final_outputs = self.activation_function(final_inputs)
        # 输出层误差为 (target - actual)
        output_errors = targets - final_outputs
        # 计算隐藏层输出误差
        hidden_errors = numpy.dot(self.who.T, output_errors)
        # 更新隐藏层至输出层的权重
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose(hidden_outputs))
        # 更新输入层至隐藏层的权重
        self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))
        Pass
```

## 4.2 卷积神经网络部分

卷积神经网络的代码在形式上与 BPNN 是相似的，在此介绍一下特色部分（卷积层）。

在卷积神经网络中，每个卷积层的组成部分是类似的。1、配置卷积核所需的存储空间；2、设置卷积功能函数（输入信号、卷积核指明、滑动步长）；3、激活层设置（激活函数）；4、池化运算设置。代码如下。

```
W = tf.Variable(tf.truncated_normal(shape=[3, 3, 32, 64], dtype=tf.float32, stddev=1e-2))
```

```
conv = tf.nn.conv2d(pool1, W, strides=[1, 1, 1, 1], padding="SAME")
bn = tf.layers.batch_normalization(conv2, training=train_sign)
relu = tf.nn.relu(bn2)
pool = tf.nn.max_pool(relu2, strides=[1, 2, 2, 1], ksize=[1, 3, 3, 1],
padding="VALID")
```

# 五、实验结论与分析

## 5.1 BP 神经网络结果与分析

本次实验采用三层反馈神经网络。实验结果如表 1 所示。可见三层反馈神经网络对于十分类识别还是太勉强了，最高的准确率也只有近 25% 。但是 BPNN 是有效果的，大于 10% 就是比蒙好。而且有意思的是，BP 网络随着隐藏层神经元个数的增加，性能是好后坏的。这应该是对应着的过多的神经元会造成对训练集过拟合而对测试集表现不佳的情况。在这种超出网络能力的情况下学习率是越小越好的。

表 1 BP 神经网络实验数据

| 输入层神经元个数 | 隐藏层神经元个数 | 输出层神经元个数 | 学习率 | 准确率 |
|---|---|---|---|---|
| 3072 | 175 | 10 | 0.6 | 0.1054 |
| 3072 | | 10 | 0.3 | 0.1472 |
| 3072 | | 10 | 0.2 | 0.1610 |
| 3072 | | 10 | 0.1 | 0.2404 |
| 3072 | 300 | 10 | 0.1 | 0.2468 |
| 3072 | 400 | 10 | | 0.2262 |
| 3072 | 600 | 10 | | 0.2358 |
| 3072 | 1000 | 10 | | 0.2220 |

## 5.2 卷积神经网络结果与分析

本次实验所使用的卷积神经网络规模及策略：分批次每次训练 100 张图片；网络结构：卷积层 1+卷积层 2+卷积层 3+全连接层 1+全连接层 2；各个卷积层都含有池化层和激活层，采用最大值池化和 ReLU 激活函数；卷积层 1 用 32 个 3×3×3 卷积核，卷积层 2 用 64 个 3×3×32 卷积核，卷积层 3 用 128 个 3×3×64 卷积核；损失函数用交叉熵损失；参数优化用 tensorflow 的 AdamOptimizer 优化器。

实验结果如图 11 所示。对于 CIFAR-10 数据集，该卷积神经网络的识别准确率为 64.44%，相比 BP 神经网络有了巨大的改善。从每个批次（10000 张）训练过程来看，训练准确率提升至 50%还是比较快的，第一个批次中就有出现。然而至 60%以上的准确率就比较缓慢。也就是说卷积神经网络的训练效果(正确率)是斜率减小的上升曲线，很有可能稳定收敛于一个值，那么这个收敛值距离 100%的距离就确定了该神经网络在理论上对该问题是否能到达完美的表现。

增加卷积核的数目可以提高网络的准确率，总结如表 2 所示。对于更大规模的网络我的电脑就跑不动了。

卷积神经网络在图像识别上有巨大的优势主要是归功于卷积有能增大神经元视野等功能，和卷积神经网络的性质契合了图像的多层次结构、特征局部性、平移不变性三个特性。除此之外，还有一个有趣的辅助功能是遗忘率，模仿人类记忆的遗忘，一次迭代中是全连接层中一定比例的神经元失去参数更新的能力，能防止网络对训练集过拟合，从而具备更好的广泛性。

