# 计算机视觉 Final Project

**陈铭涛**

16340024

## 1. 作业要求

输入图像为：

普通 A4 纸，上有手写的如下信息：

a. 学号

b. 手机号

c. 身份证号

所有输入格式一致，数字不粘连，但是拍照时角度可能不正。

输出：

a. 每个主要步骤的输出结果，包括对 A4 纸的矫正结果，行的切割，以及单个字符的切割结果。

b. 对 A4 纸上信息进行识别，以 xlsx 格式保存。

## 2. 程序主要流程

提交的程序包含 C++和 Python 编写的两个部分。C++程序包含了 A4 纸的矫正、A4 纸行信息的识别、数字的切割以及使用 Adaboost 对数字信息的识别。Python 部分则使用 TensorFlow 库通过卷积神经网络对数字信息进行识别，与 Adaboost 相比识别效果有较大的改进。对于每个给出的测试

数据集都根据目录结构给出了 SHELL 脚本。脚本将自动调用 Python 与 C++程序，程序执行中将所有数字图像分到识别出的对应的文件目录下，然后生成一个记录每个图片的信息的 csv 文件，最后脚本将调用一个 Python 程序将 csv 文件转为 xlsx 文件。

## 3. 程序各个主要步骤简述

a. A4 纸矫正：

A4 纸的识别与矫正部分由第七次作业的代码修改而来，主要的流程如下：

- 使用 K-means 算法根据图像像素值分为两个聚类，输出图像的高灰度聚类置为 255，地灰度聚类置为 0.
- 使用区域生长去除图像中除中心部分的纸张之外的其他纸张及亮度较高的部分。
- 遍历图像，将所有 8 邻域均为 255 的点去除，输出图像仅保留 A4 纸的边缘
- 使用霍夫变换求出 A4 纸的四条边，获取 4 个角点坐标
- 根据 4 个角点坐标计算从原图到矫正后图像的单应性矩阵，对生成图像的每个坐标值使用单应性矩阵计算原图中的坐标，使用双线性插值计算出矫正后的像素值。

程序将把对每一张图片矫正后的 a4 纸图像保存在执行目录的 a4 文件夹下。

b. 自适应二值化处理

在第 7 次作业中 Adaboost 识别图像前会先将 A4 纸图像进行全局固定

阈值二值化处理，由于当时测试数据较少，可以找出一个适合所有测试数据的阈值；但在本次作业中测试数据较多，而且很多图片有阴影区域造成影响，无法找出一个适合所有的全局阈值，因此改用了局部的自适应阈值二值化处理。其原理是对每一个像素点计算以其为中心的一定范围内的像素点平均值，若该像素点值大于平均值的一定阈值比例，则认为将该点置为 255，否则置 0，在本次因为使用了 MNIST 数据集训练，一样地将高于阈值的点置 0，其余置 255.

自适应二值化的处理结果将保存在程序执行目录下的 thresh 文件夹中。

c. 图像分割

对图像的分割方法为首先对图像沿着边缘去除灰度值为255的区域，这一步的目的是去除在 A4 矫正过程中可能将一部分背景包括在了图像边缘对图像分割所造成的影响。对处理后的图片，使用投影法进行分割，首先求出垂直方向下每一行的有效像素个数，根据获得的数据即可求出图像中数字的行数以及每行的位置。然后，对每行像素作水平方向上每一列的有效像素数求和，即可求得每行上数字的个数和位置。然而上述过程仅在理想状态下有较好的效果，测试数据中却有许多字迹歪斜或者粘连的情况，当这种情况出现时分割效果变差，因此对于分出的每一个数字还要再次尝试进行分割获取实际的单个数字，然而这种做法的缺点是最终获得的每行的数字可能顺序不准确，而且因为不能识别每行中到底有多少个数字，最终生成的结果中对应行的数字是不正确的。

图像分割的结果将以 “图片序号-行数-行中位置.jpg” 的格式 存入 segmentations 文件夹中。

d. Adaboost 进行数字识别

Adaboost 通过将多个将多个稍微好于随机的弱分类器进行组合获得一个强分类器，在作业 7 中已经进行了使用，此次仅包括了 Adaboost 的预测部分的代码，使用的数据为作业 7 中训练获得的数据。

