

计算机视觉期末项目

Part 2












































16340220 王培钰 电子政务

因为在part1部分时已经交了完整的项目分析报告，所以这里只是做一个简单的两部分实验效果分析。

实验流程

1. 先将存放100张图的文件夹内的图片批量处理，生成存储切割数字的子文件夹，同时将需要存储的图片名，角点信息写入txt文件，方便之后用python读入xlsx中，如下所示：

<div> 此电脑 > DATA (D:) > 计算机视觉 > FinalProject > testSet > imageoutput </div>				
nt	<input type="checkbox"/>	名称	修改日期	类型
		大小		
		image15331178	2019/1/2 15:13	文件夹
		image15331220	2019/1/2 13:57	文件夹
	<input checked="" type="checkbox"/>	image15331419	2019/1/2 14:43	文件夹
		image15350039	2019/1/2 12:01	文件夹
		image15350043	2019/1/2 12:01	文件夹
		image15352057	2019/1/2 14:43	文件夹
		image15353005	2019/1/2 12:01	文件夹
		image16340006	2019/1/2 12:01	文件夹
		image16340017	2019/1/2 12:01	文件夹
		image16340023	2019/1/2 14:48	文件夹
		image16340024	2019/1/2 12:01	文件夹
		image16340025	2019/1/2 13:59	文件夹
		image16340028	2019/1/2 17:18	文件夹
		image16340032	2019/1/2 12:01	文件夹
		image16340042	2019/1/2 12:01	文件夹
		image16340044	2019/1/2 12:01	文件夹
s (C:)		image16340046	2019/1/2 12:01	文件夹
D:)		image16340049	2019/1/2 12:01	文件夹
E:)		image16340050	2019/1/2 12:01	文件夹
F:)		image16340054	2019/1/2 12:01	文件夹
RY (G:)		image16340055	2019/1/2 12:01	文件夹
U:)		image16340056	2019/1/2 13:59	文件夹
		image16340064	2019/1/2 12:01	文件夹
		image16340071	2019/1/2 12:01	文件夹
		image16340072	2019/1/2 12:01	文件夹
		image16340084	2019/1/2 12:01	文件夹
		image16340085	2019/1/2 17:22	文件夹
		image16340088	2019/1/2 12:01	文件夹
		image16340089	2019/1/2 12:01	文件夹
		image16340090	2019/1/2 17:28	文件夹
		image16340094	2019/1/2 17:33	文件夹
		image16340109	2019/1/2 12:01	文件夹
		image16340115	2019/1/2 12:01	文件夹
		image16340116	2019/1/2 12:01	文件夹
		image16340121	2019/1/2 12:01	文件夹
		image16340125	2019/1/2 12:01	文件夹
		image16340128	2019/1/2 17:43	文件夹
		image16340129	2019/1/2 17:46	文件夹
		image16340131	2019/1/2 12:01	文件夹
		image16340133	2019/1/2 12:01	文件夹
		image16340135	2019/1/2 12:01	文件夹
		image16340139	2019/1/2 12:01	文件夹
		image16340141	2019/1/2 12:01	文件夹