```
读取: C:/Users/Administrator/cifar-10-batches-py/data_batch_1
训练进度 1000 张, 损失值 = 2.120020, 训练准确率 = 0.24000
训练进度 2000 张, 损失值 = 2.049079, 训练准确率 = 0.28000
训练进度 3000 张, 损失值 = 1.942002, 训练准确率 = 0.33000
训练进度 4000 张, 损失值 = 1.936224, 训练准确率 = 0.40000
训练进度 5000 张, 损失值 = 1.773005, 训练准确率 = 0.38000
训练进度 6000 张, 损失值 = 1.786416, 训练准确率 = 0.35000
训练进度 7000 张, 损失值 = 1.661458, 训练准确率 = 0.40000
训练进度 8000 张, 损失值 = 1.656765, 训练准确率 = 0.40000
训练进度 9000 张, 损失值 = 1.487873, 训练准确率 = 0.52000
训练进度 10000 张, 损失值 = 1.544215, 训练准确率 = 0.45000
读取: C:/Users/Administrator/cifar-10-batches-py/data_batch_2
训练进度 11000 张, 损失值 = 1.617430, 训练准确率 = 0.42000
训练进度 12000 张, 损失值 = 1.607819, 训练准确率 = 0.43000
训练进度 13000 张, 损失值 = 1.673723, 训练准确率 = 0.44000
训练进度 14000 张, 损失值 = 1.680256, 训练准确率 = 0.42000
训练进度 15000 张, 损失值 = 1.520309, 训练准确率 = 0.41000
训练进度 33000 张, 损失值 = 1.747602, 训练准确率 = 0.42000
训练进度 34000 张, 损失值 = 1.324086, 训练准确率 = 0.60000
训练进度 35000 张, 损失值 = 0.996730, 训练准确率 = 0.61000
训练进度 36000 张, 损失值 = 1.329111, 训练准确率 = 0.54000
训练进度 37000 张, 损失值 = 1.363611, 训练准确率 = 0.47000
训练进度 38000 张, 损失值 = 1.374782, 训练准确率 = 0.55000
训练进度 39000 张, 损失值 = 1.303843, 训练准确率 = 0.57000
训练进度 40000 张, 损失值 = 1.205213, 训练准确率 = 0.53000
读取: C:/Users/Administrator/cifar-10-batches-py/data_batch_5
训练进度 41000 张, 损失值 = 1.101252, 训练准确率 = 0.56000
训练进度 42000 张, 损失值 = 1.035850, 训练准确率 = 0.64000
训练进度 43000 张, 损失值 = 1.066236, 训练准确率 = 0.62000
训练进度 44000 张, 损失值 = 0.923987, 训练准确率 = 0.64000
训练进度 45000 张, 损失值 = 1.099420, 训练准确率 = 0.58000
