Final Project

学号：17343124
姓名：伍斌
完成时间：2020.11.0

实验要求

- **每位同学根据上次课的数据集完成如下任务：**
  1. 每个同学按照例图，用干净 A4 纸写 10 张，并拍照作为测试用图. (作业 5)
  2. 校正图像为标准的 A4 图(作业 4 的代码);

  方法提示：采用**图像分割方法**或者 **Hough 变化**等.
  3. 切割字符(把所有数字切割成单个字符并输出);

  方法提示：再分割字符，最后做切割。

  要求：该步骤要求在实验报告给出每张图像的分割结果，每行的分割结果和每个字符的切割结果。
  4. 用 Adaboost 或者 SVM 训练一个手写体数字的分类器(训练数据集可以用 MNIST)，针对相关数据进行改良；(作业 5)

  要求：可以增加训练数据，以及修改测试数据的相关信息。
  5. 识别并输出每行连串数字（学号，手机号，身份证号）。
  6. 完整的实验报告(包括上面每个步骤的结果和图像)
- **编程语言：**

  PYTHON 或者 C++。

  图像操作相关 C++库：CIMG

实现过程

实验环境：

Windows 10 64 位系统、Visual Studio Code、Anaconda（已配置好 trensorflow）

实验思路：

实验分为两部分：（一）提取图像切割数字；（二）识别数字输出 excal

（一）提取图像切割数字

从预处理到 Warping 校正都是按照之前作业的步骤将一张 A4 纸单独切割出来，这一部分就直接套用了之前的代码，不多赘述。

（1）预处理（将图片转灰度图）

```
        Process p;
        CImg<float> grayImage = p.preprocess(filename);
```

```
        grayImage.display("Gray Image");
        int scale = p.getScale();
```

转灰度图:

```
CImg<float> Process::RGBtoGray(const CImg<float>& input) {
    CImg<float> grayImage = CImg<float>(input._width, input._height, 1,
 1);
    cimg_forXY(input, x, y) {
        float r = input._atXY(x,y,0,0);
        float g = input._atXY(x,y,0,1);
        float b = input._atXY(x,y,0,2);

        int newValue = (r * 0.2126 + g * 0.7152 + b * 0.0722);
        grayImage._atXY(x,y) = newValue;
    }
    return grayImage;
}
```

(2) Canny 边缘处理

```
        Canny c;
        CImg<float> canny_result = c.run(grayImage);
        canny_result.display("Canny");
```

主要接口函数:

```
CImg<float> Canny::run(CImg<float> grayImage) {
    this->filter = createFilter(gFilterx, gFiltery, sigma);
    CImg<float> gFiltered = useFilter(grayImage, this->filter);
    CImg<float> angles;
    CImg<float> sFiltered = sobel(gFiltered, angles);
    CImg<float> non = nonMaxSupp(sFiltered, angles);
    CImg<float> thres = threshold(non, threshold_min, threshold_max);
    return thres;
}
```

(3) 霍夫变换

```
        CImg<float> hough_result = h.run(canny_result, scale);
        hough_result.display("Hough");
        vector<pair<int, int> > corner = h.getCorner();
```

主要接口函数:

```
CImg<float> Hough::run(CImg<float> thres, int scale) {
    this->thres = thres;
    for (int j = 0 ; j < theta_size; ++j) {
        this->tabSin.push_back(sin(cimg::PI*j/(theta_size)));
        this->tabCos.push_back(cos(cimg::PI*j/(theta_size)));
    }
```

```
    houghLinesTransform(thres);
    houghLinesDetect();
    findEdge();
    findPoint(scale);
    return this->edgeImage;
}
```

（4）A4 纸校正

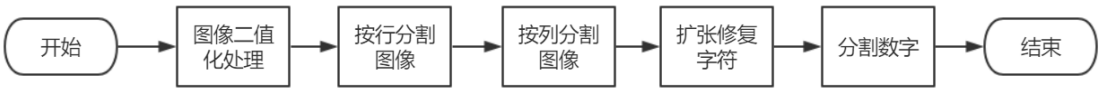
```
        ImageWarping warp;
        CImg<float> warping_result = warp.run(filename, corner);
        warping_result.display("Warping");
        string warpingpath = "./result/warping/"; // 存放校正后的图像
        warpingpath = warpingpath + filenames[i];
        warping_result.save(warpingpath.c_str());
```

