

# 计算机视觉||HW5

## 一、实验内容

## 二、实验算法

- 识别数字算法

这次一开始我仍想使用像素对比的方式来进行识别，但是准确率太低

所以改用SVM，训练分类器

SVM(Support Vector Machine)中文名为支持向量机，是常见的一种判别方法。在机器学习领域，是一个有监督的学习模型，通常用来进行模式识别、分类以及回归分析。

## 三、程序设计

- 图1

- 手动计算数字的位置，画框，并且截图出数字的部分得到

```
void DNum::drawNumImg() {  
  
    srcImg2.load_bmp("1.bmp");  
    srcImg2.display("0");  
    CImg<int> temp;  
    temp.resize(srcImg2.width(), srcImg2.height(), 1, 1, 0);  
    temp = srcImg2;  
    int x0[4] = {139,200,180,1430};  
    int y0[4] = {120,1400,1460,1460};  
    int x1[4] = {2080,1936,232,2000};  
    int y1[4] = {183,1427,1480,1500};  
    int color[4] = { 1,3,2,2 };  
    for (int i = 0; i < 4; i++) {  
        temp = draw_rect(temp, x0[i], y0[i], x1[i], y1[i], color[i]);  
    }  
    getnewImg(srcImg2, x0[0], y0[0], x1[0], y1[0], "num1.bmp");  
    getnewImg(srcImg2, x0[1], y0[1], x1[1], y1[1], "num2.bmp");  
    getnewImg(srcImg2, x0[2], y0[2], x1[2], y1[2], "num3.bmp");  
    getnewImg(srcImg2, x0[3], y0[3], x1[3], y1[3], "num4.bmp");  
    temp.display("drawNumImg");  
    //srcImg2.save("drawNumImg.bmp");  
}
```

- getnewImg函数和draw\_rect函数

```
CImg<int> draw_rect(CImg<double> img,int x0, int y0, int x1, int y1,int  
color) {  
    if (color == 1) {  
        img.draw_rectangle(x0, y0, x0, y1, red, 1);  
        img.draw_rectangle(x0, y1, x1, y1, red, 1);  
        img.draw_rectangle(x1, y0, x1, y1, red, 1);  
        img.draw_rectangle(x0, y0, x1, y0, red, 1);  
    }
```

```

    }
    else if (color == 2) {
        img.draw_rectangle(x0, y0, x0, y1, green, 1);
        img.draw_rectangle(x0, y1, x1, y1, green, 1);
        img.draw_rectangle(x1, y0, x1, y1, green, 1);
        img.draw_rectangle(x0, y0, x1, y0, green, 1);
    }
    else if (color == 3) {
        img.draw_rectangle(x0, y0, x0, y1, blue, 1);
        img.draw_rectangle(x0, y1, x1, y1, blue, 1);
        img.draw_rectangle(x1, y0, x1, y1, blue, 1);
        img.draw_rectangle(x0, y0, x1, y0, blue, 1);
    }
    return img;
}

void getnewImg(CImg<double> img, int x0, int y0, int x1, int y1, const
char *filename) {
    //img1.load_bmp("result1.bmp");

    CImg<double> newImg;
    int wid = x1 - x0;
    int hei = y1 - y0;
    newImg.resize(wid, hei, 1, 1, 0);

    for (int i = x0; i < x1; i++) {
        for (int j = y0; j < y1; j++) {
            newImg(i - x0, j - y0) = img(i, j);
        }
    }
    newImg.display("newImg");
    newImg.save(filename);
}