<input type="checkbox"/> 名称	修改日期	类型	大小
 15331178	2019/1/2 15:31	文本文档	1 KB
<input type="checkbox"/>  15331220	2019/1/2 13:52	文本文档	1 KB
 15331419	2019/1/2 14:43	文本文档	1 KB
 15350039	2019/1/2 12:01	文本文档	1 KB
 15350043	2019/1/2 12:01	文本文档	1 KB
 15352057	2019/1/2 14:43	文本文档	1 KB
 15353005	2019/1/2 12:01	文本文档	1 KB
 16340006	2019/1/2 12:01	文本文档	1 KB
 16340017	2019/1/2 12:01	文本文档	1 KB
 16340023	2019/1/2 12:01	文本文档	1 KB
 16340024	2019/1/2 12:01	文本文档	1 KB
 16340025	2019/1/2 13:59	文本文档	1 KB
 16340028	2019/1/2 12:01	文本文档	1 KB
 16340032	2019/1/2 12:01	文本文档	1 KB
 16340042	2019/1/2 12:01	文本文档	1 KB
 16340044	2019/1/2 12:01	文本文档	1 KB
 16340046	2019/1/2 12:01	文本文档	1 KB
 16340049	2019/1/2 12:01	文本文档	1 KB
 16340050	2019/1/2 12:01	文本文档	1 KB
 16340054	2019/1/2 12:01	文本文档	1 KB
 16340055	2019/1/2 12:01	文本文档	1 KB
 16340056	2019/1/2 13:59	文本文档	1 KB
 16340064	2019/1/2 12:01	文本文档	1 KB
 16340071	2019/1/2 12:01	文本文档	1 KB
 16340072	2019/1/2 12:01	文本文档	1 KB
 16340084	2019/1/2 12:01	文本文档	1 KB
 16340085	2019/1/2 12:01	文本文档	1 KB
 16340088	2019/1/2 12:01	文本文档	1 KB
 16340089	2019/1/2 12:01	文本文档	1 KB
 16340090	2019/1/2 12:01	文本文档	1 KB
 16340094	2019/1/2 12:01	文本文档	1 KB
 16340109	2019/1/2 12:01	文本文档	1 KB
 16340115	2019/1/2 12:01	文本文档	1 KB
 16340116	2019/1/2 12:01	文本文档	1 KB
 16340121	2019/1/2 12:01	文本文档	1 KB
 16340125	2019/1/2 12:01	文本文档	1 KB
 16340128	2019/1/2 12:01	文本文档	1 KB
 16340129	2019/1/2 12:01	文本文档	1 KB
 16340131	2019/1/2 12:01	文本文档	1 KB
 16340133	2019/1/2 12:01	文本文档	1 KB
 16340135	2019/1/2 12:01	文本文档	1 KB
 16340139	2019/1/2 12:01	文本文档	1 KB
 16340141	2019/1/2 12:01	文本文档	1 KB

类型: 文本文档

大小: 60 字节

修改日期: 2019/1/2 13:52

```
struct dirent *ptr;
DIR *dir;
dir = opendir("./testSet/bmp");
string txtpath = "./testSet/imagetxt/";
```

```

string outpath = "./testSet/imageoutput/";
while ((ptr=readdir(dir)) != NULL) {
    if (ptr->d_name[0] == '.')
        continue;
    CImg<float> img;
    //stringstream ss;
    //ss << i;
    //string picname = ss.str() + ".jpg";
    string s = ptr->d_name;
    string ss = ptr->d_name;
    ss.erase(8);
    string picpath = "./testSet/bmp/";
    string picname = picpath + s;
    string txtname = txtpath + ss + ".txt";
    string outfile = outpath + "image" + ss;
    ofstream oFile;
    oFile.open(txtname.c_str(), ios::out|ios::trunc);
    oFile << s << endl;
    oFile.close();
    img.load(picname.c_str());
    ImageSegmentation picture(img);
    picture.process(outfile, txtname.c_str());
}

```

2. 然后用python将各个子文件夹的图片批量处理：

```

if __name__=="__main__":

    #todo('./imageoutput/image15331178', './imageoutput/15331178.txt')
    path = './imageoutput/'
    txtpath = './imagetxt/'
    dirs = os.listdir(path)
    for dir in dirs:
        temp = dir[5:]
        temppic = path + dir + '/'
        temptxt = txtpath + temp + '.txt'
        todo(tempic, temptxt)

```

- 这里tensorflow比较有坑，如果在一个进程内多次导入ckpt会报错

```
saver.restore(sess, "../model/model.ckpt")#这里使用了之前保存的模型参数
```

- 所以我们只需在函数开始处加一个声明清空图对应的全局栈内所有信息

```
tf.reset_default_graph()
```

实验分析

虽然说期末项目是把平时做的一些东西整合到一起，但实际做的时候就会发现其实整合到一起直接搞的话并不会有那么好的效果，其实canny, hough还有矫正部分倒还比较好办，主要是调到合适的参数同时排除掉一部分拍摄不好图片的特殊线条噪点等的影响。