Adaboost 获得的结果将会把识别的图像根据识别出的数字放入 AdaboostResult 文件夹中对应数字的目录，若无法识别则放入-1 目录。同时会在 AdaboostResult/xlsx 目录下对每个图片生成一个 csv 文件，包含了对应图像中每一行识别出的数据。在执行目录下还会生成一个 Adaboost-result.csv 文件，以给出的例子格式保存每张图的前三行数据。然而，在本次作业中

e. 卷积神经网络进行数字识别

在本次程序中另外使用了 TensorFlow 来训练卷积神经网络对数字进行识别。卷积神经网络的主要组成部分包括卷积层，池化层和密集层。相对于传统的 BP 神经网络，卷积神经网络同样使用反向传播的方法进行训练，但是仅有密集层为全连接。卷积神经网络中的卷积层作用为对图像中的每个子区域应用卷积核，从而输出图像特征。池化层一般跟在卷积层后，作用为降低卷积层输出的特征图维度，减少神经网络的处理时间，处理方法有取区域的最大值或平均值进行池化。

在本次代码实现中主要参考了 TensorFlow 官方教程进行 CNN 的构建，通过对 MNIST 数据集进行训练后获得的结果对 MNIST 测试集准确率达到 97.87%，但是对于作业中的数据实际的准确率目测仅有 30%左右。因此，在代码下 CNN 文件夹中还包含了从给出的图片中经过图片

分割后选出的部分数字作为训练集进行训练，以及 560 个数字作为测试集，经过训练后对 MNIST 测试集准确率为 39%，对自制的测试集准确率为 99.8%，对作业中图片数据的识别效果远好于此前使用 MNIST 数据集时的效果。

## 4. 文件结构

提交的文件根目录下中包含三个文件夹：ImageData 文件夹包含了作业第一部分的 10 张测试图片和测试结果，part2Data 文件夹包含了第二部分的各个文件夹的原图片与测试结果，project 文件夹中包含 C++与 Python 的代码文件。

AdaboostPredict 文件夹中包含了 A4 纸矫正，图像分割以及 Adaboost 进行识别的代码，使用 cmake 构建，要求系统具有 OpenCV 环境。编译指令如下：

```
#Linux, Mac
cmake .
make -j 4

#Windows (Visual Studio Command Line Prompt)
cmake -G "NMake Makefiles" .
nmake
```

C++代码中各个部分作用如下：

- utils.h 与 utils.cpp 文件包含多个类进行图像处理时都可能用到的函数
- OpenCV_utils 包含 CImg 到 OpenCV 的 Mat 的转换方法以及对测试数字图像数据的预处理
- Adaboost 文件夹中包含于 adaboost 有关的代码，执行时从执行目录

下的 mnist 文件夹中读取训练好的 Adaboost 数据。

- a4 文件夹下包含 a4 纸矫正相关的代码文件
- segmentation 文件夹下包含与数字的分割相关的代码文件。
- 主程序的使用方法如下：

  ./recognition lower higher [skipA4] [skipAdaboost]

  其中 lower 代表文件的起始序号，higher 代表最后一个文件的序号-1，程序将从执行目录下的 testcases 文件夹读取图片并输出结果至对应文件夹。若指定 skipA4 参数, 则程序直接从 a4 文件夹读取已矫正的输入，不进行矫正操作，若指定 skipAdaboost，则程序不会进行 Adaboost 识别数字的操作。