```

○ 结果

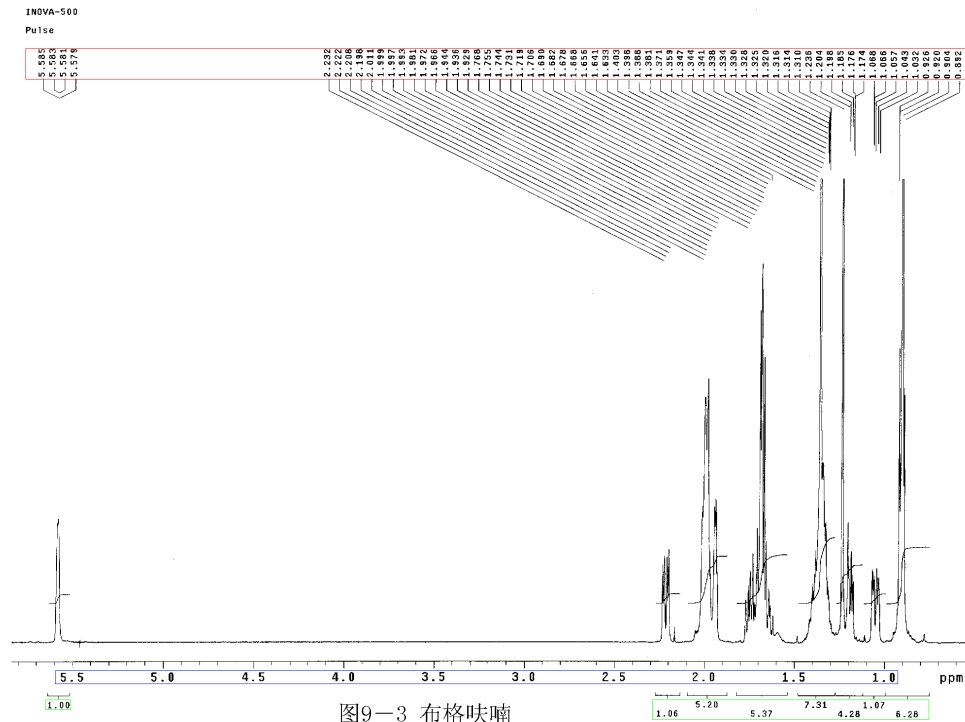


图9-3 布格呋喃

3

### ○ 识别数字

首先需要训练分类器，需要一些数字的样本，所以需要先对图片一进行提取数字，保存并分类数字，得到数据集

#### ■ 框出数字

```
Mat srcImage = imread(filename);
Mat dstImage, grayImage, Image;
srcImage.copyTo(dstImage);
cvtColor(srcImage, grayImage, COLOR_BGR2GRAY);
threshold(grayImage, Image, 48, 255, THRESH_BINARY_INV);
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(Image, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_NONE);
int i = 0;

vector<vector<Point>>::iterator It;
vector<vector<Point>>::iterator temp;
vector<vector<Point>>::iterator itor2;
Mat copyImage = Image.clone();
Rect rect[1000];
for (It = contours.begin(); It < contours.end(); It++) {

    //画出可包围数字的最小矩形
    int num = (*It).size();
    if (num > 100) {
        continue;
    }
    Point2f vertex[4];
    rect[i] = boundingRect(*It);
    vertex[0] = rect[i].tl();
    //矩阵左上角的点
    vertex[1].x = (float)rect[i].tl().x, vertex[1].y =
(float)rect[i].br().y;
    //矩阵左下方的点
```

```

        vertex[2] = rect[i].br();
        //矩阵右下角的点
        vertex[3].x = (float)rect[i].br().x, vertex[3].y =
(float)rect[i].tl().y;        //矩阵右上方的点

        for (int j = 0; j < 4; j++)
            line(dstImage, vertex[j], vertex[(j + 1) % 4],
Scalar(0, 0, 255), 1);

        con[i].x = vertex[0].x;        //根据中心点判断图像的
位置
        con[i].y = (vertex[1].y + vertex[2].y) / 2.0;
        con[i].order = i;
        i++;
    }
    //namedWindow("number", WINDOW_AUTOSIZE);
    //imshow("number", dstImage);
    //sort(con, con + i);