主要接口函数:

```
CImg<float> ImageWarping::run(string filename, vector<pair<int,int> > c
orner) {
    this->srcImg.load(filename.c_str());
    // 读取并标定角点顺序
    this->sort_corner = SortCorner(corner);
    // 判断方向
    // 0 为Vertical，1 为Horizontal
    if (this->direction == 0)
        this->result = CImg<float>(420, 594, 1, 3, 0);
    else
        this->result = CImg<float>(594, 420, 1, 3, 0);
    // 计算投影矩阵
    this->trans_matrix = ComputeMatrix(this->sort_corner);
    // 投影
    cimg_forXY(this->result, x, y) {
        pair<int, int> point = Transform(this->trans_matrix, make_pair(
x,y));
        int u = point.first;
        int v = point.second;
        this->result._atXY(x,y,0,0) = this->srcImg._atXY(u,v,0,0);
        this->result._atXY(x,y,0,1) = this->srcImg._atXY(u,v,0,1);
        this->result._atXY(x,y,0,2) = this->srcImg._atXY(u,v,0,2);
    }
    return this->result;
}
```

（5）数字字符切割（本次项目的难点）

（参考博客 https://blog.csdn.net/qq_33000225/article/details/73123880 ）



首先将 Warping 得到的图像，也就是校正后的 A4 纸做二值化处理，然后根据垂直方向的直方图，按行分割二值化后的图像，此时每行都包含一行数字（可能有多列），然后根据水平方向的直方图，按列分割每一行，得到被行列分割的图像，此时分割的每一个区域包含一个完整的数字串，然后对每一个区域进行扩张操作从而修复断裂字符，最后，使用连通区域标记算法从左到右分割数字字符，将分割得到的字符图像及其文件名保存下来，文件名保存下来的目的是为了后续数字识别工作可以方便加载要进行识别的字符图像。

具体步骤：

图像二值化：代码采用了全局阈值分割算法，阈值为 135。全局阈值分割就是将小于设定阈值的像素设为 0，否则设为 255。

按行分割图像：利用垂直方向直方图的波峰和谷底，可以找到由黑转白或由白转黑的拐点，两个拐点的中间值就是按行分割的分割点。而由于可能出现由断裂的点造成的波峰和谷底，此时应该抛弃由此得到的分割点，因此我们可以统计子图的黑色像素个数，只有超过一定比例该子图才被认为存在完整数字，故可将其分割出来。

按列分割图像：利用水平方向直方图的波峰，计算所有波峰之间间距的均值，只有当间距大于均值的一定倍数时，才被视为要进行列分割，分割得到的每一个子图都包含一个完整的数字串。

扩张修复字符：做二值化时阈值太小容易造成字符断裂的问题，因此需要对每一张子图，进行扩张操作。首先，假设当前位置为白色像素点，检查上下 1 各单位像素和左右 2 个单位像素，统计黑色像素的总数。只有当黑色像素的总数大于 0，才把该白色像素点设为黑色。其次，假设当前位置为白色像素点，检测上下左右 1 个单位像素，若为黑色，则把该白色像素点设为黑色。

分割数字：使用连通区域标记算法从左到右分割数字字符。首先扫描图像第一列和第一行，每个黑色像素点作为一类，打上标记，然后按列进行遍历，遇到黑色像素点，检测其左下、左前、左上、正上这四个位置的像素点，再找到它们的最小类标记，将其余标记的黑色像素点以及当前位置的像素点也标记为最小类标记；如果这四个位置的像素点都不是黑色的，则当前位置作为新类，并打上新标记。最后同一类即代表是同一个数字，据此可以提取分割单个数字字符。

① ImageSegmentation 类所涉及所有函数

```
private:
    // 图像二值化处理
    CImg<float> convertToBinaryImg(CImg<float> warpingResult);
    // 做 y 方向的直方图，找到行与行之间的分割线
    void findDividingLine();
    // 通过分割线，将图片划分为一行行
    void divideIntoBarItemImg();
    //对每一张划分的图的数字，做扩张
    void dilateImg(int barItemIndex);
```

```
    // 连通区域标记算法
    void connectedRegionsTagging(int barItemIndex);
    // 存储分割后每一张数字的图以及对应的文件名称
    void saveSingleNumImg(int barItemIndex);
    // 添加新的类 tag
    void addNewTag(int x, int y, int barItemIndex);
    // 在正上、左上、左中、左下这四个邻点中找到最小的 tag
    void findMinTag(int x, int y, int &minTag, Point &minTagPointPos, i
nt barItemIndex);
    // 合并某个点(x,y)所属类别
    void mergeTagImageAndList(int x, int y, const int minTag, const Poi
nt minTagPointPos, int barItemIndex);
    // 获取单个数字的包围盒
    void getBoundingOfSingleNum(int listIndex, int& xMin, int& xMax, in
t& yMin, int& yMax);
    // 根据 X 方向直方图判断真实的拐点
    vector<int> getInflectionPosXs(const CImg<float>& XHistogramImage);
    // 获取一行行的子图的水平分割线
    vector<int> getDivideLineXofSubImage(const CImg<float>& subImg);
    // X 方向 2 个单位的负扩张，Y 方向 1 个单位的正扩张
    int getDilateXXY(const CImg<float>& Img, int x, int y);
    // XY 方向的正扩张
    int getDilateXY(const CImg<float>& Img, int x, int y);
    // 对单个数字图像做 Y 方向腐蚀操作
    CImg<float> eroseImg(CImg<float>& Img);
    // 分割行子图，得到列子图
    vector<CImg<float>> getRowItemImgSet(const CImg<float>& lineImg, ve
ctor<int> _dividePosXset);
```