CNN 目录下包含了使用卷积神经网络的 Python 代码，各个文件作用如下：

- tf_cnn_mnist_test.py,

  tf_cnn_mnist_train.py

  以上 2 个文件分别对 MNIST 数据集进行测试与训练，训练结果保存在 mnist 文件夹下。
- tf_cnn_mySet_test.py,

  tf_cnn_mySet_train.py

  以上 2 个文件分别对自制的数据集进行测试与训练，训练结果保存在 myTrainSet 文件夹下。
- tf_cnn_mnist_predict.py

  tf_cnn_mySet_predict.py

  以上两个文件提供对传入的数字使用 MNIST 训练的模型以及自制数

据集训练的模型进行数字识别的函数，若直接运行则从命令行读取文件名列表并返回所有接受的文件名对应的图片所识别的数字。

- tf_cnn_mnist_run.py

  tf_cnn_mySet_run.py

  以上两个文件使用方法为./py lower higher，lower 和 higher 参数意义为文件序号的范围，程序将读取上面 C++程序分割完后的数字文件进行识别并将图片放入结果中对应的文件夹中，最后生成 csv 文件。

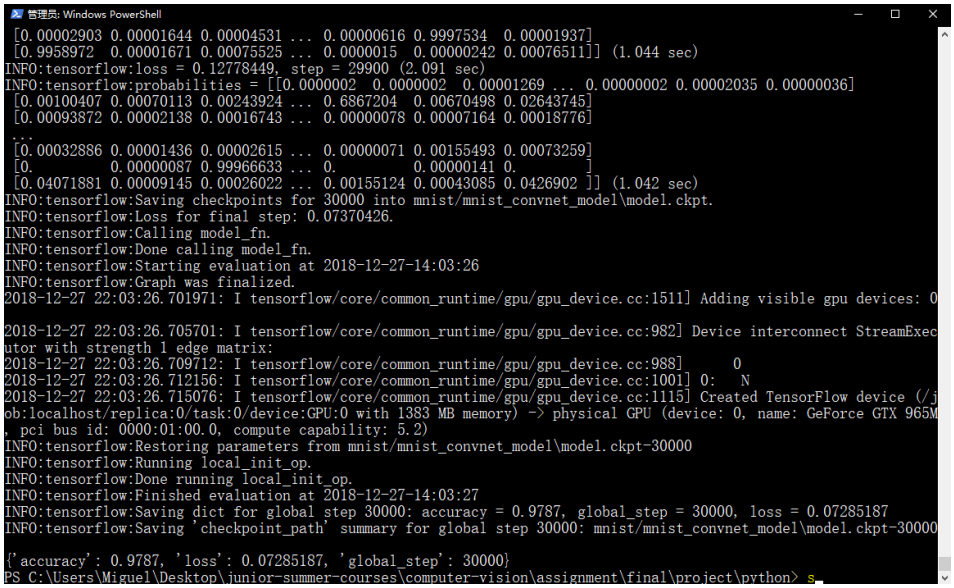
此外，在每个测试数据下都放置了 5 个 SHELL 脚本文件，作用如下：

- clean.sh: 清理运行程序所生成的所有文件
- init.sh: 运行程序前需先运行此脚本生成所有文件夹，否则运行过程中会出错。
- adaboost.sh 执行完整的 adaboost 识别数字程序，并自动将生成的 csv 转为 xlsx，需在运行参数中指定图片文件序号范围。
- CNN_mnist.sh：执行 C++代码中除 Adaboost 以外的部分，并调用 Python 使用 MNIST 数据集训练的模型进行识别输出结果，需在运行参数中指定图片文件序号范围，若已生成过 A4 矫正则可通过 skipA4 参数跳过 A4 矫正。
- CNN_mnist.sh：执行 C++代码中除 Adaboost 以外的部分，并调用 Python 使用 自制的数据集训练的模型进行识别输出结果，