```

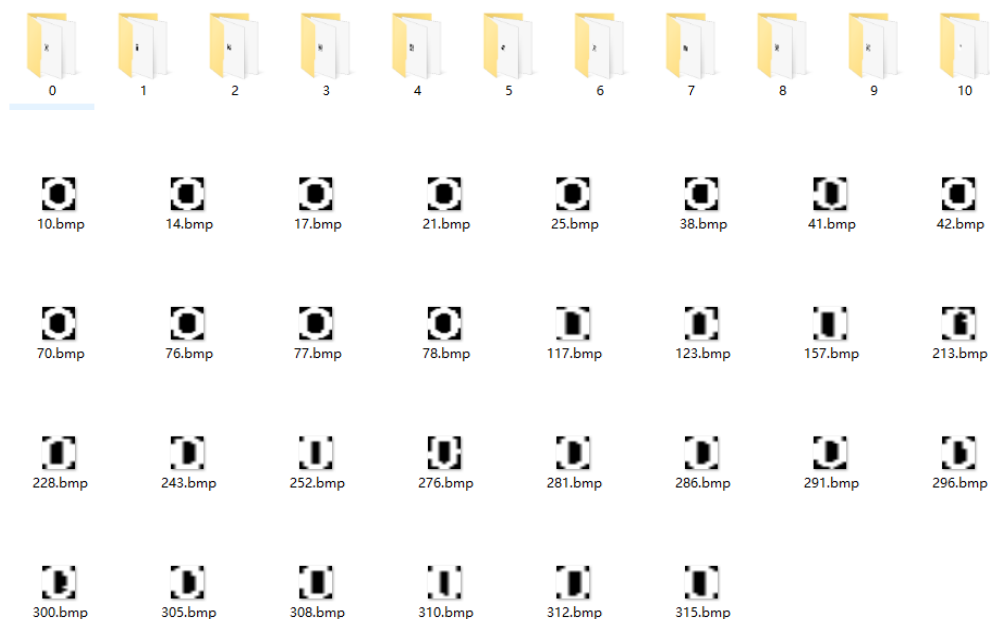
#### ■ 使用下列代码保存数字样本

```

for (int t = i - 1, name = 0; t >= 0; t--, name++)
{
    int k = con[t].order;
    Mat tempImg;
    resize(copyImage(rect[k]), tempImg, Size(30, 30));
    ostringstream fileName;
    fileName << "numberImage/" << name << ".bmp";
    imwrite(fileName.str(), tempImg);
}
//imwrite("number2.bmp", dstImage);

```

#### ■ 结果



(例子)

#### ■ 分类器训练 函数 void trainSVM();//训练分类器

```

void DNum::trainSVM() {
    //核心思路: //获取一张图片后会将图片特征写入到容器中,
    //紧接着会将标签写入另一个容器中, 这样就保证了特征
    // 和标签是一一对应的关系。

    ///=====读取训练数据
    =====///
    const int classsum = 11;//图片共有10类, 可修改
    const int imagesum = 30;//每类有张图片, 可修改
    //训了样本图片与测试图片的尺寸应该一样
    const int imageRows = 30;//图片尺寸
    const int imageCols = 30;
    //训练数据, 每一行一个训练图片
    Mat trainingData;
    //训练样本标签
    Mat labels;
    //最终的训练样本标签
    Mat clas;
    //最终的训练数据
    Mat traindata;
    //////////////////////////////////从指定文件夹下提取图片////////////////////////////////////
    for (int p = 0; p < classsum; p++)//依次提取0到9文件夹中的图片
    {
        oss << "D:/c31/计算机视觉/Hw5/hw4/hw4/numberImage/";
        num += 1;//num从0到9

        int label = num;
        oss << num << "/*.bmp";//图片名字后缀, oss可以结合数字与字符串
        string pattern = oss.str();//oss.str()输出oss字符串, 并且赋给
pattern
        oss.str("");//每次循环后把oss字符串清空
        vector<Mat> input_images;
        vector<String> input_images_name;
        glob(pattern, input_images_name, false);
        //为false时, 仅仅遍历指定文件夹内符合模式的文件, 当为true时, 会同时遍
        历指定文件夹的子文件夹
        //此时input_images_name存放符合条件的图片地址
        int all_num = input_images_name.size();
        //文件下总共有几个图片
        //cout << num << ":总共有" << all_num << "个图片待测试" <<
endl;

        for (int i = 0; i < imagesum; i++)//依次循环遍历每个文件夹中的
        图片
        {
            cvtColor(imread(input_images_name[i]), yangben_gray,
            COLOR_BGR2GRAY);//灰度变换
            threshold(yangben_gray, yangben_thresh, 0, 255,
            THRESH_OTSU);//二值化
            //循环读取每张图片并且依次放在vector<Mat> input_images内
            input_images.push_back(yangben_thresh);
            dealimage = input_images[i];

            //注意: 我们简单粗暴将整个图的所有像素作为了特征, 因为我们关注更多
            的是整个的训练过程
            //, 所以选择了最简单的方式完成特征提取工作, 除此中外,
            //特征提取的方式有很多, 比如LBP, HOG等等