主要改动的部分一部分是图像分割的部分，开始时考虑了很多办法，比如最初简单的固定值全局阈值分割，再到老师讲的OSTU大津法以及全局迭代分割或者双峰分割，效果都很不好。后来想到自己的思路太过于局限，一直都在考虑全局分割，而整幅图的光照各处不同，效果肯定不会很好，所以想到了如果局部分割效果会不会好一点。开始时想的方法也比较简单，只是类似将纸张的每一行裁成等比例的块进行局部求阈值，但这种方法缺点是如果块裁的太大那么会损失很多需要的细节，如果太小又会出现棋盘效应。之后又查了资料以及询问了大佬的意见，了解到opencv有一个封装的自适应阈值分割算法，于是查了opencv源码，阅读之后自己重写了一下这个函数，倒不是很麻烦。然后发现效果很不错，关键点细节基本都在。

然后再就是对A4纸中的数字一个个分割出来，开始想的是将纸张先横向进行灰度统计，然后取高峰间的直线，这样可以将每一行的数字分割出来，然后再用相同的办法对于每一行竖向灰度统计然后再切割出单个数字。但实际发现这样效果对于10张图还好，但对于后面100张测试图就有点炸。于是考虑用K-means的方法进行聚类，但聚类的方法局限性比较强，要将每一行切割出并确定好每一行数字的个数并且需要合适的初始质点，不适用。于是参考了网上大佬的一个叫做区域连通标记的方法，采用二次扫描的办法，对连通区域内的数字切割出来，整体效果还不错。但缺点是有点上下有覆盖情况的图片比较难切割出来，如果结合投影法一起会好一些。

然后到了单个数字识别，开始用老师讲的分分类器Adaboost以及SVM，但是效果一般。尤其是Adaboost，对于mnist的测试集，也只有百分之60多的正确率。于是参考tensorflow中文社区上CNN的讲解用CNN的方法对数字进行识别，mnist测试集的识别率最高达到了99.3%，对于自己的数字识别效果也是不错的。

10张图：

1. 针对那十张图来说，整体识别率还是不错的，但是那十张图其实很多比较大的难度是在于有的图片外部光照过强导致难以通过霍夫变换拟合出较好的直线，于是针对此类图片采用如下办法，对外层高光区域进行滤波，减弱边缘突出程度。

```
result = imgIn.get_blur(3.0)
cimg_for_insideXY(result, x, y, DIFF) {
    result(x, y) = imgIn(x, y);
}
```