训练进度 46000 张, 损失值 = 1.027712, 训练准确率 = 0.61000
训练进度 47000 张, 损失值 = 0.831468, 训练准确率 = 0.70000
训练进度 48000 张, 损失值 = 1.120438, 训练准确率 = 0.63000
```

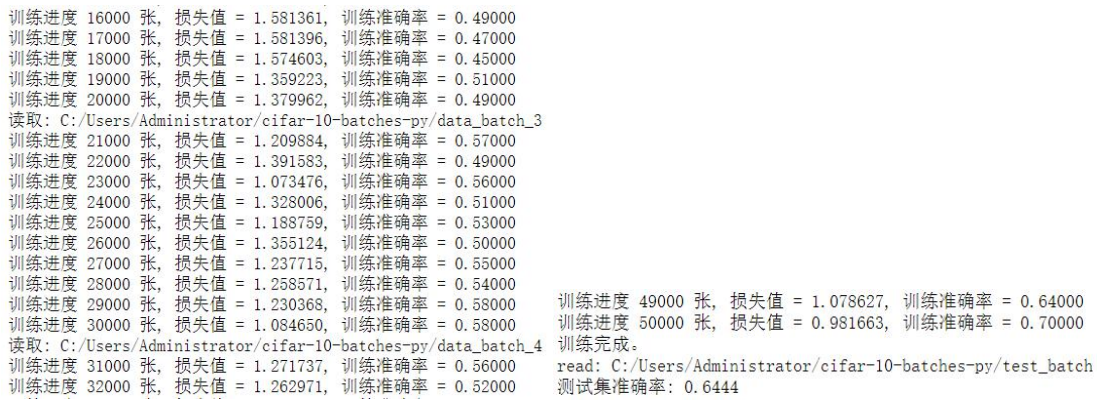


图 11　卷积神经网络运行结果

表 2 不同卷积核数目对比

| 准确率 | 卷积层 1<br>卷积核数目 | 卷积层 2<br>卷积核数目 | 卷积层 3<br>卷积核数目 |
|---|---|---|---|
| 0.5579 | 8 | 16 | 32 |
| 0.6160 | 16 | 32 | 64 |
| 0.6444 | 32 | 64 | 128 |
| 0.6547 | 64 | 128 | 256 |

## 六、总结与心得

本次实验介绍了 CIFAR-10 数据集、BP 神经网络和卷积神经网络的原理；运用网络对 CIFAR-10 数据集训练、测试；分析了 BP 神经网络和卷积神经网络在 CIFAR-10 数据集上的表现效果。根据实验结果，卷积神经网络在图像识别上的表现比 BP 神经网络优越得多。

卷积神经网络虽然在数学、神经学等理论根据上肯定难得不行，但是在形式上并不难理解。若想编程实现，有 BP 神经网络的编程基础和对卷积神经网络形式有清楚的理解，应该就可以实现。但是由于大量繁琐的细节，编程耗时会很长。Tensorflow 是一款很好的学习机器学习（例如神经网络）的工具，使用它可以极

大地简化卷积神经网络的编程实现。

本次课程作业使我收获很大。我正好可以通过这个契机熟悉一下 python 的使用。之前只看过 python 基础的书籍，对这门语言的理解只限于纸上。也因此我选择了相对简单的 BP 神经网络作为切入点，在熟练之后试着实现更复杂的卷积神经网络。而且在理论知识的学习上，BP 神经网络也能作为理解 CNN 的基础。

在编程实现中我发现自己对于 BPNN 的几个细节还不是很清楚，在加上初次使用 python 带来的额外麻烦，导致我开始时用 BPNN 实现识别图像的效率低。但实现之后对于知识的理解确实更加深刻、对 python 也更加熟悉。以此为基础，阅读了关于卷积神经网络的书籍，理解了 CNN 的原理。很开心的觉得 CNN 的形式并不复杂，数学推理也能看得懂。总之很开心，确实学到了知识。

# 七、参考文献

1、《卷积神经网络的 Python 实现》 著者：单建华著；出版日期：2019 ；出版社：人民邮电出版社。

2、《TensorFlow 与卷积神经网络从算法入门到项目实战》 著者：华超编著；出版日期：2019；出版社：电子工业出版社。

## 附：文件说明

1、课程报告：Y30190753+王志强+课程报告.pdf

2、BP 神经网络于 cifar-10 程序：cifar_10-BPNN.py

3、卷积神经网络于 cifar-10 程序：卷积神经网络_cifar-10.py

4、对原 kaggle 题目的 BP 神经网络分类程序：车分类识别-BPNN.py

5、用于读取 cifar-10 的函数：cifar_reader.py

# Python 代码附录

## 1、 BP 神经网络对 kaggle 选题

