手写数字识别

14331098 黄建武 计应

实验环境

- Linux Deepin 15.4.1
- g++ 6.3.0
- C++11
- CImg 1.6.2
- opencv 2.4.13

参考资料

- A4纸边缘提取
- 透视变换 Perspective Transformation
- CImgList 介绍
- 基于Opencv库中SVM模块的MNIST手写字识别数据库识别

实验要求

输入图像中有一张A4纸，纸上有一个或多个手写的数字串，通过校正图像为标准的 A4 纸，用 Adaboost 或者 SVM 训练一个手写体数字的分类器，然后切割字符，识别并输出连串数字。

实验方法

- 对输入图像进行灰度图转换，并进行高斯模糊。
- 计算灰度图的梯度。
- 进行Hough变换, 求出霍夫空间的峰值点。
- 找出投票数最大的4个点即A4纸的4个角点。
- 对A4纸进行Warping转为标准普通A4纸并裁剪。
- 使用 opencv 中的 SVM 算法训练手写数字的分类器。
- 对矫正后的 A4 纸上的数字进行切割。
- 对切割生成的单个数字的图片进行缩放，缩放到 28x28大小。
- 使用训练得到的分类器进行预测。

实验实现

1. 在Ex5中已经可以求出A4纸4个角点的坐标后，并且已经实现了 A4 纸的矫正，所以在 Ex5 代码的基础上，对图片进行矫正处理，将矫正后图片保存，后面切割的环节需要用到。
2. 安装 opencv，配置环境，使用 opencv 中的 `CvSVM` 训练 Mnist 手写数据，将训练的模型结果保存到 xml 文件中，方便下次运行直接调用，跳过训练环节，节省运行时间。
3. 将矫正后的图像进行切割，先把图像二值化，然后使用扫描的方法，先按行遍历图像像素，记录每行中白色出现的次数，为了排除干扰点的影响，设置一个阈值，当次数超过阈值时，记录当前行数，作为手写数字的开始行，接着继续往下遍历，当某行的白色次数少于阈值时，记录当前行数，作为手写数字的结束

行，中间这一段图像即可作为一组手写数字，保存到文件中，然后继续往下遍历，知道图片末尾，一张图片可能存在多组手写数字。

4. 将得到的手写数字按列遍历，方法与步骤 3类似，可将一组手写数字切割成单个数字。
5. 将切割得到的单个数字进行缩放，缩放到训练集图片的大小，即 28x28，否则训练集和测试集维度不同无法进行预测。
6. 对缩放后的数字图片进行预测。

进行训练时要注意 Mnist数据的大小端问题

```
//大端存储转换为小端存储
int reverseInt(int i) {
    unsigned char c1, c2, c3, c4;

    c1 = i & 255;
    c2 = (i >> 8) & 255;//右移位  与操作
    c3 = (i >> 16) & 255;
    c4 = (i >> 24) & 255;

    return ((int)c1 << 24) + ((int)c2 << 16) + ((int)c3 << 8) + c4;
}
```

进行训练

```
void train() {
    const char* model = "mnist_svm.xml";
    ifstream file(model);
    if (file.is_open()) {
        // load the svm
        cout << "开始导入SVM文件...\n";
        svm.load(model);
        cout << "成功导入SVM文件...\n";
    }
    else {
        // 1. Set up training data
        Mat trainData;
        Mat labels;
        string trainImage = "../data/trainset/train-images.idx3-ubyte" ;
        string trainLabel = "../data/trainset/train-labels.idx1-ubyte" ;
        trainData = read_mnist_image(trainImage);
        labels = read_mnist_label(trainLabel);

        cout << trainData.rows << " " << trainData.cols << endl;
        cout << labels.rows << " " << labels.cols << endl;

        // 2. Set up the support vector machines parameters
        cv::SVMParams params;
        params.svm_type = SVM::C_SVC;
```

```
        params.kernel_type = SVM::POLY;
        params.C = 10.0;
        params.gamma = 0.01;
        params.degree= 3;
        params.term_crit = cv::TermCriteria(CV_TERMCRIT_ITER, 1000, FLT_EPSILON);

        svm.get_params();

        // 3. Train the svm
        cout << "Starting training process" << endl;
        svm.train(trainData, labels,Mat(),Mat(),params);
        cout << "Finished training process... \n";

        // 4. save the svm
        svm.save("mnist_svm.xml");
        cout << "save as mnist_svm.xml" << endl;
    }
}
```

进行预测

```
int predict(string name) {
    Mat gray = imread(name.c_str()), img, img_scaled;
    cvtColor(gray, img, CV_RGB2GRAY);
    img.convertTo(img, CV_32FC1, 1.0 / 255.0);
    resize(img, img_scaled, cv::Size(28, 28), 0, 0, CV_INTER_LINEAR);
    return svm.predict(img_scaled.reshape(1, 1));
}
```