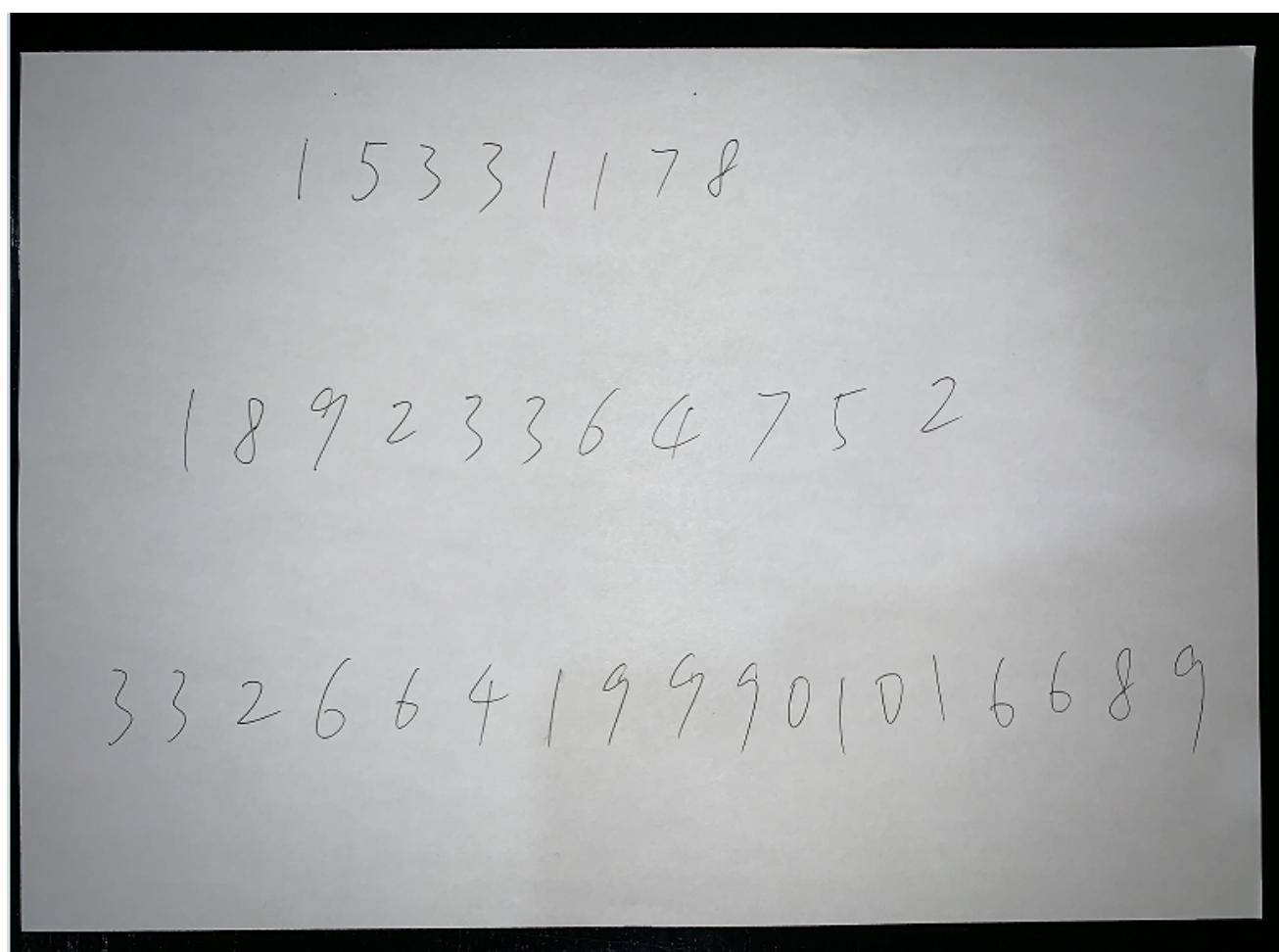
- 这十张图识别的准确率还是很高的，差不多90%多。

	A	B	C	D	E	F	G	H	I
1		图片名	角点1	角点2	角点3	角点4	学号	手机号	身份证号
2	0	1. jpg	(76, 267)	(1251, 247)	(1280, 1905)	(105, 1926)	15331353	13277132461	456757132466331578
3	1	2. jpg	(90, 220)	(1283, 220)	(1283, 1863)	(119, 1883)	15331180	13260831048	442313171803223057
4	2	2. jpg	(90, 220)	(1283, 220)	(1283, 1863)	(119, 1883)	15331180	13260831048	442313197803273057
5	3	2. jpg	(90, 220)	(1283, 220)	(1283, 1863)	(119, 1883)	15331180	13260831048	442313197803273057
6	4	3. jpg	(177, 286)	(1213, 250)	(1238, 1691)	(226, 1709)	15331024	13827418392	441221194612273012
7	5	3. jpg	(177, 286)	(1213, 250)	(1238, 1691)	(226, 1709)	15331029	13827418372	441721119612273012
8	6	3. jpg	(177, 286)	(1213, 250)	(1238, 1691)	(226, 1709)	15331027	13827418392	441721199612273012
9	7	4. jpg	(1327, 307)	(1205, 1702)	(1205, 307)	(70, 307)	15331344	15521145678	350102177602294321
10	8	4. jpg	(1327, 307)	(1205, 1702)	(1205, 307)	(70, 307)	15331344	15521145678	350102199602294321
11	9	4. jpg	(1327, 307)	(1205, 1702)	(1205, 307)	(70, 307)	15331344	15521145678	350102199602294321
12	10	5. jpg	(38, 75)	(1200, 35)	(1200, 1546)	(90, 1546)	15331364	13632552831	441281734503013435
13	11	5. jpg	(38, 75)	(1200, 35)	(1200, 1546)	(90, 1546)	15331364	13632552831	441281734503013435
14	12	5. jpg	(38, 75)	(1200, 35)	(1200, 1546)	(90, 1546)	15331369	13632550831	941281734503093435
