

# *A new way of digit recognition*

—based on Bayesian probability (Python OpenCV)

## 手写体识别数字的一种新方法

——基于贝叶斯概率

July 10.10 2022

Author: Xuzhe\_2021211896\_徐喆

School: HFUT

E-mail: 15021699275@163.com

### Contents

1. 摘要与发展现状
2. 数学基础理论
3. 程序展示
4. 程序实例
5. 算法优势
6. 改进与分析

Index

## Abstract:

实际上手写体的数字识别是人工智能编程领域的入门课，现在主流的数字识别都是基于简单的 K-近邻算法，在处理数字识别的效率是极高的。但是 K-近邻算法要求的算力是相当高的（相对于嵌入式而言），比如使用 K-近邻算法以 STM32 为平台进行车牌识别，如果使用 STM32F1 系列识别速度是相当慢的，当然使用 STM32F4 乃至 STM32F7 系列的情况可能有所改观，但这无疑增加了成本。

从传统机器学习的角度来看，对图像的特征提取是相当困难的，如何在使用相对较小的资源来精确的刻画图像的特征是如今研究的方向，即将数据降维处理。

从图形学的角度来看，数字实际上是线的位置组合，其核心元素是线性的，意味着对其进行抽象加法的处理将会相当有效。使用 K-近邻算法来仅仅识别数字可以说是大材小用并且浪费资源，本文创新性地提出如下的数学模型，可以极其高效并且简单的识别如数字这一类的线性图像[1]。

[1]:所谓线性图像，指的是该图像的关键特征是由点与线构成的，而不是非线性的，如圆，椭圆，离散点…

## Mathematical model:

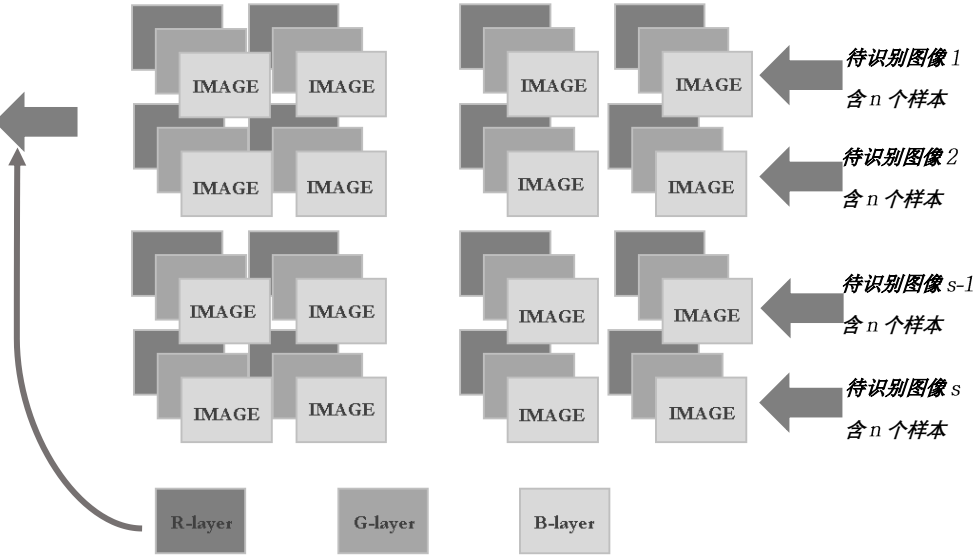
Simple Matrix:

$$\begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{s1} & \cdots & A_{sn} \end{bmatrix} = S$$

Image matrix:

### NOTE:

实际上，仅仅对与手写体识别的处理而言，由于样本图像是黑白的，所以只需要提取一个位面即可。



$$A_i \begin{pmatrix} a_{11} & \cdots & a_{1p_1} & \cdots & a_{1p_r} & \cdots & a_{1p_r} \\ a_{21} & \cdots & a_{2p_1} & \cdots & a_{2p_r} & \cdots & a_{2p_r} \\ \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{q1} & \cdots & a_{qp_1} & \cdots & a_{qp_r} & \cdots & a_{qp_r} \end{pmatrix}$$

$$\left\{ \begin{bmatrix} a_{11} & \cdots & a_{1p_1} \\ \vdots & \ddots & \vdots \\ a_{q1} & \cdots & a_{qp_1} \end{bmatrix}, \dots, \begin{bmatrix} a_{11} & \cdots & a_{1p_r} \\ \vdots & \ddots & \vdots \\ a_{q1} & \cdots & a_{qp_r} \end{bmatrix} \right\}$$

生成特征矩阵 (贝叶斯概率矩阵):  $\alpha_i$

$$\begin{pmatrix} \left( \begin{matrix} \sum_1^{p_1} \sum a_{1i} / p_1 & \cdots & \sum_1^{p_r} \sum a_{1i} / p_r \\ \vdots & \ddots & \vdots \\ \sum_1^{p_1} \sum a_{qi} / p_1 & \cdots & \sum_1^{p_r} \sum a_{qi} / p_r \end{matrix} \right) \\ \left( \begin{matrix} \sum_1^{p_1} \sum b_{1i} / p_1 & \cdots & \sum_1^{p_r} \sum b_{1i} / p_r \\ \vdots & \ddots & \vdots \\ \sum_1^{p_1} \sum b_{qi} / p_1 & \cdots & \sum_1^{p_r} \sum b_{qi} / p_r \end{matrix} \right) \\ \vdots \\ \left( \begin{matrix} \sum_1^{p_1} \sum \delta_{1i} / p_1 & \cdots & \sum_1^{p_r} \sum \delta_{1i} / p_r \\ \vdots & \ddots & \vdots \\ \sum_1^{p_1} \sum \delta_{qi} / p_1 & \cdots & \sum_1^{p_r} \sum \delta_{qi} / p_r \end{matrix} \right) \end{pmatrix} = \alpha_i$$

生成偏移矩阵 (权重矩阵):  $\mu$

$$\left( \begin{pmatrix} \varepsilon_{11} \\ \vdots \\ \varepsilon_{q1} \end{pmatrix} \quad \dots \quad \begin{pmatrix} \varepsilon_{1r} \\ \vdots \\ \varepsilon_{qr} \end{pmatrix} \right) = \boldsymbol{\mu}$$

输入样本矩阵 (对象矩阵):  $\mathbf{A}$

$$\mathbf{A} \begin{pmatrix} a_{11} & \dots & a_{1p_1} & \dots & a_{1p_r} & \dots & a_{1p_r} \\ a_{21} & \dots & a_{2p_1} & \dots & a_{2p_r} & \dots & a_{2p_r} \\ \vdots & \dots & \vdots & \dots & \vdots & \dots & \vdots \\ a_{q1} & \dots & a_{qp_1} & \dots & a_{qp_r} & \dots & a_{qp_r} \end{pmatrix} = \mathbf{A}$$

生成样本特征矩阵:  $\mathbf{A}^F$

$$\begin{pmatrix} \sum_1^{p_1} \sum a_{1i} & \dots & \sum_1^{p_r} \sum a_{1i} / p_r \\ \vdots & \ddots & \vdots \\ \sum_1^{p_1} \sum a_{qi} & \dots & \sum_1^{p_r} \sum a_{qi} / p_r \end{pmatrix} = \mathbf{A}^F$$

生成偏离矩阵:  $\mathbf{D}$

$$\begin{pmatrix} (|\alpha_1 - \mathbf{A}^F * \boldsymbol{\mu}|) \\ \vdots \\ (|\alpha_s - \mathbf{A}^F * \boldsymbol{\mu}|) \end{pmatrix} = \mathbf{D}$$

Then: 生成偏差向量

$$\mathbf{D} \longrightarrow \begin{pmatrix} \sum (|\alpha_1 - \mathbf{A}^F * \boldsymbol{\mu}|) \\ \vdots \\ \sum (|\alpha_s - \mathbf{A}^F * \boldsymbol{\mu}|) \end{pmatrix} = \vec{\mathbf{D}}$$

