手写数字mnist数据集识别与云应用

**160300431 冯正阳**

1. 模型（/data&model）

我做的这个mnist数据集识别是在42000张28x28像素手写灰度图片基础上进行训练的，用相对简单的模型达到了公认的不自生成扩展数据情况下的最优识别率98%。

1.1 数据集分割和预处理

数据集按照5：1：1分割成训练集，交叉验证集和测试集，使用mnist.py中pre()函数将csv数据分割，并将标签0-9用one-hot方法重新编码成向量，然后将它们存为numpy数组到硬盘上，节约后续训练选参时的时间。

1.2 训练过程简述

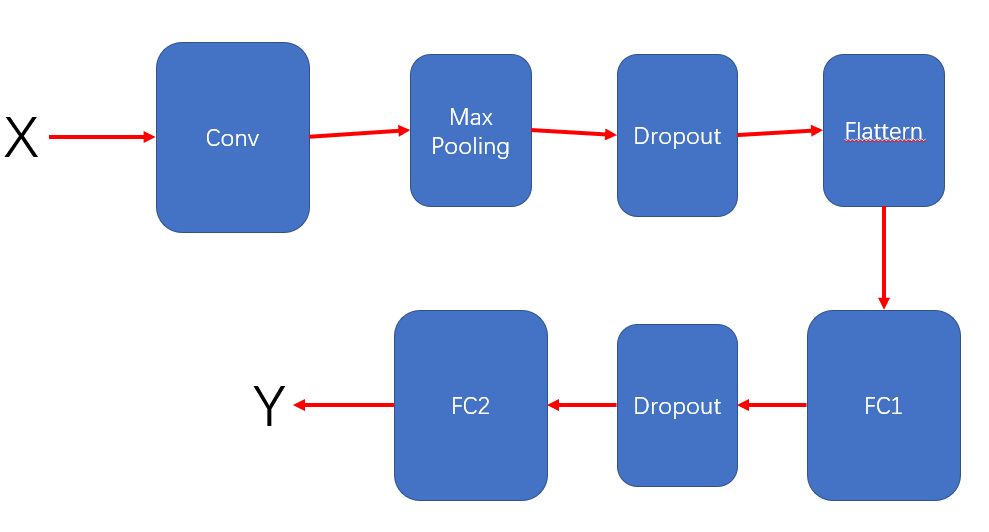
注意：训练过程均使用交互式的ipython控制台完成，mnist.py提供的是关键函数，方便重复训练，训练时没有多少代码量，我提供的train.csv即我用的42000条数据。

训练时先调用fastPre()将数据一次性读入内存，从而避免硬盘反复读写（虽然我的电脑没有出现内存不足，但若内存空间不足，可以考虑读入字符串到内存，然后在神经网络中加一层字符串解析，因为计算时一次只用几十组数据，所以只会稍微多费一点点计算时间，而内存使用可以节约很多，还不用反复硬盘读写），附加作图函数visualize()来查看个别图片具体是什么样子。

模型均使用keras实现（keras模型保存参数，可以手动训练几轮看看情况再训练），后端为tensorflow。首先对图片数据进行归一化加快训练速度（除以255），通过不断地训练和模型参数选择，最终选用的加两遍Dropout的CNN，解决了过拟合问题，经过“艰苦”的计算，最终在训练集，交叉验证集，测试集分别达到了99.2%，98.5%，98.3% 的识别准确率，/trainedModels/CNN2.h5是最终训练好的模型，参考文献中列出的链接1是kaggle上目前最好的实现，他在文中提到没有训练集扩展的情况下在kaggle测试数据下准确度只达到了98.114%， 而我在相同测试数据下达到了98.185%，而且后面那些达到99%以上的方法实际上已经是在不同程度上过拟合测试数据了。

1.3 模型介绍

网络结构简图如下：



模型细节摘要如下：

Layer Output Shape Param #

=================================================================

input (None, 784, 1) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

reshape (None, 28, 28, 1) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

convolution (None, 24, 24, 7) 182

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch normalization (None, 24, 24, 7) 28

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tanh (None, 24, 24, 7) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max pooling (None, 8, 8, 7) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (None, 448) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (None, 448) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (None, 30) 13470

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

relu (None, 30) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (None, 30) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch normalization (None, 30) 120

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (None, 10) 310

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

softmax (None, 10) 0

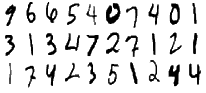
=================================================================

Total params: 14,110

Trainable params: 14,036

Non-trainable params: 74

对应mnist.py中的model2()函数，仅使用一个卷积层,外加两个全连接层，通过两次概率递增的dropout达到良好正则化效果。关键在于卷积层参数的选择，我用了7个5x5像素的特征提取器，这个选择是通过观察训练集图像得来的，