15	13	6. jpg	(117, 258)	(1235, 239)	(1235, 1781)	(170, 1781)	15331266	13619154721	445301191810252734
16	14	6. jpg	(117, 258)	(1235, 239)	(1235, 1781)	(170, 1781)	15331096	13619154721	445301199010022734
17	15	6. jpg	(117, 258)	(1235, 239)	(1235, 1781)	(170, 1781)	15331046	13619154721	445301111810252734
18	16	7. jpg	(59, 312)	(1183, 312)	(1210, 1882)	(87, 1920)	15331351	13719274506	445202200001010058
19	17	7. jpg	(59, 312)	(1183, 312)	(1210, 1882)	(87, 1920)	15331351	13119275506	445707200001010258
20	18	7. jpg	(59, 312)	(1183, 312)	(1210, 1882)	(87, 1920)	15331351	13717275506	445202200001010058
21	19	8. jpg	(258, 241)	(1317, 390)	(1108, 1875)	(49, 1726)	15331052	15560563783	350426199712221012
22	20	8. jpg	(258, 241)	(1317, 390)	(1108, 1875)	(49, 1726)	13331052	15560563783	350426191712221012
23	21	8. jpg	(258, 241)	(1317, 390)	(1108, 1875)	(49, 1726)	15331052	15560563183	330426117712221012
24	22	9. jpg	(1334, 43)	(1715, 861)	(541, 1460)	(99, 591)	15331348	15626411418	440302189610022218
25	23	9. jpg	(1334, 43)	(1715, 861)	(541, 1460)	(99, 591)	15331348	15626411418	440302199610022218
26	24	9. jpg	(1334, 43)	(1715, 861)	(541, 1460)	(99, 591)	15331348	15626411418	440302199610022218
27	25	10. jpg	(451, 325)	(1382, 557)	(1057, 1961)	(41, 1669)	15331347	18950182323	350105199705042312
28	26	10. jpg	(451, 325)	(1382, 557)	(1057, 1961)	(41, 1669)	15331347	18950182323	350105199705042314
29	27	10. jpg	(451, 325)	(1382, 557)	(1057, 1961)	(41, 1669)	15331347	18750182323	350105171705092314