NOTE:

上述求和符的含义是将所有的矩阵元素求和, 即从矩阵空间以普通求和的方式映射到数域, 再构成一个向量。

NOTE:

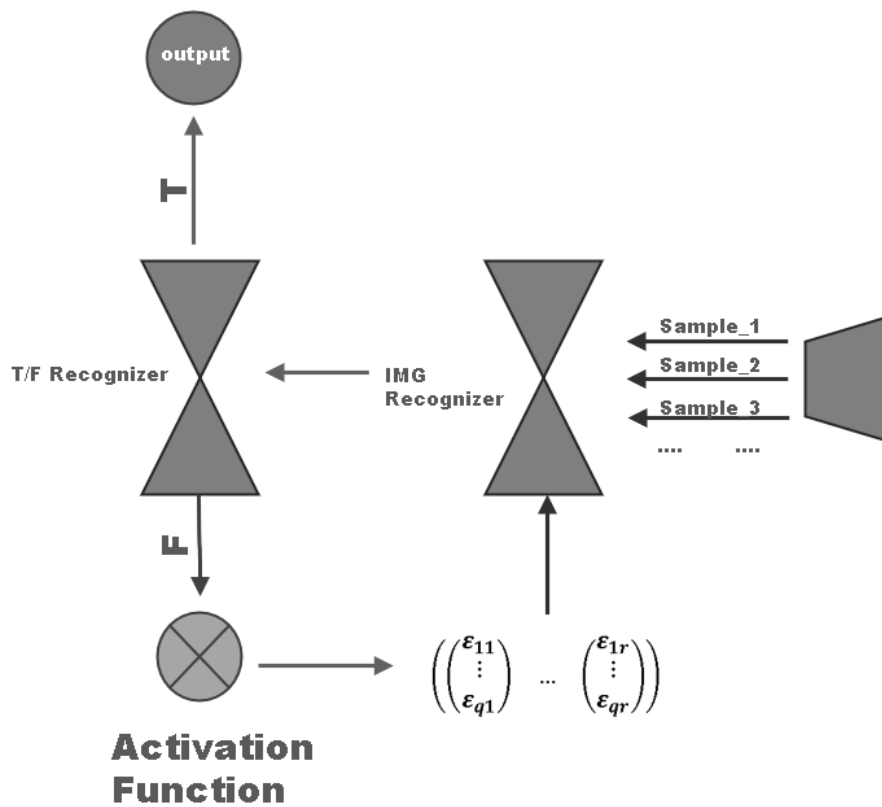
$\vec{\mathbf{D}}$  就是我们所求得量, 其最大分量*i*的位置代表输入图像与样本*i*的拟合度最高。

**NOTE:**

实际上权重矩阵我们现在未知，当然我们可以假定权重矩阵中的元素全部为 1，即

$$\left( \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \quad \dots \quad \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right)$$

但效果比较差，我们想程序自动矫正特征矩阵，所以下列使用向量机的方法，算法流程如下（基于MCP神经元）



## Program presentation:

### 特征向量生成部分:

```
1 '''
2 Remark:
3 从总体思路上来说，K近邻算法的确十分先进，
4 但是数字识别可以说是大材小用了，
5 主体思路是通过泛值处理获得每个手写体的两个特征向量
6 并且使用贝叶斯概率学进行计算，从理论上来说是完全可行的
7 '''
8 import cv2 as cv
9 import numpy as np
10 filename = 'digits.png'
11 t_value = 2
12 img = cv.imread(filename)
13 gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
14 # 现在我们将图像分割为5000个单元格，每个单元格为20x20
15 cells = [np.hsplit(row, 100) for row in np.vsplit(gray, 50)]
16 img_staff = np.array(cells).astype(np.float32)
17 # 使其成为一个Numpy数组。它的大小将是 (50, 100, 20, 20)
18 feature_Vector = np.zeros((10, 20), dtype=np.float32)
19 # 那么我将feature_Vector 作为训练集
20 row, col, img_x, img_y = img_staff.shape
21 for num_index in range(10):
22     for staff_index in range(num_index*5, (num_index+1)*5):
23         for y in range(col):
24             for x in range(img_x):
25                 feature_Vector[num_index, x] = (np.sum(img_staff[staff_index, y, x]) + feature_Vector[num_index, x]) / t_value
26 np.savez('feature_vector.npz', feature_Vector = feature_Vector)
27 print('Data build successfully! -By HFUT C14')
28
29
```