```

```

        //我们利用reshape()函数完成特征提取，
        //reshape(1, 1)的结果就是原图像对应的矩阵将被拉伸成一个一行的向量，作为特征向量。
        dealimage = dealimage.reshape(1, 1); //图片序列化
        trainingData.push_back(dealimage); //序列化后的图片依次存入
        labels.push_back(label); //把每个图片对应的标签依次存入
    }
}
//图片数据和标签转变下
Mat(trainingData).copyTo(traindata); //复制
traindata.convertTo(traindata, CV_32FC1); //更改图片数据的类型，必要，不然会出错
Mat(labels).copyTo(clas); //复制

////=====创建SVM模型
=====////
// 创建分类器并设置参数
SVM_params = SVM::create();
SVM_params->setType(SVM::C_SVC); //C_SVC用于分类，C_SVR用于回归
SVM_params->setKernel(SVM::LINEAR); //LINEAR线性核函数。SIGMOID为高斯核函数

SVM_params->setDegree(0); //核函数中的参数degree, 针对多项式核函数;
SVM_params->setGamma(1); //核函数中的参数gamma, 针对多项式/RBF/SIGMOID核函数;
SVM_params->setCoef0(0); //核函数中的参数, 针对多项式/SIGMOID核函数;
SVM_params->setC(1); //SVM最优问题参数, 设置C-SVC, EPS_SVR和NU_SVR的参数;
SVM_params->setNu(0); //SVM最优问题参数, 设置NU_SVC, ONE_CLASS 和 NU_SVR的参数;
SVM_params->setP(0); //SVM最优问题参数, 设置EPS_SVR 中损失函数p的值。
//结束条件, 即训练1000次或者误差小于0.01结束
SVM_params->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER + TermCriteria::EPS, 1000, 0.01));

//训练数据和标签的结合
Ptr<TrainData> tData = TrainData::create(traindata, ROW_SAMPLE, clas);

// 训练分类器
SVM_params->train(tData); //训练

//保存模型

cout << "训练好了!!!" << endl;

}

```

#### ■ 对图1的上部数字进行数字识别

关于小数点的识别，虽然分类器可以识别小数点，当label为10的时候表示为小数点，但是因为数字的排序关系，无法按顺序输出，需要对con这个数据结构的比较函数进行修改，才能使数字的排序顺序符合图片的位置，现在还没有解决这个问题。

```

//真正检测函数
int four = 0;

```

```

for (int j = i-1; j >=0 ; j--)
{
    //调整矩阵 与数据库图片大小一致
    int k = con[j].order;
    //cout << con[j].order << " " << con[j].x << " " <<
con[j].y << endl;
    int rows1 = copyImage(rect[k]).rows;
    int cols1 = copyImage(rect[k]).cols;
    if (rows1*cols1 <= 20) {
        //cout << ".";
        continue;
    }
    Mat tempRect = Mat::zeros(db[0].size(), db[0].type());
    resize(copyImage(rect[k]), tempRect, tempRect.size());
    int rows = tempRect.rows;
    int cols = tempRect.cols*tempRect.channels();
    //cvtColor(tempRect, tempRect, COLOR_BGR2GRAY);
    threshold(tempRect, tempRect, 0, 255, THRESH_OTSU);
    tempRect = tempRect.reshape(1, 1);
    Mat inputtemp;
    inputtemp.push_back(tempRect);
    inputtemp.convertTo(inputtemp, CV_32FC1); //更改图片数据的类型,
必要, 不然会出错

    float r = SVM_params->predict(inputtemp); //对所有行进行预测
    cout << r;
    //小数点逻辑关系
    //num1

    four++;
    if (four >= 4) {
        four = 0;
        cout << endl;
    }
    else if (four == 1) {
        cout << ".";
    }
}

}