② 主要接口函数

```
void ImageSegmentation::run(CImg<float> warpingResult, const string bas
eAddress) {
    if (_access(baseAddress.c_str(), 0) == -1)
        _mkdir(baseAddress.c_str());
    basePath = baseAddress + "/"; // 数字分割图片存储地址
    binaryImg = convertToBinaryImg(warpingResult);
    binaryImg.display("Binary");
    // 行分割，按照行划分数字
    findDividingLine();
    divideIntoBarItemImg();
    // histogramImg.display("Histogram");
    // dividingImg.display("Divide");
    // 对每张子图操作
    for (int i = 0; i < subImageSet.size(); i++) {
```

```
        dilateImg(i); // 对分割后每一张数字的图的数字，做扩张
        connectedRegionsTagging(i); // 连通区域标记算法
        saveSingleNumImg(i); // 存储分割后每一张数字的图以及对应的文件名称
        //cout << imglisttxt.c_str() << endl;
    }
}
```

（二）识别数字输出 excal

本部分已在 Ex5 中完成（**采用神经网络的多层感知器 MLPClassifier**），这里不贴出具体代码赘述。

首先加载 mnist 数据集，然后使用神经网络的多层感知器 MLPClassifier 进行训练，再把训练得到的模型保存下来，以用于后续数字识别工作。加载第二步得到的模型，再根据第一步得到的存储数字字符图像文件名的文件，逐一加载字符图像，然后送入预先训练好的模型进行数字识别，将单个字符预测结果存入一个字符串，再将字符串存入一个 python 列表，最后利用 pandas 的 ExcelWriter 将列表存入一个 Excel 文件。

实验结果

（这部分由于篇幅原因只选取三个例子）

【结果 1】

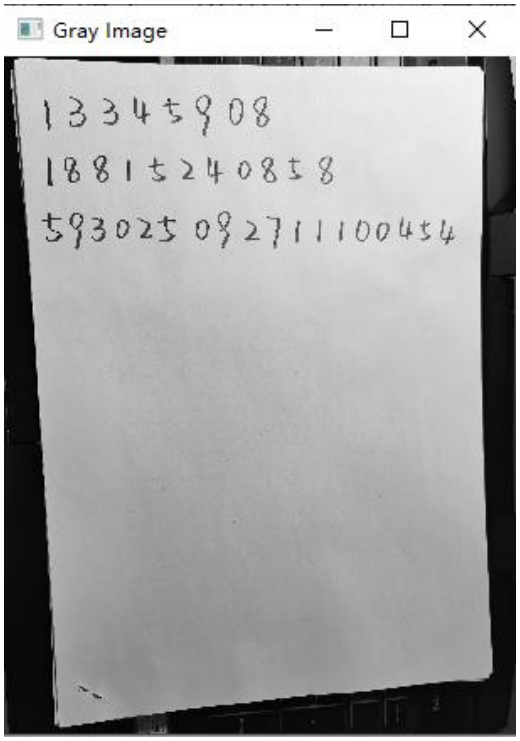
首先使用下面命令编译:

```
g++ -std=c++11 -o main.exe main.cpp Process.cpp Canny.cpp Hough.cpp ImageWarping.cpp ImageSegmentation.cpp -O2 -lgdi32
```