80张图：

对于那80张测试图就很难达到这么好的效果，主要是很多图片例如有一些涂掉的噪点或者汉字啥的，这种本来考虑如果有时间做的话可以通过数字识别CNN的softmax之后的概率情况，只有当最高概率大于一定值才保留，这样就能去掉一些模棱两可的数字，但后来要期末复习时间真的不够了，可以考虑下学期如果选曾老师的实训在完善一下。

1. 第一幅图就比较坑，是一个横向的图片，所以批量处理的时候第一幅就bug了，然后其实这倒比较简单，通过点之间的长度以及到原点的欧式距离判断纸张类型，然后读入。



2. 还有一种比较难处理的图片就是字写得特别“秀气”的，比如下图中间一行，因为这种比较难把握膨胀的程度，膨胀大了要么就成了黑球要么就数字连到一起切不开，小了分割效果不好，所以采用了邻域点数统计，点数达到一个较高的值才取黑。

15 3 5 205 7

4 414 26 1998 0301 0016

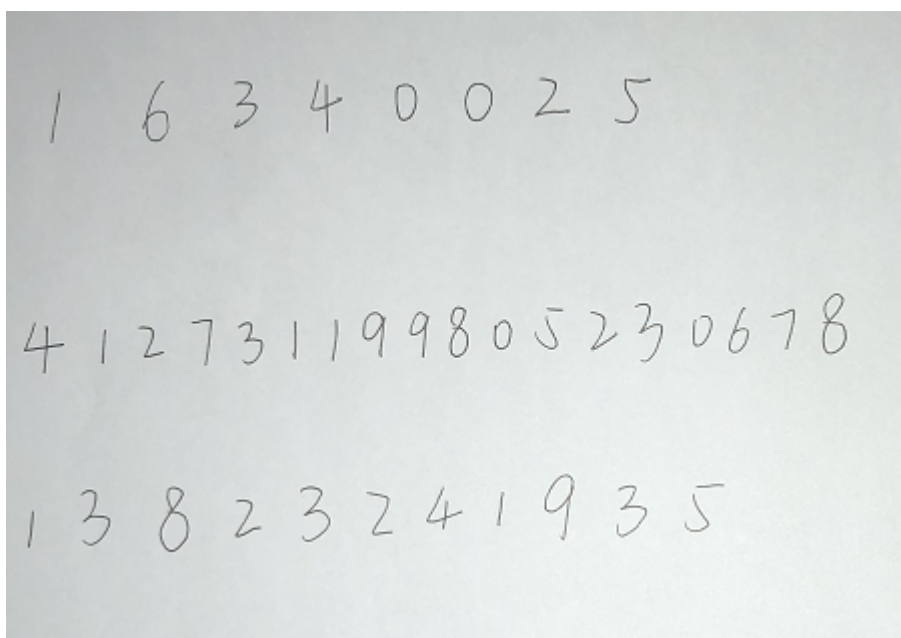
159 890 444 20

3. 还有一种图就是存在上下重叠的，比如说两个2在一起就很容易重叠，比如说下图这样两个2：，采用的方法是要在膨胀前就尽可能处理好，再横向添加一个负的参数滤波器，减弱数字间的连接程度。

• 22

4. 然后对于手机号和身份证顺序换了的问题，学号判断简单，就是八位数字，然后手机号的话虽然是11位，但有的图片存在分类效果不好，可能会切掉证件的11位，那就很糟糕了，通过如下正则表达式形式进行判断，那么剩下的就是证件号了。

```
import re
def ReTel(tn):
    reg = "1[3|4|5|7|8][0-9]{9}"
    return re.findall(reg, tn)
```



5. 最后是最难处理的也就是有字的那种图，开始想的方法就是上面提到的通过数字识别CNN的softmax之后的概率情况，只有当最高概率大于一定值才保留，这样就能去掉一些模棱两可的数字。但是时间不够最后就只是手动把那些字用邻域的空白矩阵覆盖住了。