```
#  读取图像所需
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os
import numpy
import scipy.special
import csv
import matplotlib.pyplot
%matplotlib inline
# BP 神经网络类
class neuralNetwork:
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        # 权重矩阵，wih 和 who
        #从节点 i 至下一层节点 j 的权重为 w_i_j
        self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))
        self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.onodes, self.hnodes))
        # 学习率
        self.lr = learningrate
        # sigmoid 作为激活函数
        self.activation_function = lambda x: scipy.special.expit(x)
        pass
    # 神经网络的训练
    def train(self, inputs_list, targets_list):
        # 输入行向量特征，处理为列向量
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T
        # 计算输入隐藏层的信号
        hidden_inputs = numpy.dot(self.wih, inputs)
        # 计算隐藏层输出信号
        hidden_outputs = self.activation_function(hidden_inputs)
        # 计算输出层输入信号
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # 计算最终输出信号
```

```
            final_outputs = self.activation_function(final_inputs)
            # 输出层误差为 (target - actual)
            output_errors = targets - final_outputs
            # 计算隐藏层输出误差
            hidden_errors = numpy.dot(self.who.T, output_errors)
            # 更新隐藏层至输出层的权重
            self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)),
numpy.transpose(hidden_outputs))
            # 更新输入层至隐藏层的权重
            self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)),
numpy.transpose(inputs))
            pass
        # 运行神经网络
        def query(self, inputs_list):
            inputs = numpy.array(inputs_list, ndmin=2).T
            hidden_inputs = numpy.dot(self.wih, inputs)
            hidden_outputs = self.activation_function(hidden_inputs)
            final_inputs = numpy.dot(self.who, hidden_outputs)
            final_outputs = self.activation_function(final_inputs)
            return final_outputs
# main
# 读图像文件函数
def load_Img(imgDir,imgFoldName):
    imgs = os.listdir(imgDir+imgFoldName)
    imgNum = len(imgs)          #文件夹中图片数量
    img0 = Image.open(imgDir+imgFoldName+"/"+imgs[0])
    arr0 = np.array(img0)
    [a1,a2,a3]=arr0.shape
    data = np.empty((imgNum,a1,a2,a3))
    for i in range (imgNum):
        img = Image.open(imgDir+imgFoldName+"/"+imgs[i])
        arr = np.array(img)
        data[i,:,:,:] = arr     # 依次读取图片转换成像素图
      return data
      # 读图像
      craterDir = "C:/Users/Administrator/Kaggle/选取的训练集/"
      foldName = "bicycle"
      d_bicycle=load_Img(craterDir,foldName)
      craterDir = "C:/Users/Administrator/Kaggle/选取的训练集/"
      foldName = "car"
      d_car=load_Img(craterDir,foldName)
      craterDir = "C:/Users/Administrator/Kaggle/选取的训练集/"
      foldName = "motorbike"
      d_motorbike=load_Img(craterDir,foldName)
```

```
# 将三维数据矩阵拉成列向量
for i in range(len(d_bicycle)):
    t1=d_bicycle[i].reshape(375*500*3,)
    t1=t1.reshape(-1,1)
    if i==0:
        da_bicycle=t1
    if i!=0:
        da_bicycle=np.append(da_bicycle,t1,axis=1)
# 将三维数据矩阵拉成列向量
for i in range(len(d_motorbike)):
    t1=d_motorbike[i].reshape(375*500*3,)
    t1=t1.reshape(-1,1)
    if i==0:
        da_motorbike=t1
    if i!=0:
        da_motorbike=np.append(da_motorbike,t1,axis=1)
da_car=da_car.T;da_motorbike=da_motorbike.T
# number of input, hidden and output nodes
input_nodes = 562500
hidden_nodes = 800
output_nodes = 3
# learning rate
learning_rate = 0.3
# create instance of neural network
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes, learning_rate)    #BP 神经网络框架
#读入 bicycle 训练
epochs = 1
for e in range(epochs):
    for record in da_bicycle:
        all_values=record
        inputs = all_values / 255.0 * 0.99 + 0.01      #所有数据归一化
        targets = numpy.zeros(output_nodes) + 0.01
        targets[0] = 0.99
        n.train(inputs, targets)
        pass
    Pass
#读入 car 训练
epochs = 1
for e in range(epochs):
    for record in da_car:
        all_values=record
        inputs = all_values / 255.0 * 0.99 + 0.01      #所有数据归一化
        targets = numpy.zeros(output_nodes) + 0.01
        targets[1] = 0.99
```