## 图像识别器:

```
1  """
2  这里是算法的核心部分
3  """
4  import numpy as np
5  import cv2 as cv
6
7  filename = 'feature_vector.npz'
8  threaHold = [
9      1.0, 1.0, 1.0, 1.0, 1.0,
10     1.0, 1.0, 1.0, 1.0, 1.0,
11     1.0, 1.0, 1.0, 1.0, 1.0,
12     1.0, 1.0, 1.0, 1.0, 1.2,
13 ]
14 erro = 0
15 all_ex = 0
16 # 首先我们先导入数据
17 data = np.load(filename)
18 feature_Vector = data['feature_Vector']
19 num, col = feature_Vector.shape
20 err_bar = np.zeros((10, 20), dtype=np.float32)
21 print('The shape of Feature_Vector is:', feature_Vector.shape)
22
23
24 def distinguish_num(num_img):
25     for index in range(num):
26         for i in range(col):
27             err_bar[index, i] = np.abs(np.sum(num_img[i]) * threaHold[i] - feature_Vector[index, i])
28
29
30 def give_num():
31     ans = []
32     for i in range(10):
33         ans.append(np.sum(err_bar[i]))
34     return ans
35
36
37 def num_output():
38     return give_num().index(min(give_num()))
39
40
41 img = cv.imread('digits.png')
42 gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
43 # 现在我们将图像分割为5000个单元格，每个单元格为20x20，进行训练
44 cells = [np.hsplit(row, 100) for row in np.vsplit(gray, 50)]
45 img_staff = np.array(cells).astype(np.float32)
46
47
48 # 接下来开始训练矫正
49 def num_correct(img, index):
50     for i in range(col):
51         if np.sum(img[i]) > feature_Vector[index, i]:
52             threaHold[i] = 1 - feature_Vector[index, i] / np.sum(img[i]) * 0.2
53         elif np.sum(img[i]) < feature_Vector[index, i]:
54             threaHold[i] = 1 + np.sum(img[i]) * 0.4 / feature_Vector[index, i]
55         else:
56             pass
57     row, col_, img_x, img_y = img_staff.shape
```

```

49 def num_correct(img, index):
50     for i in range(col):
51         if np.sum(img[i]) > feature_Vector[index, i]:
52             threaHold[i] = 1 - feature_Vector[index, i] / np.sum(img[i]) * 0.2
53         elif np.sum(img[i]) < feature_Vector[index, i]:
54             threaHold[i] = 1 + np.sum(img[i]) * 0.4 / feature_Vector[index, i]
55         else:
56             pass
57     row, col_, img_x, img_y = img_staff.shape
58     for num_index in range(10):
59         for staff_index in range(num_index * 5, (num_index + 1) * 5):
60             for i in range(col_):
61                 distinguish_num(img_staff[staff_index, i])
62                 if num_output() != num_index:
63                     num_correct(img_staff[staff_index, i], num_index)
64     print('Generate successfully!! -HFUT:', threaHold)
65
66
67 def accurate_calu():
68     global all_ex, erro
69     for num_index in range(10):
70         for staff_index in range(num_index * 5, (num_index + 1) * 5):
71             for i in range(col_):
72                 distinguish_num(img_staff[staff_index, i])
73                 # print(num_output())
74                 all_ex = all_ex + 1
75                 if num_output() != num_index:
76                     erro = erro + 1
77     print(f'The accurate is {(all_ex - erro) / all_ex * 100} %')
78     return f'{(all_ex - erro) / all_ex * 100} %'