图片行遍历切割

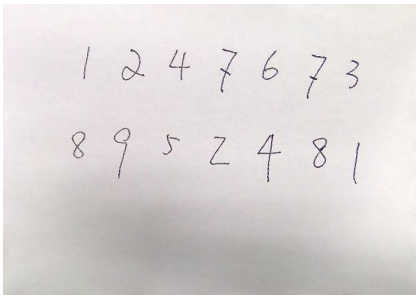
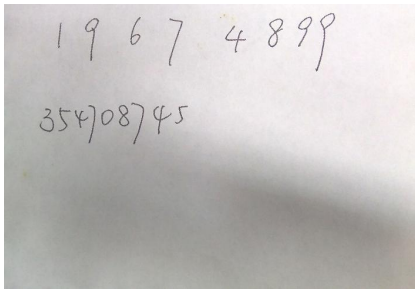
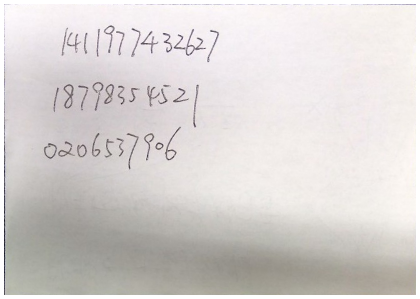
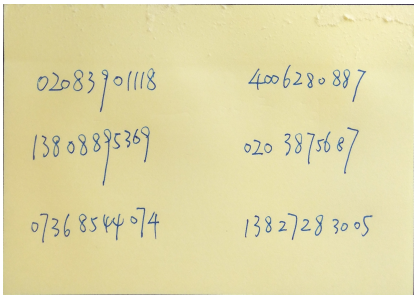
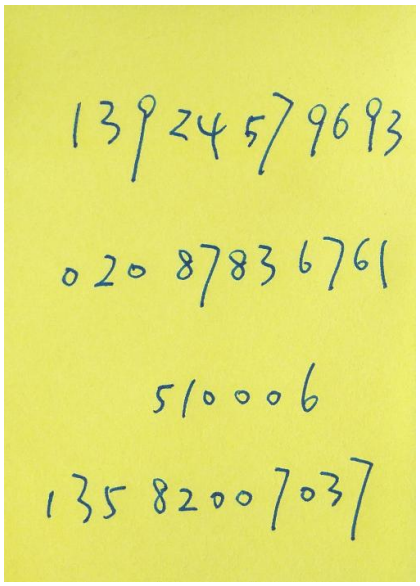
```
// 按行切割
CImgList<unsigned char> cutRow(CImg<unsigned char>& img) {
  CImgList<unsigned char> rows;
  int row_start = 0, row_end = -1;
  while (row_end < img.height()) {
    row_start = row_end + 1;
    for (int i = row_start; i < img.height(); ++i) {
      int ct = 0;
      for (int j = 0; j < img.width(); ++j)
        if (img(j, i) == 255)
          ++ct;

      if (ct > 8) {
        row_start = i;
        break;
      }
    }
```

```
    row_end = row_start;
    for (int i = row_start + 1; i < img.height(); ++i) {
      int ct = 0;
      for (int j = 0; j < img.width(); ++j)
        if (img(j, i) == 255)
          ++ct;

      if (ct < 5) {
        row_end = i;
        break;
      }
    }

    if (row_end - row_start > 30) {
      CImg<unsigned char> row = CImg<unsigned char>(img.width(),
                                                    row_end - row_start, 1, 1, 255);
      for (int i = 0; i < row.height(); ++i)
        for (int j = 0; j < row.width(); ++j)
          row(j, i) = img(j, row_start + i);

      rows.push_back(row);
    }
  }

  return rows;
}
```

完整实现请看源代码。

- 实验结果

  纠正后的图片

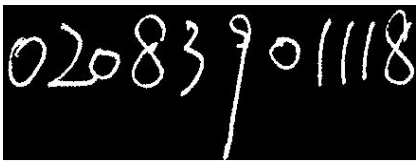
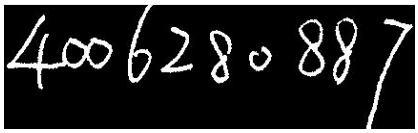
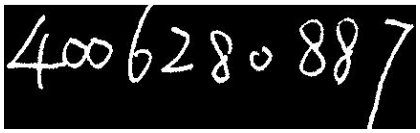
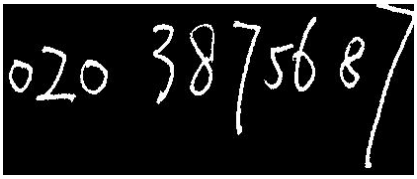