```
            n.train(inputs, targets)
            pass
        Pass
#读入 motorbike 训练
epochs = 1
for e in range(epochs):
    for record in da_motorbike:
        all_values=record
        inputs = all_values / 255.0 * 0.99 + 0.01     #所有数据归一化
        targets = numpy.zeros(output_nodes) + 0.01
        targets[2] = 0.99
        n.train(inputs, targets)
        pass
    Pass
#   load the mnist test data CSV file into a list
#   转载测试集
craterDir = "C:/Users/Administrator/Kaggle/"
foldName = "选取的测试集"
d_test=load_Img(craterDir,foldName)
for i in range(len(d_test)):
    t1=d_test[i].reshape(375*500*3,)
    t1=t1.reshape(-1,1)
    if i==0:
        da_test=t1
    if i!=0:
        da_test=np.append(da_test,t1,axis=1)
da_test=da_test.T
da_test.shape
# 测试神经网络
# 记录判别状态用数 组
scorecard = []
for record in da_test:
    all_values = record
    correct_label = que_ding_de_zhi
    inputs = all_values / 255.0 * 0.99 + 0.01
    # 询问输出层
    outputs = n.query(inputs)
    # 取最大值为神经网络判断的类别
    label = numpy.argmax(outputs)
    # 判断正确增加 1，否则增加 0
    if (label == correct_label):
        scorecard.append(1)
    else:
        scorecard.append(0)
```

```
                pass
            Pass
        # 计算正确率
        scorecard_array = numpy.asarray(scorecard)
        print ("正确率 = ", scorecard_array.sum() / scorecard_array.size)
```

## 2、 BP 神经网络对 CIFAR-10

```
from PIL import Image
import numpy as np
import os
from __future__ import print_function
from six.moves import cPickle as pickle
from scipy.misc import imread
import platform
import numpy
import scipy.special
import csv
import matplotlib.pyplot
%matplotlib inline
import cifar_10_read          # 用来装载 cifar-10
[train_data,train_label,test_data,test_label]=cifar_10_read.load_CIFAR10("C:/Users/Administrator/cifar-10-batches-py/")
# BP 神经网络类
class neuralNetwork:
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))
        self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.onodes, self.hnodes))
        self.lr = learningrate
        self.activation_function = lambda x: scipy.special.expit(x)
        pass
    def train(self, inputs_list, targets_list)
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T
        hidden_inputs = numpy.dot(self.wih, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)
        final_inputs = numpy.dot(self.who, hidden_outputs)
        final_outputs = self.activation_function(final_inputs)
        output_errors = targets - final_outputs
        hidden_errors = numpy.dot(self.who.T, output_errors)
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)),
```

```
numpy.transpose(hidden_outputs))
            self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)),
numpy.transpose(inputs))
            pass
        def query(self, inputs_list):
            inputs = numpy.array(inputs_list, ndmin=2).T
            hidden_inputs = numpy.dot(self.wih, inputs)
            hidden_outputs = self.activation_function(hidden_inputs)
            final_inputs = numpy.dot(self.who, hidden_outputs)
            final_outputs = self.activation_function(final_inputs)
            return final_outputs
# number of input, hidden and output nodes
input_nodes = 3072
hidden_nodes = 1000
output_nodes = 10
# learning rate
learning_rate = 0.1
# create instance of neural network
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes, learning_rate)    #BP 神经网络框架
da_train = numpy.loadtxt(open("C:/Users/Administrator/作业/featvector.csv","rb"), delimiter=",", skiprows=0)
da_test = numpy.loadtxt(open("C:/Users/Administrator/作业/featvector_test.csv","rb"), delimiter=",",
skiprows=0)
jishu=0
for record in da_train:
            all_values=record
            inputs = all_values / 255.0 * 0.99 + 0.01      #所有数据归一化
            targets = numpy.zeros(output_nodes) + 0.01
            targets[train_label[jishu]] = 0.99
            n.train(inputs, targets)
            jishu=jishu+1
            Pass
#  测试神经网络
#  记录判别状态用数组
jishu=0
scorecard = []
for record in da_test:
        all_values = record
        correct_label = test_label[jishu]
        inputs = all_values / 255.0 * 0.99 + 0.01
        #  询问输出层
        outputs = n.query(inputs)
        #  取最大值为神经网络判断的类别
        label = numpy.argmax(outputs)
        #  判断正确增加 1，否则增加 0
```