（每个数字实际为28x28像素）

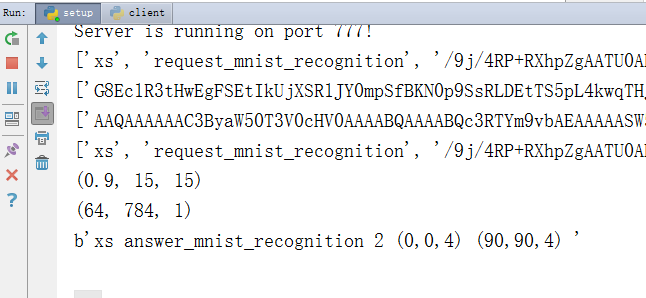
通过观察发现能区别十个数字的特征其实就是大概5x5这么大，特征形式也不算多，所以选择了7个特征提取器，不多不少。

2. 云应用（/app）

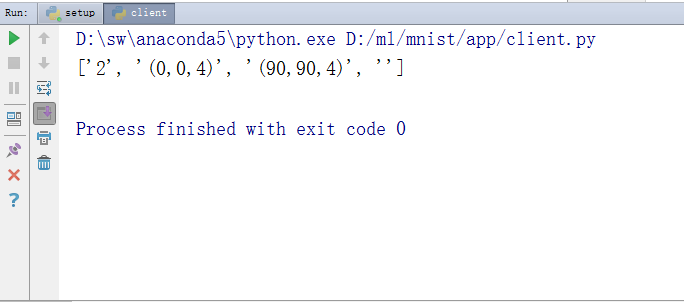
其实模型训练三月份就完成了，最近我在考虑如何应用这个模型，产生了做一个云应用的想法，具体就是在服务器上加载训练好的模型，监听777端口，接受客户根据自定义通讯协议传来的base64编码的图片，对图片内的手写数字做识别。

运行效果图如下：

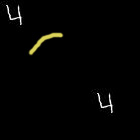
服务器：



客户端：



使用的图片：



1.1 通讯协议定义

详见目录下[api.txt](app/api.txt)

1.2 服务器端（/cloudServer）

我的服务器比较水，运算速度慢，网速慢，内存小，搞不好就被这协议弄死机了，所以综合考虑之下建议用本地测试。我在客户端接口函数里连的是127.0.0.1本地端口777，所以测试效果的话建议在本地运行setup.py开启服务器，当然了，如果有合适的服务器这个云应用就能用了（虽然就识别个手写数字，感觉没啥大用处）。

服务器端主要是按照上述定义的协议，先在内存里放好训练好的模型，然后用套接字编程的方式接收base64编码并将其进行一次格式转换（RGB->灰度），转换方法详见mnistServer.py里面的convert()函数。按照接收到的threshold（即识别确定程度达到该值才返回结果），xStride(即横向步长)，yStride(即纵向步长)，用滑动窗口法对图像进行识别，核心算法详见mnistServer.py第113-140行。最终给出每个识别到的数字所在28x28像素框的左上角坐标。

坐标图示如下：



其他都是些简单的东西，没啥值得一提的。

1.3 客户端(/SDK)

这里只是我基于上述通信协议实现的一个简单的SDK样板，mnistCloud.py即为开发包，提供了一个接口函数叫做mnistRec(图片文件名， 识别阈值， 横向跨度， 纵向跨度)。可以像sample.py里写的那样使用它。它会基于套接字编程向本地服务器127.0.0.1的777端口发送上述自定义协议的包。并将服务器的响应用python列表格式返回。

这只是一个样板，基于类似的方法或在我提供的SDK基础上可以实现诸如导入图片标记手写数字位置和识别结果展示的应用，图片数据分析应用等，而这些应用都不需要有python科学计算环境，只需要有网络，就能从服务器获取图像识别结果。

1.4 云应用目前还存在的问题

训练模型的时候很完美，但做成应用不得不说还存在很多问题，通过日后的学习我希望自己能有所提高，做出更实用的东西。

#1 mnist集没有“不是数字”这个标签，这将导致几乎所有不是数字的格子都被高概率识别成1，使得我不得不取消对数字1的识别。

#2 模型基于高度集成的keras框架，没有能够用卷积方法实现滑动窗口法（详见参考文献链接2），导致识别速度慢，最高可在步长为1时慢上千倍，导致我不得不把最低步长规定在14以上，带来了一定精度降低。

#3 健壮性低。其一表现在我自定义的通讯协议存在安全漏洞（比如不给结束符\*就会大大占用云服务器资源，但是不用结束符的话又怕接收不完图片！其他还没注意到，但估计还有），其二表现在模型本身是基于黑底白字的识别，在RGB复杂的色彩情境下压缩成灰度会导致一定程度的不精准。

#4 业界还有更好的方法。我本来做的叫“图像识别”，我用滑动窗口法把它扩展成了“目标检测”，但实际上目标检测领域公认有更好的方法叫做YOLO，然而我没有对应的数据集（详见参考文献3）。

参考文献链接：

1. <https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6>

2. [Convolution implementation of sliding windows](reference/ConvImpSlidingWindows.pdf)

3. [YOLO](reference/YOLO-originalPaper.pdf)