```



## 图形化界面：

```
82     创建用户交互界面
83     '''
84     drawing = False # 按下鼠标则为真
85
86
87     def nothing(x):
88         pass
89
90
91     def draw(event, x, y, flags, param):
92         global drawing
93         if event == cv.EVENT_LBUTTONDOWN: # 响应鼠标按下
94             drawing = True
95         elif event == cv.EVENT_MOUSEMOVE: # 响应鼠标移动
96             if drawing == True:
97                 img_window[y:y + 20, x:x + 20] = (255, 255, 255)
98         elif event == cv.EVENT_LBUTTONUP: # 响应鼠标松开
99             drawing = False
100
101
102     img_window = np.zeros((300, 300, 3), np.uint8)
103     cv.namedWindow('image')
104     # 创建颜色变化的轨迹栏
105     accuracy = 'accuracy'
106     clear = 'clear'
107     distinguish = 'dist-num'
108     append = 'append'
109     right = 'right'
110     cv.createTrackbar(right, 'image', 0, 9, nothing) # 所写之字的正确数字
111     cv.createTrackbar(append, 'image', 0, 1, nothing) # 加入训练集中
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

127     if ac == 1:
128         cv.setTrackbarPos(accuracy, 'image', 1)
129         cv.putText(img_window, accurate_calu(), (5,290), cv.FONT_HERSHEY_COMPLEX_SMALL, 1, (100, 200, 200), 1)
130         cv.setTrackbarPos(accuracy, 'image', 0)
131     if c == 1:
132         cv.setTrackbarPos(clear, 'image', 1)
133         img_window[:] = (0, 0, 0)
134         flag = 0
135     if d == 1: # 识别数字
136         cv.setTrackbarPos(distinguish, 'image', 1)
137         target_img = img_window.copy()
138         target_img = cv.resize(target_img, (20, 20))
139         target_img = cv.cvtColor(target_img, cv.COLOR_BGR2GRAY)
140         distinguish_num(target_img)
141         cv.putText(img_window, f'{num_output()}', (5, 30), cv.FONT_HERSHEY_COMPLEX_SMALL, 1, (100, 200, 200), 1)
142     if a == 1 and flag == 0:
143         #重新特征定位
144         flag = 1
145         cv.setTrackbarPos(append, 'image', 1)
146         r = cv.getTrackbarPos(right, 'image')
147         print('Adding numbers --HFUT ', str(r))
148         if r != num_output():
149             try:
150                 num_correct(target_img, r)
151                 cv.putText(img_window, f'Adding successfully --HFUT', (5, 30), cv.FONT_HERSHEY_PLAIN, 1, (100, 200, 200),
152                             1)
153             except NameError:
154                 cv.putText(img_window, f'Please dist-num please', (5, 30), cv.FONT_HERSHEY_PLAIN, 1,
155                             (100, 200, 200), 1)
156         else:
157             pass




```

Program presentation II:

首先先生成特征矩阵（即贝叶斯矩阵）并且通过.npz的 float64 位形式保存。

Data build successfully! -By HFUT C14

Process finished with exit code 0

 digits.png  
 feature\_vector.npz  
 knn\_data.npz

特征矩阵形式：

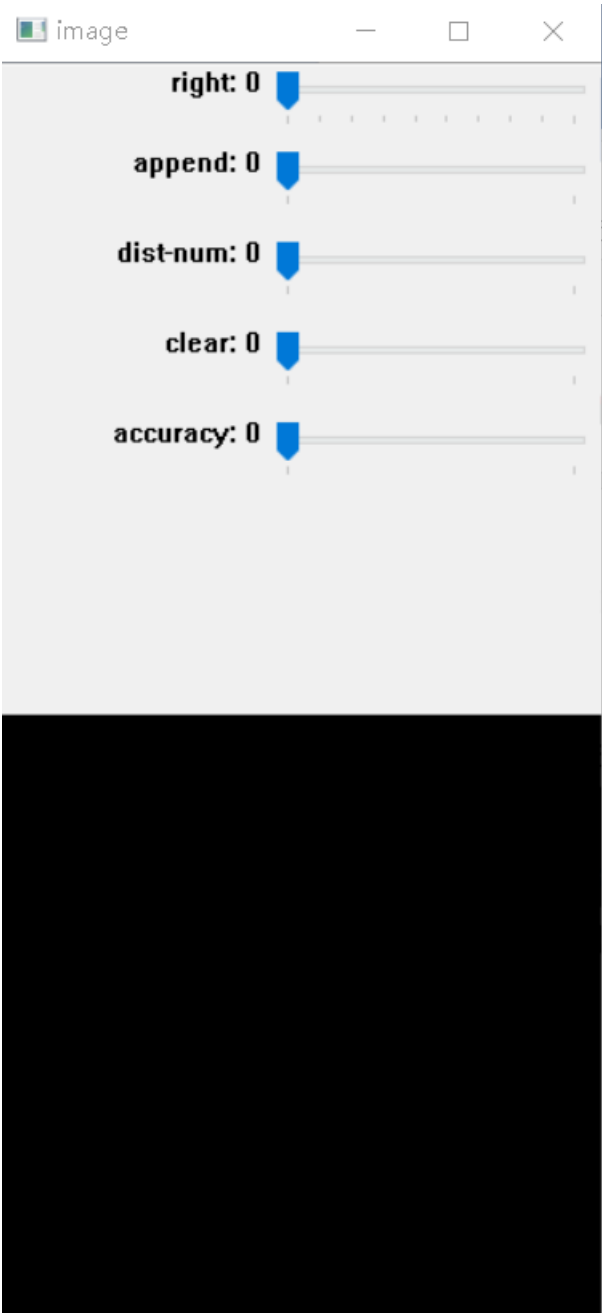
	2	3	4	5	6	7	8	9	
0	5296	164.55191	1004.69006	1729.72217	2034.30566	2075.97168	2009.36963	1737.11462	146
1	52171	400.56018	599.41418	632.67017	678.94312	669.70435	648.98157	655.75537	614
2	3.12833	1302.99829	1456.33447	1531.22131	1853.70520	1462.50098	826.79590	550.35248	539
3	1586	470.45624	1539.28723	1590.52271	1087.93298	898.85449	1083.17883	1299.70581	103
4	0209	148.89948	570.01642	865.10364	1082.13428	1106.90979	1244.41895	1461.74670	199
5	7495	14.80680	193.89281	965.52161	1335.59302	796.91284	576.39240	1044.79028	116
6	5.36633	745.32385	692.03357	608.40491	543.95593	546.71710	608.45923	986.07935	155
7	0000	0.00000	87.48870	1037.35974	2021.21375	1798.60986	1051.70203	768.25330	883
8	4110	275.76117	977.13202	1498.90649	1558.85925	1363.67017	1298.91638	1589.43555	125
9	5.08691	695.14087	1357.37793	2541.47070	1797.51208	1360.69043	1400.64832	1358.59119	149

# 其次运行主程序生成特征矩阵：

```
01 img_y = (int) 20
01 num = (int) 10
01 num_index = (int) 9
01 row = (int) 50
01 staff_index = (int) 49
[+] threshHold = (list: 20) [1.0909753356281162, 1.0585363986352838, 1.0795184481995541, 1.0966710532831665, 1.2531351014960916, ... Vie
[+] Special Variables
```