```
    if (label == correct_label):
        scorecard.append(1)
    else:
        scorecard.append(0)
        pass
    pass
     jishu=jishu+1
     # 计算正确率
     scorecard_array = numpy.asarray(scorecard)
     print ("正确率 = ", scorecard_array.sum() / scorecard_array.size)
```

## 3、 卷积神经网络对 CIFAR-10

```
import tensorflow.compat.v1 as tf        # 需要用 tensorflow
tf.disable_v2_behavior()
import cifar_reader                      # 读取 cifar-10 的函数
batch_size = 100          # 每次处理 100 个图片，分批处理
s = 0
show_s = 10
train_iter = 50000        # 共 50000 张训练图片
# placeholder()函数是在神经网络构建 graph 的时候在模型中的占位，此时并没有把要输入的数据传入模型，
# 它只会分配必要的内存。等建立 session，在会话中，运行模型的时候通过 feed_dict()函数向占位符喂入
数据。
input = tf.placeholder(dtype=tf.float32, shape=[None, 32, 32, 3])   # 给 input_x 占个位 ，None 个数不确定
y_lab = tf.placeholder(dtype=tf.float32, shape=[None, 10])                  # 输出分值
rember = tf.placeholder(tf.float32)        # 遗忘率
train_sign = tf.placeholder(tf.bool)
####  卷积层 1
## tf.Variable()
## tf.truncated_normal(shape, mean, stddev) :shape 表示生成张量的维度，mean 是均值，stddev 是标准差。
# 这个函数产生正太分布，均值和标准差自己设定
W1 = tf.Variable(tf.truncated_normal([3, 3, 3, 32], dtype=tf.float32, stddev=1e-2))
# 随机初始化权重，使用 64 个卷积核
'''
   tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, name=None)
   除去 name 参数用以指定该操作的 name，与方法有关的一共五个参数：
   input：指需要做卷积的输入图像，它要求是一个 Tensor，具有[batch, in_height, in_width, in_channels]这样
的 shape，
具体含义是[训练时一个 batch 的图片数量, 图片高度, 图片宽度, 图像通道数],注意这是一个 4 维的 Tensor，
要求类型为 float32 和 float64 其中之一
   filter：相当于 CNN 中的卷积核，它要求是一个 Tensor，具有[filter_height, filter_width, in_channels,
out_channels]这样的 shape，具体含义是[卷积核的高度，卷积核的宽度，图像通道数，卷积核个数]，要求
类型与参数 input 相同，有一个地方需要注意，第三维 in_channels，就是参数 input 的第四维
   strides：卷积时在图像每一维的步长，这是一个一维的向量，长度 4
```