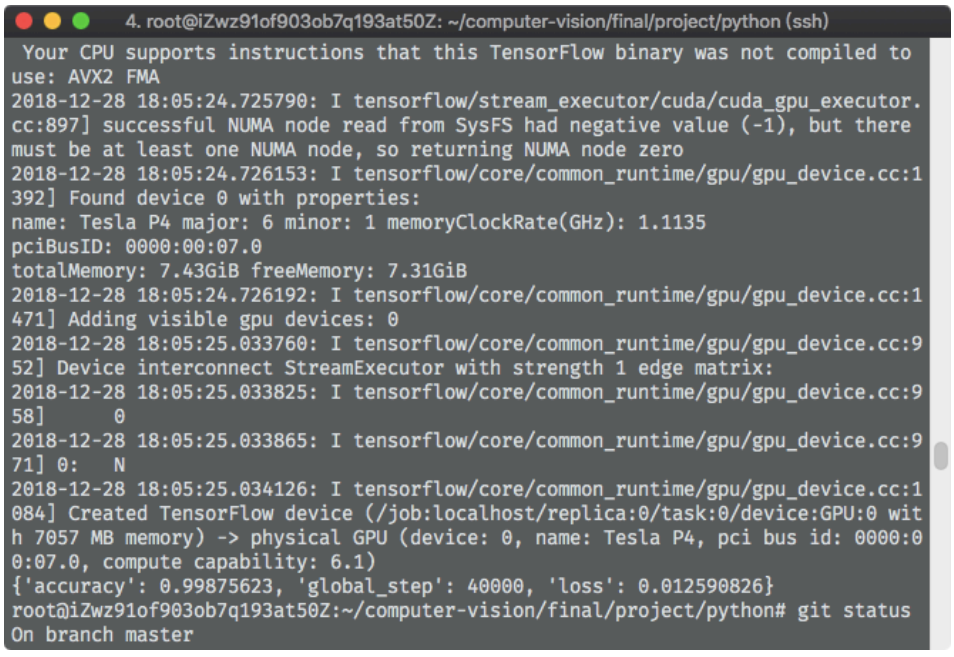
## 5. 程序运行结果

CNN 训练的结果如下：

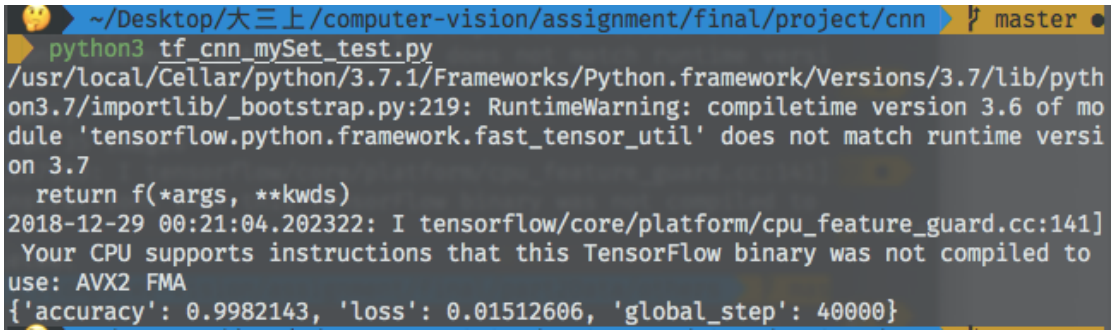
在搭载 GTX 965M 的 Windows 笔记本下进行 MNIST 数据集训练结果，准确率为 97.87%：




在阿里云 GPU 实例下进行自制的数据集的训练，对训练集自身进行测试准确率为 99.87%：



对自制的测试集进行训练，准确率为 99.82%：




对给出的 10 张图片测试获得的 xlxs 文件如下：

**Adaboost(X 代表无法识别的数字)**:

| filename | studentID | phone | citizenID |
|---|---|---|---|
| 0.jpg | 17717791 | 1X7577X7X11 | 47X57X1773XX17111777 |
| 1.jpg | 15771117 | 171X41557X3 | XXX515171177256775 |
| 2.jpg | 17X71777 | 17775115777 | 571711111517X77717 |
| 3.jpg | 17771777 | 17991115X75 | 975179177137977771 |
| 4.jpg | 15X71X71 | 1X1X7X77111 | 7713X73XX777517115 |
| 5.jpg | 75XX7105 | 1X61766X171 | 8451X117711736113X |
| 6.jpg | 51151575 | 51757757777 | XX7777777771717557 |
| 7.jpg | 17X717X1 | 17167X7771X | 1X5155117117251571 |
| 8.jpg | 15X91377 | 1X17X7XX37X70177X517X1177XX7757X7777X | 15776X11X15 |
| 9.jpg | 17771577 | 15777137777 | 7771771751567X9719 |