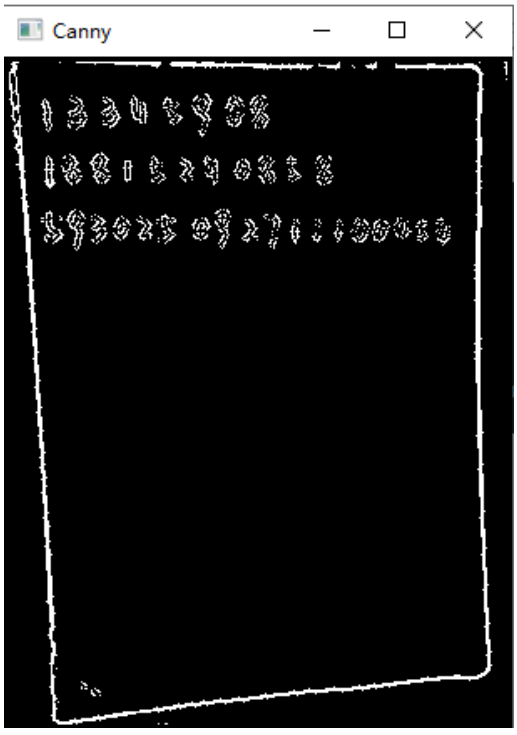
然后运行:

```
main
```

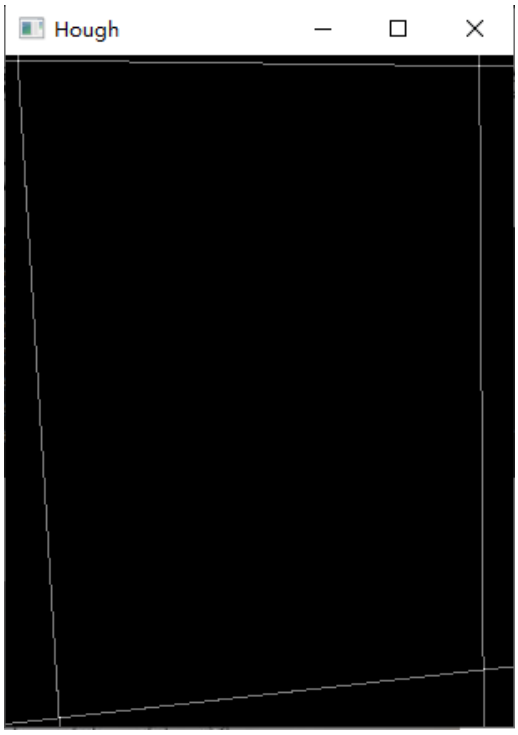
转灰度图:



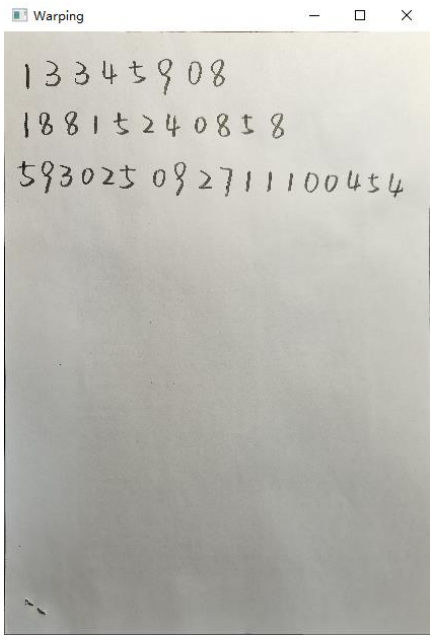
Canny 算子提取边缘:



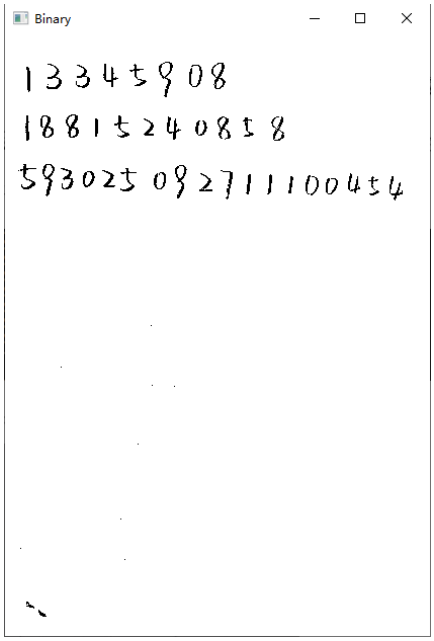
Hough 变换提取直线:



Warping 得到 A4 纸：



二值化：



输出相关数值：

```
./testdata/13345908.bmp
Gray Image: this = 00000000007df2b0, size = (304,402,1,1) [477 Kio], data = (float*)0000000000f31c90..0000000000fa920f (non-shared) = [ 10 11 10 11 11 8 13 16 ... 40 42 42 42 40 41 41 41 ], min = 0, max = 253, mean = 154.986, std = 68.0805, coords_min = (105,1,0,0), coords_max = (284,15,0,0).
Canny: this = 00000000007df2d0, size = (304,402,1,1) [477 Kio], data = (float*)00000000118eeba0..0000000011966111f (non-shared) = [ 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 ], min = 0, max = 255, mean = 10.1889, std = 49.9437, coords_min = (0,0,0,0), coords_max = (5,3,0,0).
(x, y) = (31.994, 395.788)
(x, y) = (7.29235, 3.16598)
(x, y) = (285.046, 367.046)
(x, y) = (282.781, 6.62805)
Hough: this = 00000000007df2f0, size = (304,402,1,1) [477 Kio], data = (float*)00000000011968dc0..000000000119e033f (non-shared) = [ 0 0 0 0 0 0 0 127.5 ... 0 0 0 0 0 0 0 0 ], min = 0, max = 255, mean = 1.47836, std = 13.6884, coords_min = (0,0,0,0), coords_max = (7,3,0,0).
Warping: this = 00000000007df310, size = (420,594,1,3) [2923 Kio], data = (float*)000000001f37d040..000000001f657e9f (non-shared) = [ 210 214 212 208 213 212 212 211 ... 158 158 158 158 159 159 159 159 ], min = 0, max = 255, mean = 181.618, std = 28.6062, coords_min = (0,241,0,0), coords_max = (382,5,0,0).
Binary: this = 00000000007df520, size = (420,594,1,1) [974 Kio], data = (float*)000000000119e0f90..0000000011ad49af (non-shared) = [ 255 255 255 255 255 255 255 255 ... 255 255 255 255 255 255 255 255 ], min = 0, max = 255, mean = 251.645, std = 29.0547, coords_min = (158,27,0,0), coords_max = (0,0,0,0).
```