```
   padding：string 类型的量，只能是"SAME","VALID"其中之一
   结果返回一个 Tensor，这个输出，就是我们常说的 feature map，shape 仍然是[batch, height, width, channels]这种形式。
'''
conv1 = tf.nn.conv2d(input, W1, strides=(1, 1, 1, 1), padding="VALID")
# padding="VALID"，窗口移动，数据不足时直接舍弃
# BN 批量规范化
bn1 = tf.layers.batch_normalization(conv1, training=train_sign)
# 这个函数 tf.nn.relu()的作用是计算激活函数 relu，即 max(features, 0):将大于 0 的保持不变，小于 0 的数置为 0。
relu1 = tf.nn.relu(bn1)
# 池化层，选用最大法池化
pool1 = tf.nn.max_pool(relu1, strides=[1, 2, 2, 1], padding="VALID", ksize=[1, 3, 3, 1])
'''tf.nn.max_pool(value, ksize, strides, padding, name=None)
参数是四个，和卷积很类似：
第一个参数 value：需要池化的输入，一般池化层接在卷积层后面，所以输入通常是 feature map，依然是[batch, height, width, channels]这样的 shape
第二个参数 ksize：池化窗口的大小，取一个四维向量，一般是[1, height, width, 1]，因为我们不想在 batch 和 channels 上做池化，所以这两个维度设为了 1
第三个参数 strides：和卷积类似，窗口在每一个维度上滑动的步长，一般也是[1, stride,stride, 1]
第四个参数 padding：和卷积类似，可以取'VALID' 或者'SAME'
返回一个 Tensor，类型不变，shape 仍然是[batch, height, width, channels]这种形式'''
#### 卷积层 2
W2 = tf.Variable(tf.truncated_normal(shape=[3, 3, 32, 64], dtype=tf.float32, stddev=1e-2))
conv2 = tf.nn.conv2d(pool1, W2, strides=[1, 1, 1, 1], padding="SAME")
bn2 = tf.layers.batch_normalization(conv2, training=train_sign)
relu2 = tf.nn.relu(bn2)
pool2 = tf.nn.max_pool(relu2, strides=[1, 2, 2, 1], ksize=[1, 3, 3, 1], padding="VALID")
#### 卷积层 3
W3 = tf.Variable(tf.truncated_normal(shape=[3, 3, 64, 128], dtype=tf.float32, stddev=1e-1))
conv3 = tf.nn.conv2d(pool2, W3, strides=[1, 1, 1, 1], padding="SAME")
bn3 = tf.layers.batch_normalization(conv3, training=train_sign)
relu3 = tf.nn.relu(bn3)
pool3 = tf.nn.max_pool(relu3, strides=[1, 2, 2, 1], ksize=[1, 3, 3, 1], padding="VALID")
# 全连接层
dense_tmp = tf.reshape(pool3, shape=[-1, 2*2*128])    # 有 2*2*128 个列
print(dense_tmp)
fc1 = tf.Variable(tf.truncated_normal(shape=[2*2*128, 512], stddev=0.01))
# tf.matmul() 矩阵点乘
bn_fc1 = tf.layers.batch_normalization(tf.matmul(dense_tmp, fc1), training=train_sign)
dense1 = tf.nn.relu(bn_fc1)
dropout1 = tf.nn.dropout(dense1, rember)
'''tf.nn.dropout()是 tensorflow 里面为了防止或减轻过拟合而使用的函数，它一般用在全连接层
Dropout 就是在不同的训练过程中随机扔掉一部分神经元。也就是让某个神经元的激活值以一定的概率 p，
```

```
让其停止工作，这次训练过程中不更新权值，也不参加神经网络的计算。但是它的权重得保留下来（只是暂时不更新而已），因为下次样本输入时它可能又得工作了'''
# 全连接层 2
fc2 = tf.Variable(tf.truncated_normal(shape=[512, 10], stddev=0.01))
out = tf.matmul(dropout1, fc2)
print(out)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=out, labels=y_lab))
optimizer = tf.train.AdamOptimizer(0.01).minimize(cost)
dr = cifar_reader.Cifar10DataReader(cifar_folder="C:/Users/Administrator/cifar-10-batches-py/")
# 测试网络
correct_pred = tf.equal(tf.argmax(out, 1), tf.argmax(y_lab, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
# 初始化所有的共享变量
init = tf.initialize_all_variables()
saver = tf.train.Saver()
# 开启一个训练
with tf.Session() as sess:
    sess.run(init)
    step = 1
    while s * batch_size < train_iter:
        s += 1
        batch_xs, batch_ys = dr.next_train_data(batch_size)
        # 获取批数据,计算精度，损失值
        opt, acc, loss = sess.run([optimizer, accuracy, cost],feed_dict={input: batch_xs, y_lab: batch_ys, rember:
0.6, train_sign: True})
        if s % show_s == 0:
            print ("训练进度 " + str(s*batch_size)+ " 张" + "，损失值 = " + "{:.6f}".format(loss) + "，训练准
确率 = " + "{:.5f}".format(acc))
    print ("训练完成。")
    num_examples = 10000
    d,l = dr.next_test_data(num_examples)
    print ("测试集准确率:", sess.run(accuracy, feed_dict={input: d, y_lab: l, rember: 1.0, train_sign: True}))
    saver.save(sess, "model_tmp/cifar10_demo.ckpt")
```