姓名： 周翔宇

学号： 16341024

手机号： 15626264024

身份证号： 410882199808088053

- 最终将数据全部写入xlsx中：

	B	C	D	E	F	G	H	I
1	图片名	角点1	角点2	角点3	角点4	学号	手机号	身份证号
2	'15331178. bmp	' (12, 69)	' (1514, 61)	' (1522, 1121)	' (13, 1137)	'15331178	'18723366152	'332568177701016687
3	'15331220. bmp	' (34, 67)	' (1036, 49)	' (1061, 1483)	' (34, 1501)	'15331220	'15521220011	'51310219761127611
4	'15331419. bmp	' (189, 192)	' (1052, 177)	' (1007, 1465)	' (127, 1388)	'15331419	'13891173100701	'2521496275196170504717
5	'15350039. bmp	' (112, 140)	' (1094, 122)	' (1118, 1545)	' (112, 1527)	'15350037	'13599063728	'440103200103165501
6	'15350043. bmp	' (84, 53)	' (1100, 35)	' (1125, 1498)	' (84, 1498)	'15350043	'158202050701	'440107191701270618
7	'15352057. bmp	' (165, 161)	' (1091, 144)	' (1067, 1517)	' (120, 1451)	'1535297	'15181084426	'491426119883010016
8	'15353005. bmp	' (146, 80)	' (1124, 80)	' (1024, 1506)	' (78, 1390)	'15353005	'15521336318	'147701178012011711
9	'16340006. bmp	' (135, 121)	' (1090, 121)	' (1138, 1506)	' (111, 1525)	'16380006	'13307088217	'510822199812226881
10	'16340017. bmp	' (90, 123)	' (1081, 123)	' (1081, 1537)	' (90, 1519)	'16340011	'13572833323	'440301198105201234
11	'16340023. bmp	' (65, 65)	' (1110, 11)	' (1110, 1519)	' (89, 1501)	'16340023	'13432769341	'441581199811275972
12	'16340024. bmp	' (67, 57)	' (1076, 57)	' (1076, 1480)	' (67, 1480)	'16340024	'18188800111	'440119981234567870
13	'16340025. bmp	' (108, 139)	' (1036, 106)	' (1036, 1519)	' (62, 1452)	'16340025	'13823269933	'412731199805230678
14	'16340028. bmp	' (130, 137)	' (1055, 121)	' (1124, 1447)	' (153, 1498)	'16340028	'13060520007	'445102199811111001
15	'16340028. bmp	' (130, 137)	' (1055, 121)	' (1124, 1447)	' (153, 1498)	'13540000	'13064750001	'455103199911224511
16	'16340032. bmp	' (66, 117)	' (1052, 100)	' (1077, 1515)	' (66, 1533)	'16340032	'15890182290	'241440803189806148
17	'16340042. bmp	' (81, 71)	' (1058, 71)	' (1058, 1458)	' (81, 1441)	'16380042	'18637665571	'941127199810246428
18	'16340044. bmp	' (58, 54)	' (1099, 36)	' (1099, 1506)	' (83, 1488)	'16340044	'13322632720	'441602199808230419
19	'16340046. bmp	' (70, 68)	' (1119, 50)	' (1119, 1521)	' (95, 1521)	'16340046	'13725583658	'661301188711110111
20	'16340049. bmp	' (106, 43)	' (1104, 43)	' (1104, 1479)	' (81, 1461)	'16340047	'150777704386	'470129199609175943
21	'16340050. bmp	' (129, 116)	' (1077, 99)	' (1077, 1418)	' (151, 1435)	'16340050	'1342273619	'441002181603183659
22	'16340054. bmp	' (75, 41)	' (1091, 59)	' (1067, 1493)	' (50, 1475)	'16340054	'13354325528	'441883191811080421
23	'16340055. bmp	' (51, 82)	' (1079, 63)	' (1104, 1544)	' (51, 1544)	'16340055	'18420053672	'441427199802081020
24	'16340056. bmp	' (54, 49)	' (1119, 30)	' (1145, 1536)	' (80, 1555)	'16340056	'18026587823	'441301187711270666
25	'16340064. bmp	' (133, 120)	' (1004, 135)	' (1026, 1415)	' (66, 1415)	'15340068	'18811881381	'440102111811151812
26	'16340071. bmp	' (120, 74)	' (1117, 74)	' (1117, 1485)	' (120, 1485)	'16340011	'18718023344	'471224111701010011
27	'16340072. bmp	' (139, 72)	' (1140, 72)	' (1140, 1515)	' (115, 1497)	'16340072	'18825086611	'440181177105245130
28	'16340084. bmp	' (210, 183)	' (1028, 168)	' (1135, 1397)	' (189, 1430)	'16340084	'13424571224	'442000189812310760
29	'16340085. bmp	' (201, 217)	' (1065, 217)	' (1043, 1506)	' (137, 1443)	'16340085	'13750401982	'640582199803216174
30	'16340088. bmp	' (86, 101)	' (1067, 50)	' (1093, 1514)	' (86, 1479)	'16340088	'15626287371	'441602199710131750
31	'16340089. bmp	' (139, 94)	' (1115, 59)	' (1140, 1522)	' (91, 1485)	'16340089	'13424111354	'440107199803150310
32	'16340090. bmp	' (107, 39)	' (1125, 20)	' (1072, 1542)	' (34, 1451)	'16390080	'135080582289	'440681111808032018
33	'16340094. bmp	' (149, 122)	' (1060, 106)	' (1083, 1400)	' (171, 1400)	'16340074	'15720227802	'454242199802303826
34	'16340109. bmp	' (124, 115)	' (1052, 115)	' (1075, 1450)	' (100, 1468)	'16340109	'15879822193	'440602188808050931
35	'16340115. bmp	' (82, 82)	' (1110, 82)	' (1110, 1556)	' (57, 1538)	'16340115	'13822385824	'440808088604190123
36	'16340116. bmp	' (120, 120)	' (1089, 120)	' (1089, 1474)	' (143, 1474)	'16340116	'13128567363	'444888199004012345
37	'16340121. bmp	' (163, 113)	' (1107, 113)	' (1131, 1458)	' (163, 1476)	'16340121	'15020537674	'372125111111123310
38	'16340125. bmp	' (148, 188)	' (1044, 157)	' (1044, 1471)	' (127, 1438)	'16340125	'15120823518	'510106111801223516
39	'16340128. bmp	' (58, 69)	' (1125, 69)	' (1100, 1567)	' (58, 1549)	'16380128	'41860124485	'07137398889235088
40	'16340129. bmp	' (68, 13)	' (1146, 13)	' (1120, 1515)	' (68, 1515)	'16340124	'13102818687	'44042119970311811

- 最终粗略的与群里收集的信息表做一对比，正确率80%左右。

```
(py35) D:\计算机视觉\FinalProject\testSet>python
the accuracy is:
83.235%
```