数字字符切割结果如下：

在 Anaconda 上运行 classify.py：

```
python classify.py
```

数字识别结果如下：

```
Predicted results:
13645808
18815249858
583025082311100454
```

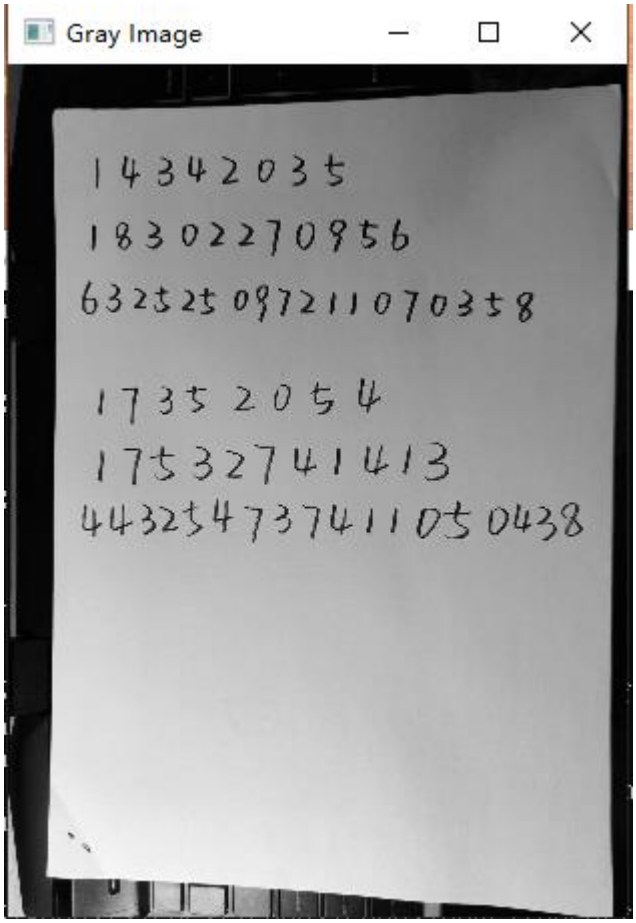
计入 Excel 表格：

13645808
18815249858
583025082311100454

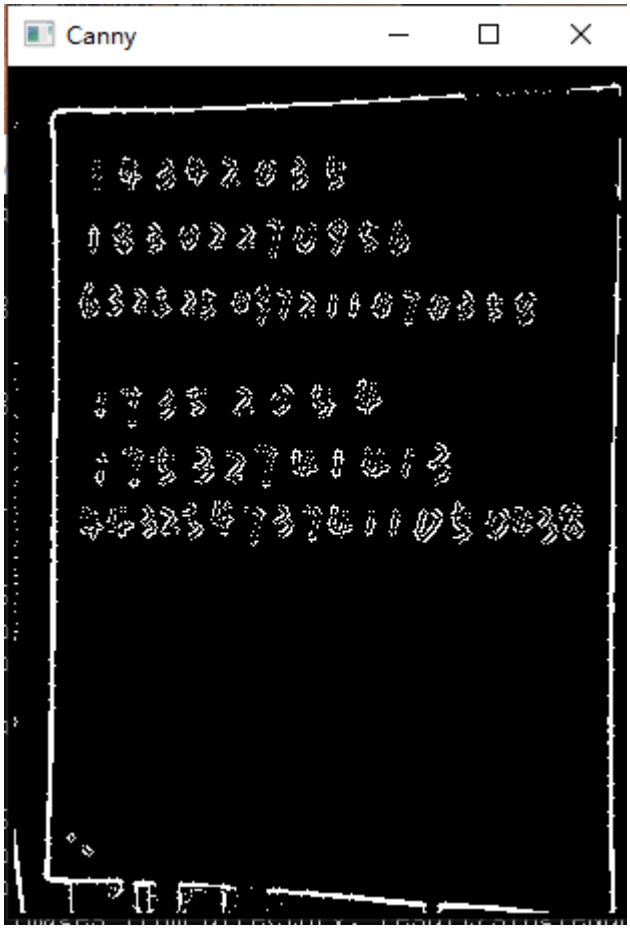
分析：本结果实现效果一般，在切割 A4 纸张并提取字符等方面表现很好，但在数字识别方面效果很差。其主要原因是：（1）训练后的识别精度依旧一般；（2）MNIST 字符集本身也有错例，如不能识别东方习惯的手写数字“9”，对西方的“9”识别就很好。如果想提高识别质量，换用更好的数据集同时采用 CNN 算法训练或许结果更好。