```

#### ■ 对图1的数据部分数字进行数字识别

在上部数字的识别基础上进行参数修改, 使输出更加符合图片位置, 这次有对con进行排序, 所以可以按顺序识别并输出需要的小数点, 识别部分修改如下。这里还需在numrecode中记录下数字, 方便第2步骤

```

float r = SVM_params->predict(inputtemp); //对所有行进行预测
if (r == 10) {
    cout << ".";
    tempNum += ".";
}
else {
    cout << r;
    tempNum += int(r)+'0';
}

```

```
//小数点逻辑关系
//num1
//num2
four++;
if (four >= 4) {
    four = 0;
    cout << endl;
    numrecode.push_back(tempNum);
    tempNum = "";
}
```

- 对图1的标尺数字进行数字识别  
只需要要修改换行的逻辑就可以了,其他算法同上
- 结果



训练好了!!!

5.585

5.583

5.581

5.579

2.232

2.222

2.208

2.198

2.011

1.999

1.997

1.993

1.981

1.972

1.966

1.944

1.936

1.929

1.768

1.755

1.744

1.731

1.719

1.706

1.690

1.682

1.678

1.668

1.656

1.641

1.633

1.403

1.398

1.388

1.381

1.371

1.359

1.347

1.344

1.341

1.338

1.334

1.330

1.328

1.325

1.320

1.316

1.314

1.310

1.236

1.204

1.198

1.185

1.176

1.174

1.068

1.066

1.057

1.043

1.032

0.926

0.920

0.904

0.892

请按任意键继续. . .

```
训练好了!!!  
5.5  
5.0  
4.5  
4.0  
3.5  
3.0  
2.5  
2.0  
1.5  
1.0  
请按任意键继续. . .
```

```
训练好了!!!  
1.00  
请按任意键继续. . .
```

```
D:\c31\计算机视觉\HW5\hw4\x64\Debug\hw4.exe  
训练好了!!!  
1.06  
5.20  
5.37  
7.31  
4.28  
1.07  
6.28  
请按任意键继续. . .
```

○ 步骤二

还是需要识别出标尺的起始位置和0.1距离对应的像素点

```
//找到标尺固定点  
int maxp = 0;  
int rulerx = 0;  
int rulery = 0;  
  
cimg_forY(temp, y) {  
    int point = 0;  
    int tx = 0;  
    cimg_forX(temp, x) {  
        if (temp(x, y) == 255) {  
            point++;  
            tx = x;  
        }  
    }  
    if (maxp < point) {  
        maxp = point;  
        rulery = y;  
        rulerx = tx;  
    }  
}  
cout << rulerx << " " << rulery << endl;  
int flag = 0;  
int yunx[2];  
for (int i = rulerx; i >= 0; i--) {  
    int point = 0;
```

```

        for (int j = rulery; j < rulery + 10; j++) {
            if (temp(i, j) == 255) {
                point++;
            }
        }
        if (point > 3) {
            yunx[flag] = i;
            flag++;
            i -= 3;
            if (flag >= 2) break;
        }
    }
    cout << yunx[0] << " " << yunx[1] << endl;

    int tempy = 91;
    vector<int> arrayi;
    for (int i = 0; i < temp.width(); i++) {
        if (temp(i, tempy) == 255) {
            while (true) {
                int point = 0;
                for (int j = tempy; j > tempy - 5; j--) {
                    if (temp(i, j) == 255) {
                        point++;
                    }
                }
                if (point > 3) {
                    arrayi.push_back(i);
                    i += 5;
                }
                i++;
                if (temp(i, tempy) == 0) {
                    arrayi.push_back(-1);
                    break;
                }
            }
        }
    }

    }
    int sizt = arrayi.size();
    for (int i = 0; i < sizt; i++) {
        cout << arrayi[i] << " ";
    }
    cout << endl;

```

■ 通过比例计算距离，并且输出结果

```

! [r2_1](D:\c31\计算机视觉\HW5\img\pic1\r2_1.PNG) int bili = yunx[0] -
yunx[1];
    int sizt_m = arrayi.size();
    vector<double> juli;
    for (int i = 0; i < sizt_m; i++) {
        if (arrayi[i] != -1) {
            double len = yunx[0] - arrayi[i];
            len = (len / bili)*0.1+0.6;

