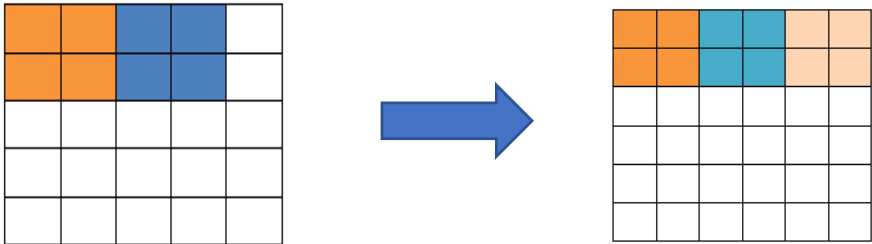
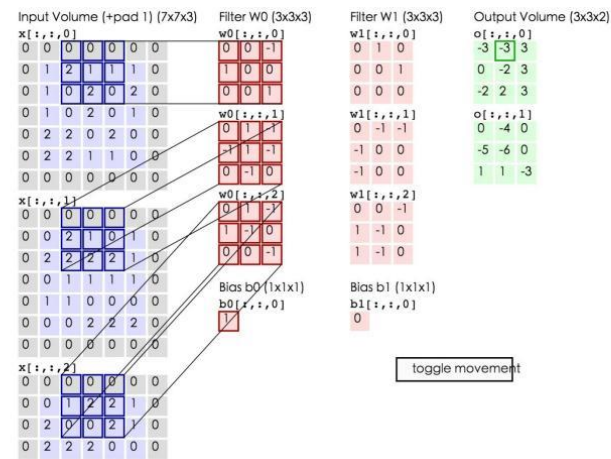


认知科学与类脑计算课程实验报告

实验题目： CNN 做手写体识别		学号： 201600181073
日期： 2019.6.6	班级： 智能 16	姓名： 唐超
实验目的： 构建一个卷积神经网络模型，实现手写数字识别。		
实验环境： Jupyter notebook & tensorflow & keras		
主要算法及步骤： <p>卷积神经网络的层级结构一般为：数据输入层、卷积计算层、激励层（最常用的为 ReLU 激活函数）、池化层（平均池化和最大池化）、全连接层（不必须）、输出层。</p> <p>数据输入层。该层要做的处理是对原始图像数据进行预处理，其中主要包括归一化操作，即将图像数据处理成均值为 0，方差为 1 的数据等。</p> <p>卷积计算层。在卷积层，有两个关键操作：局部关联，每个神经元看做一个滤波器；窗口滑动，滤波器对局部数据进行计算。先介绍卷积层遇到的几个名词：步长(stride)，即窗口一次滑动的长度；填充值(padding)，一般为 0 值填充。以下图为例，比如有一个 5*5 的图片（一个格子一个像素），我们滑动窗口取 2*2，步长取 2，那么我们发现还剩下 1 个像素没法滑完，就可以在原先的矩阵边缘加一层填充值，使其变成 6*6 的矩阵，那么窗口就可以刚好把所有像素遍历完。这就是填充值的作用。</p>  <p>卷积的计算。下图的蓝色矩阵就是输入的图像，粉色矩阵就是卷积层的神经元，这里表示了有两个神经元（w_0, w_1）。绿色矩阵就是经过卷积运算后的输出矩阵，这里的步长设置为 2。</p>		



激励层。把卷积层输出结果做非线性映射，最常用的为 ReLU 激活函数。

池化层夹在连续的卷积层中间，用于压缩数据和参数的量，减小过拟合。

程序设计&调试分析：

(1) 加载 MNIST 数据集并作预处理

下载 MNIST 数据集文件 `mnist.npz` 至本地，加载至内存，并将数据集自然划分为训练集和测试集。其中训练集和测试集的大小分别为 60000 和 10000。

```
f = np.load('mnist.npz')
x_train, y_train = f['x_train'], f['y_train']
x_test, y_test = f['x_test'], f['y_test']
f.close()
```

对手写字体图片作归一化处理：

```
x_train = x_train / 255
x_test = x_test / 255
```

对每个图片的 label 进行 one-hot 编码：

```
y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)
```

(2) 构建卷积神经网络

采用 keras 构建卷积神经网络，主要用了 3 个卷积层，每层卷积层后跟一个池化层，最后连接两个全连接层并输出，具体如下：

```
model = Sequential()
model.add(Conv2D(filters = 16, kernel_size = 2, padding = 'same', activation = 'relu', input_shape = (28, 28, 1)))
model.add(MaxPool2D(pool_size = 2))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 32, kernel_size = 2, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size = 2))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 64, kernel_size = 2, padding = 'same', activation = 'relu'))
model.add(Flatten())
model.add(Dense(500, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
```

采用交叉熵作为 loss 函数，adadelat 作为优化器。

```
model.compile(loss = 'categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
```

(3) 训练模型

在训练开始前，将数据集 reshape 为模型需要的四维张量。设置 batch_size 为 128，训练 10 个 epoch:

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
model.fit(x_train, y_train, batch_size=128, epochs = 10, verbose=1, validation_data=(x_test, y_test))
```

测试结果及分析:

训练 10 个 epoch 后，模型在训练集上准确率达到 0.9923，在测试集上准确率达到 0.9914.

```
Epoch 10/10
60000/60000 [=====] - 17s 278us/step - loss: 0.0233 - acc: 0.9923
- val_loss: 0.0251 - val_acc: 0.9914
Test score: 0.02511823370846687
Test accuracy: 0.9914
```