【结果二】失败

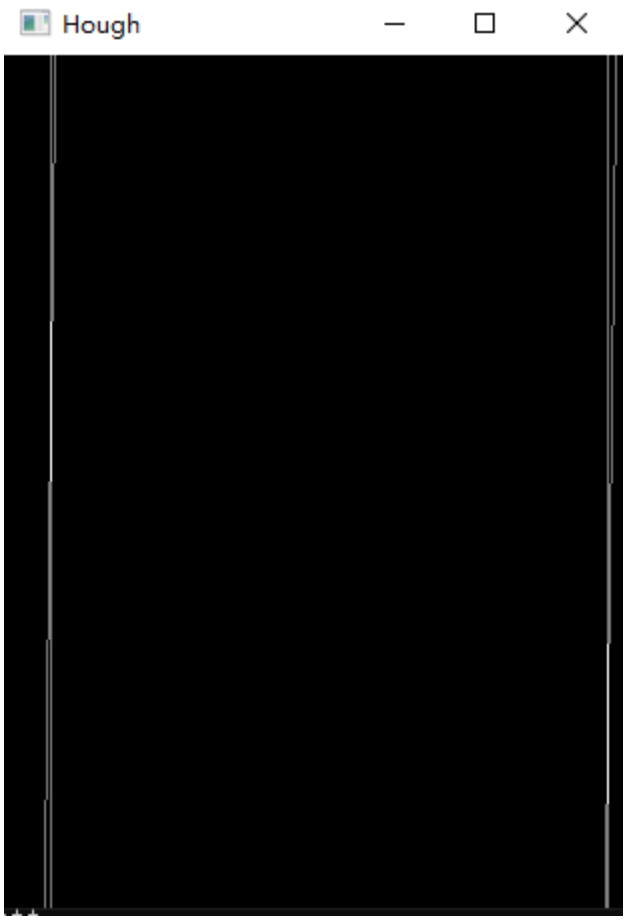
转灰度图：



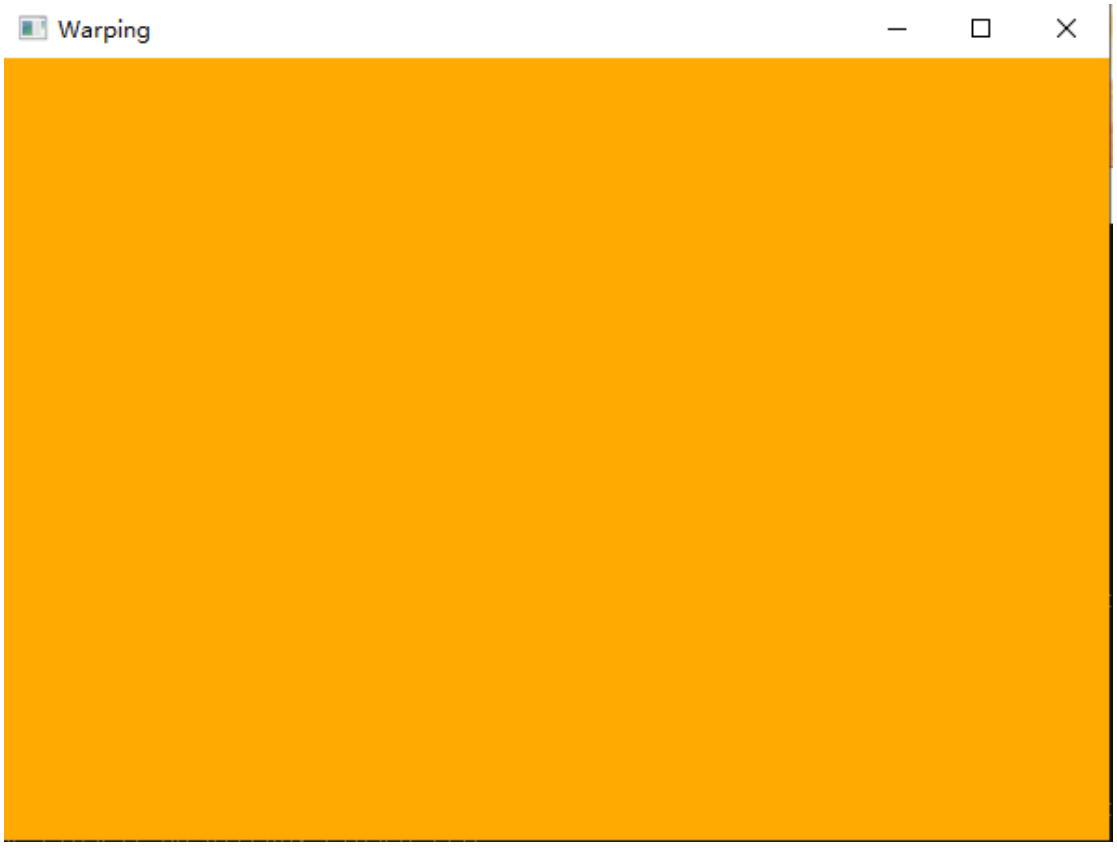
Canny 算法提取边缘：



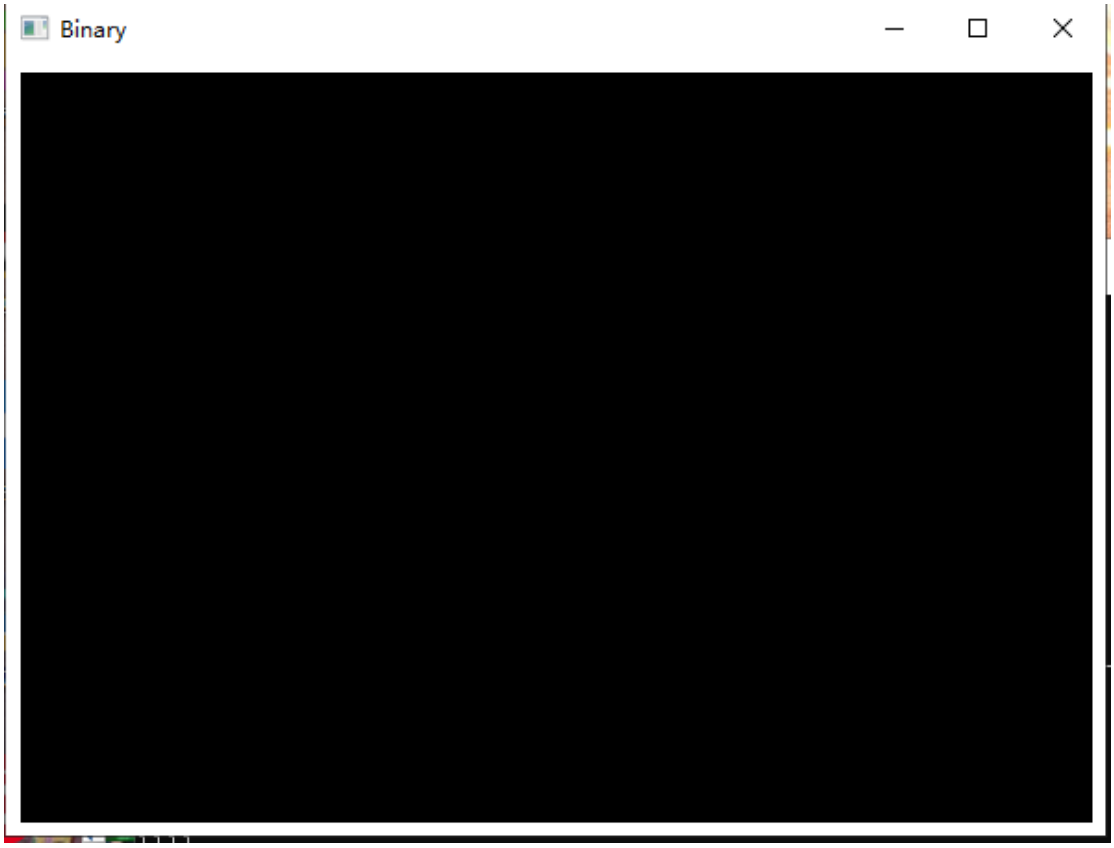
Hough 变换：



Warping（失败）

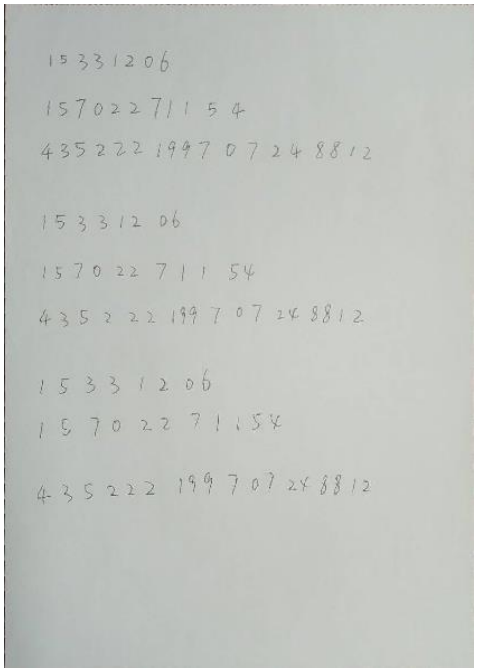


二值化：

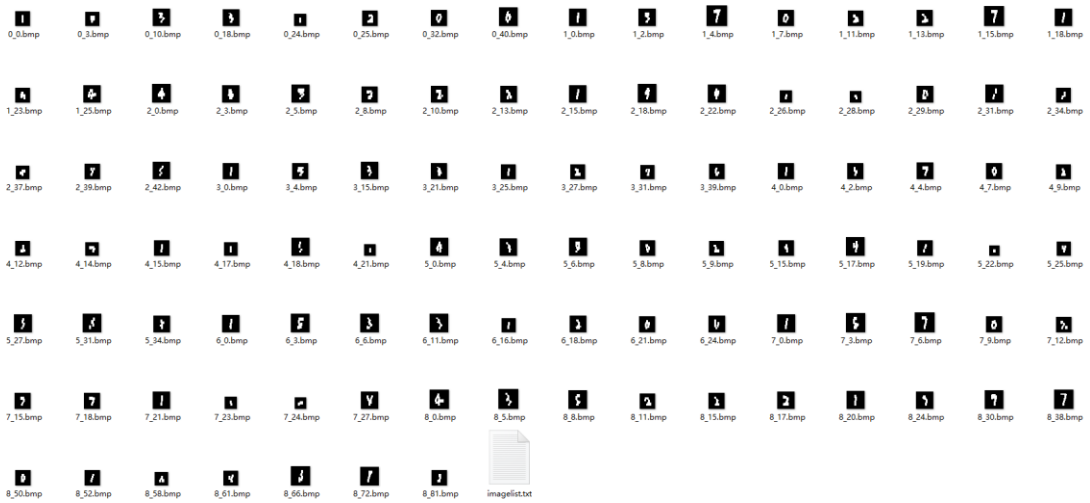


分析：本结果失败原因主要是 hough 变换未能识别出四条边，导致切割图像失败。提高拍照环境或许效果会改变。

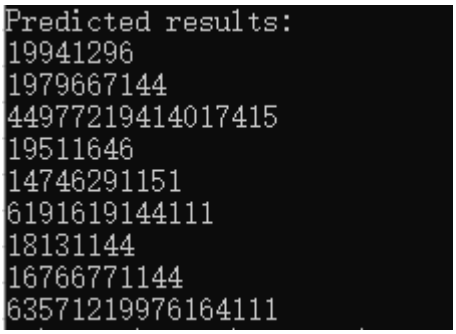
【结果三】（使用老师给的例图测试）
省去灰度图只看保存的 Warping 结果：