二值化后按行切割，以图片 1和图片4为例

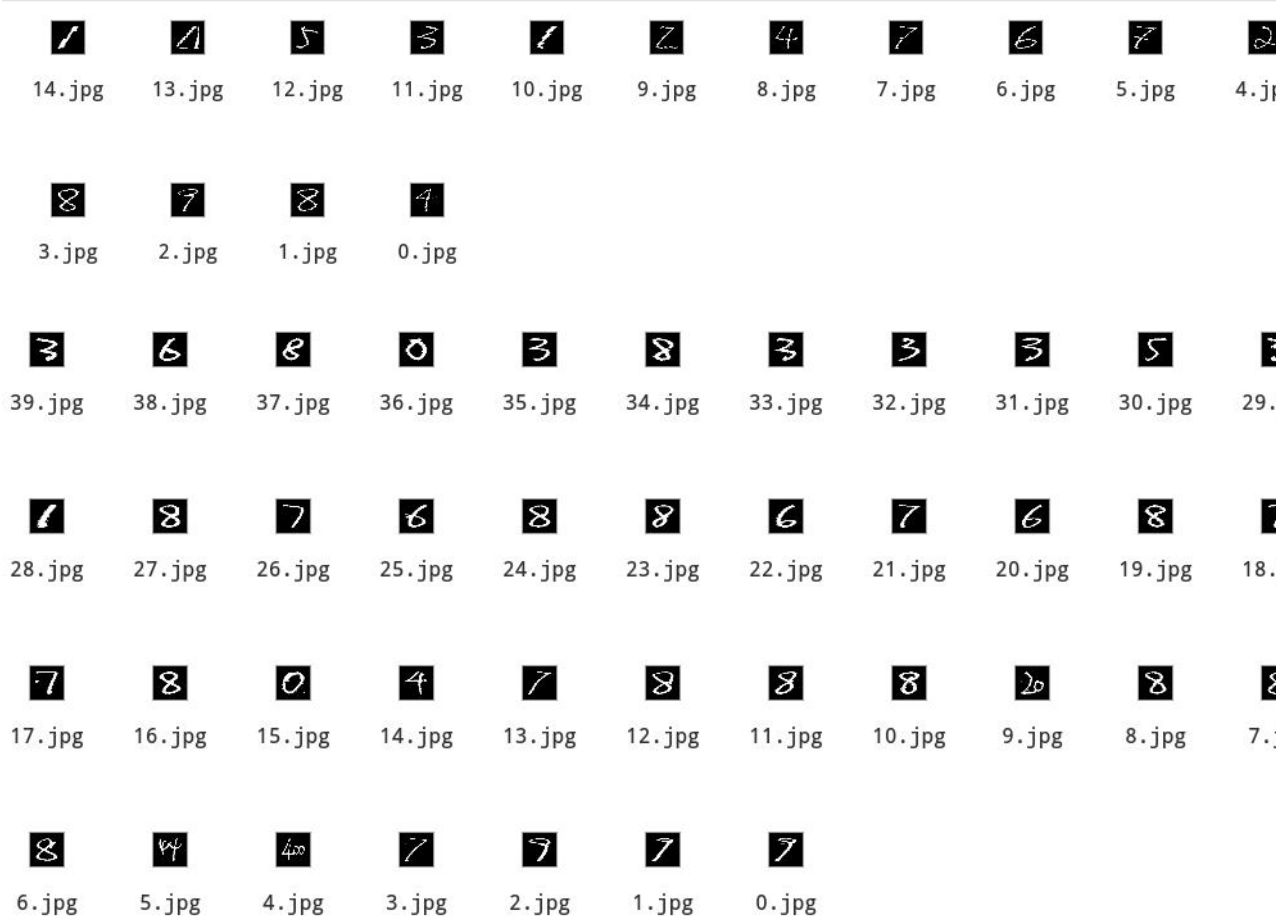




然后再将图片 4按列切割：

按列切割，以图片1和图片4为例



可以看到里面有些数字因为连笔导致切割错误

实验问题

- 读取Mnist数据的时候，一开始没有注意数据存储的格式，大小端没有注意，导致数据处理时出现错误。
- 在进行图像切割时，一开始没有设置阈值，认为出现白色就是手写数组起始行，结果有很多行有干扰点，而这些行总共也就几个白点，明显应该舍弃，设置了阈值之后，成功过滤掉不符合的行。
- 进行预测时因为没使用过 opencv，对其数据类型和方法不是很熟悉，预测的数据需要是 `CV_32FC1` 类型，而原图像是灰度图，需要进行转换，并且预测是需要将二维像素数组转为一维数组。如下所示：

```
Mat gray = imread(name.c_str()), img, img_scaled;
cvtColor(gray, img, CV_RGB2GRAY);
img.convertTo(img, CV_32FC1, 1.0 / 255.0);
resize(img, img_scaled, cv::Size(28, 28), 0, 0, CV_INTER_LINEAR);
return svm.predict(img_scaled.reshape(1, 1));
```

实验不足

A4纸矫正时所需的参数无法统一，只能对每张图片设置参数，使效果达到最优。

当手写数字有连笔时，按列分割时由于两个数字没有间隔可能会将多个数字分成一个。

分割使用的是遍历的算法，分割效率不高，且对图片的要求较高，数字不能颠倒。

对于比较模糊的数字，分类器分类会出现错误。