```

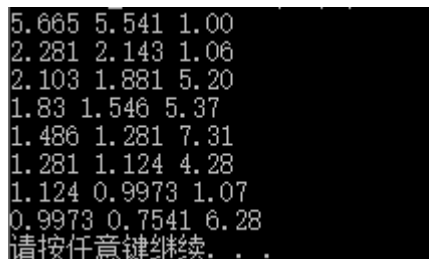
```

        juli.push_back(len);
    }
    else {
        juli.push_back(-1);
    }

}
int s = 0,e = 0;
//vector<double> num;

double numk[8] = {
    1.00, 1.06, 5.20,5.37,7.31,4.28,1.07,6.28
};
int k = 0;
for (int i = 0; i < juli.size(); i++) {
    if (juli[i] == -1) {
        e = i;
        for (int j = s; j < e-1; j++) {
            cout << numrecode[k] << " ";
            cout << setprecision(4) << juli[j] << " "<<
setprecision(4) << juli[j + 1]<< endl;
            k++;
        }
        s = i + 1;
    }
}
}

```



这里输出顺序不太对，最后才是数据

#### o 图2

这里的算法和图1相同，需要修改参数。本来一开始想不再训练分类器的，但是效果不好，所以最后还是需要重新训练分类器2。

定义函数

```

/pic2
void drawNumImg2();//划分数字块
void detectNum_2(string filename);//检测上部数字 可通用检测下部数字（数据）
void detectNum2_2(string filename);//检测标尺数字
//void detectNum3_2(string filename);//检测下部数字
void trainSVM2();//训练分类器
void drawRuler2();//标尺数字

```

结果

```
训练好了!!!  
7.42  
7.27  
7.10  
7.10  
7.09  
7.02  
7.02  
4.32  
4.32  
4.30  
4.30  
4.29  
4.28  
4.27  
2.42  
1.56  
1.32  
1.30  
1.28  
0.00  
请按任意键继续. . .
```

```
训练好了!!!  
1.00  
0.97  
2.01  
1.98  
2.99  
2.99  
3.07  
请按任意键继续. . .
```

```
训练好了!!!
7.8
7.6
7.4
7.2
7.0
6.8
6.6
6.4
6.2
6.0
5.8
5.6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
请按任意键继续. . .

coords_max = (125,5,0,0).
1.00 7.296 7.182
0.97 6.957 6.886
2.01 6.886 6.779
1.98 4.271 4.104
2.99 2.45 2.257
2.99 1.593 1.436
3.07 1.339 1.168
1: this = 0000002A3E146FA8, size = (2210,130,1,3) [3366 Kio], data = (int*)000
```

o 图3

因为图片比较模糊

一开始需要使用HW4的前后背景分离算法，使用OTSU进行前后背景分离

这样才能进行数字的识别

- 还有关于图3的数据部分的，因为上下的竖线和数字连接，所以数字识别比较困难，我先通过消除掉右边与数字无关的线条，然后再拟合直线，消除掉所有直线，这样效果比较好，但是仍然有一些小像素点，这个直接通过消去连通块 $T < 50$ 的方法去除掉

```
! [ex_num2] (D:\c31\计算机视觉\HW5\hw4\hw4\ex_num2.bmp) void
DNum::dectpoint() {
    CImg<int> temp;
```

```

temp.load_bmp("num2.bmp");
temp.display();
cimg_forXY(temp, x, y) {
    if (x >= 70) {
        temp(x, y, 0) = 255;
        temp(x, y, 1) = 255;
        temp(x, y, 2) = 255;
    }
}
vector<int> key;
cimg_forY(temp, y) {
    int point = 0;
    cimg_forX(temp, x) {
        if (temp(x, y) == 0) {
            point++;
        }
    }
    if (point > 50) {

        //point = 0;
        key.push_back(y);
    }
}
int len = key.size();
cout << len << endl;
for (int i = 0; i < key.size(); i++) {
    cimg_forX(temp, x) {
        if (temp(x, key[i]) == 0) {
            temp(x, key[i],0) = 255;
            temp(x, key[i],1) = 255;
            temp(x, key[i],2) = 255;
        }
    }
}
temp.display();
temp.save("ex_num2.bmp");
}