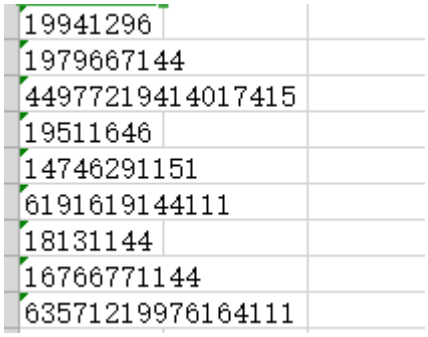
切割数字：



数字识别：



计入 Excel 表格：



分析：和结果一类似，本结果实现效果一般，在切割 A4 纸张并提取字符等方面表现很好，但在数字识别方面效果很差。其主要原因是：（1）采用 MLP 方法训练后的识别精度依旧一般；（2）MNIST 字符集本身也有错例。如果想提高识别质量，换用更好的数据集同时采用 CNN 算法训练或许结果更好。

实验总结

在读作业要求前就已知道这次作业非常有挑战性。不仅仅是将之前几次作业串起来，还有一个非常难的扩展内容：切分数字。切分数字这个思路我查阅了相当多的资料，先试了

opencv，效果很不好。最后才从学长的博客里发现了这个简单好懂的方法，但实现起来还是相当有难度的：首先是图像二值化处理，由于采用的全局阈值分割很难对所有校正后的图像做正确分割，因为容易受到拍照时亮度不均等因素的影响，导致做直方图的效果不是很好。好在还有大部分图片能够正确二值化。接下来是根据垂直方向和水平方向的直方图将图像分割成带有完整数字串的一个个区域子图，这部分的算法思想容易理解，紧接着是对断裂字符做扩张修复，最后是连通区域标记算法寻找单个数字字符并切割的过程，思想其实好懂，但实现起来比较困难。

在 Ex5 的实验（这个作业我拖了好久，直到 30 号才写出了能运行的 MLP 算法）中就有一个很大的问题，我所训练的数字分类器效果很差。原本我还是采用的 Adaboost 算法来训练，结果实操结果识别准确率接近 0%。于是我继续使用了 Ex5 的 MLP，事实证明效果还是非常一般。在查阅了更多资料后得知：在同样使用 MNIST 数据集训练的情况下，分类效果最好的是 CNN 算法，基本上准确率能达到 90%以上。可惜本学期我没有更多时间来实现这个算法。

本学期的实验在迟交了三次作业的情况下最后还是勉勉强强完成了 Final Project 的编码，这一点有点出乎意料，同时也是对我的代码能力有了一个极大的提升。再接再厉!