可以看到 Adaboost 经常出现无法识别数字的情况，而且识别到的数字准确率

非常低。

**CNN(MNIST 训练)**：

| filename | studentID | phone | citizenID |
|---|---|---|---|
| 0.jpg | 13331333 | 35211153951 | 45823713324810331318 |
| 1.jpg | 15351180 | 15282851081 | 882313177843263351 |
| 2.jpg | 15331327 | 13811818371 | 241121183612233312 |
| 3.jpg | 13331344 | 13311115818 | 533132177532277321 |
| 4.jpg | 15331367 | 13633753831 | 971281237333313835 |
| 5.jpg | 15371366 | 15617154121 | 886101171810252138 |
| 6.jpg | 13331351 | 33117315355 | 445233300051510258 |
| 7.jpg | 15351052 | 15550553783 | 550925177112221012 |
| 8.jpg | 15331348 | 14154560333201523418411884311800222l8 | 15526411418 |
| 9.jpg | 15331391 | 18733181323 | 350105174105092319 |

可看到识别准确率已高了许多，把 4 识别为 9，把 9 识别为 7，把 5 识别为 3 的

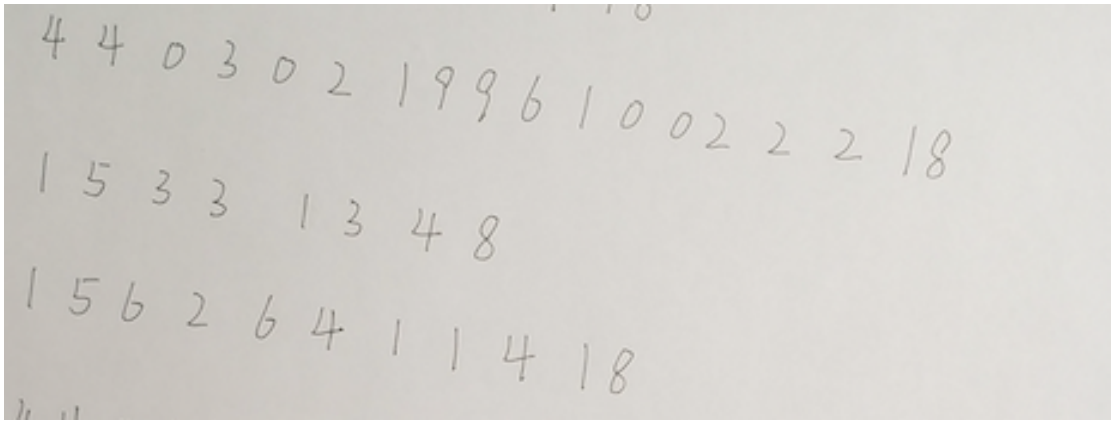
情况在错误中比较多。

**CNN（自制训练集）：**

| filename | studentID | phone | citizenID |
|---|---|---|---|
| 0.jpg | 15331333 | 93777132461 | 45675713744616331378 |
| 1.jpg | 15331180 | 13260831041 | 442313199803273059 |
| 2.jpg | 15331029 | 13827418392 | 441721199612273012 |
| 3.jpg | 15331344 | 15521145678 | 350102199602294321 |
| 4.jpg | 15331364 | 13632552831 | 441281234503093435 |
| 5.jpg | 15331646 | 13619154721 | 445301199810252734 |
| 6.jpg | 13331351 | 13719274306 | 445202200001010058 |
| 7.jpg | 15331052 | 15560563983 | 350426199712221012 |
| 8.jpg | 15331348 | 14154560333201623419411984611800222 18 | 15626411418 |
| 9.jpg | 15331347 | 18950182323 | 350106199705042314 |

可以看到使用自制的数据集训练的 CNN 可以达到较高的识别率。

另外可以看到的一个问题是在第 9 张图中，由于其第 2，3，4 行的字迹歪斜：




，从而导致程序在进行投影法分割时会将其错误地识别为一行，从而导致了错误的长输出。

识别前的各个步骤的输出图像结果也已保存在压缩包中。