```

结果

**0.98**

**1.99**

**0.99**

**1.99**

**2.25**

**3.05**

**1.98**

**2.06**

**2.01**

**3.06**





**0.98**

**1.99**

**0.99**

**1.99**

**2.25**

**3.05**

**1.98**

**2.06**

**2.01**

**3.06**

- 对数字的识别都可以采用上面图片的算法，然后也需要重新训练分类器
- 对标尺的识别需要修改一些

这次2对为一组

```
void DNum::drawRuler3()
{
    CImg<int> temp;
    temp.load_bmp("num4.bmp");
    temp.display();
    vector<int> arrayi;
    int flag = 0;
    cimg_forX(temp, x) {
        if (temp(x, 10) == 0) {
            arrayi.push_back(x);
            flag++;
            x += 5;
        }
        if (flag >= 2) {
            arrayi.push_back(-1);
            flag = 0;
        }
    }
    for (int i = 0; i < arrayi.size(); i++) {
        cout << arrayi[i] << " ";
    }
    cout << endl;
    int yun[2] = { 2129, 2108 };
    int bili = yun[0] - yun[1];
    vector<double> juli;
    for (int i = 0; i < arrayi.size(); i++) {
        if (arrayi[i] != -1) {
            double len = yun[0] - arrayi[i];
            len = double(len / bili) * 0.1;
            juli.push_back(len);
        }
        else {
            juli.push_back(-1);
        }
    }
    temp.display();
    double numk[8] = {
        1.00, 1.06, 5.20, 5.37, 7.31, 4.28, 1.07, 6.28
    };
    int e = 0, s = 0;

    int k = 0;
    for (int i = 0; i < juli.size(); i++) {
        if (juli[i] == -1) {
            e = i;
            for (int j = s; j < e - 1; j++) {
                cout << numrecode3[k] << " ";
                cout << setprecision(4) << juli[j] << " " <<
                    setprecision(4) << juli[j + 1] << endl;
            }
            k++;
            s = e;
        }
    }
}
```

```

        k++;
    }
    s = i + 1;
}
temp.display("1");
}

```

## ■ 结果

■

```

8.464
7.904
7.889
7.584
7.569
7.554
7.549
7.408
7.654
7.274
7.269
5.728
7.242
6.892
6.778
6.866
6.851
4.201
5.410
4.091
4.077
6.399
3.956
3.944
2.771
2.758
2.746
2.251
1.211
9.611
请按任意键继续. . .

```

```

训练好了!!!
0.98
9.91
0.99
9.91
2.52
3.05
9.18
2.06
2.01
3.06
请按任意键继续. . .

```

```

训练好了!!!
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
请按任意键继续. . .
0214945B9DA7 (non-shared) = 1 255
mean = 231.313, std = 74.0212, co
0.98 8.757 8.686
9.91 8.195 8.114
0.99 7.852 7.786
9.91 7.757 7.69
2.52 7.538 7.462
3.05 7.152 7.038
9.18 4.281 4.205
2.06 4.138 4.057
2.01 2.9 2.824
3.06 1.295 1.238
1: this = 0000002B0D6CFE48 size =

```

## 四、实验结果

有2个问题还没有解决导致结果还不够准确

- 数字排序问题，我本来是使用数据结构

```

struct con {
    double x, y;           //轮廓位置
    int order;             //轮廓向量contours中的第几个

    bool operator<(con &m) {
        if (x > m.x) return false;

        else return true;
    }

}con[1000];

```

来进行排序的，但是这个排序要修改下才能使得识别数字的顺序正确

- 标尺问题

使用比例计算，最后的结果仍存在偏差

解决方法：使用投影的方法投影到标尺上，然后向右遍历标尺，记录到0经过几道标坎，然后小数部分使用比例进行计算

没有时间修改了，先交....

## 五、实验心得

这次实验挺难的，主要是数据的处理这个方面。

## 六、附录

图片太多交不上去

放在GitHub上了

<https://github.com/chentf5/CV>