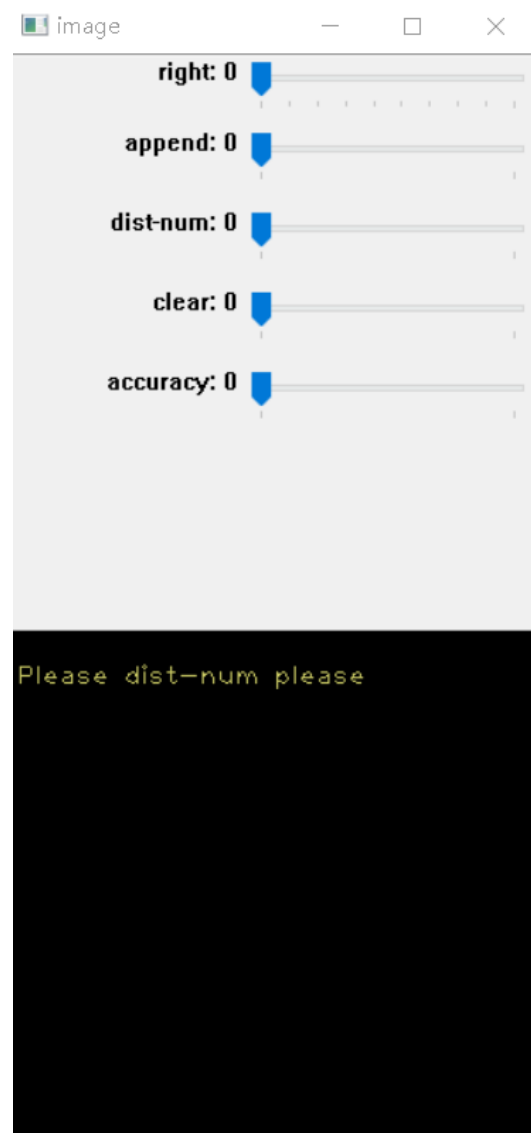
The shape of Feature\_Vector is: (10, 20)  
Generate successfully!! -MFUT: [1.0909753356281162, 1.0585363986352838, 1.0795184481995541, 1.0966710532831665, 1.2531351014960916, 1.1847749819583733, 1.1795815464612105, 1.228787

# 打开图形化界面



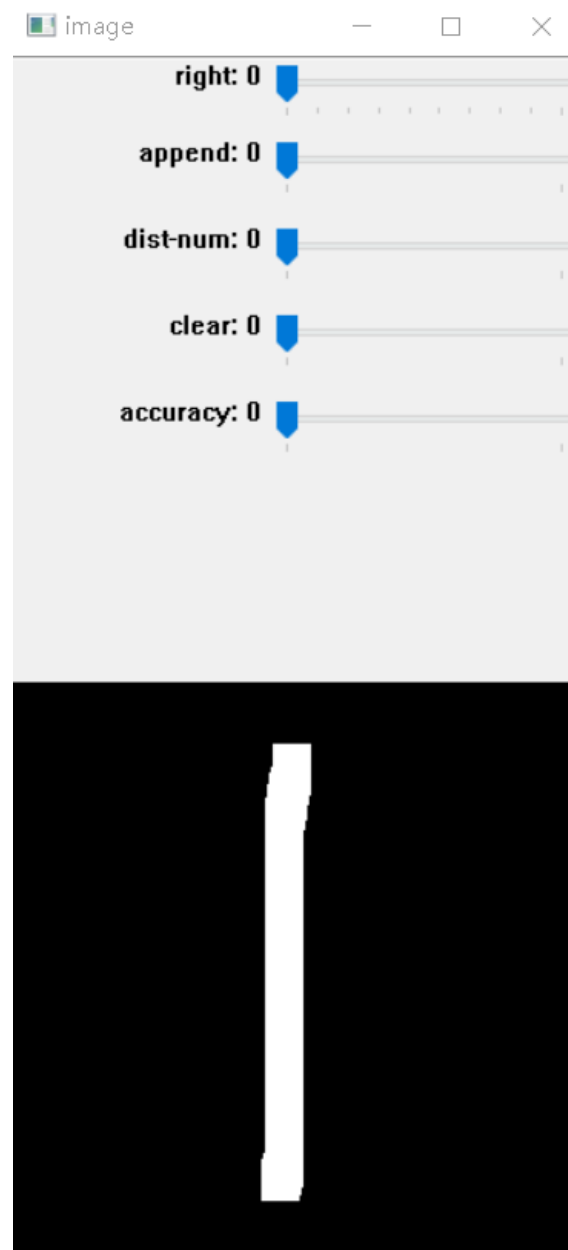
从上到下依次为：

- 1.手动矫正按键
- 2.调整权重矩阵按键
- 3.识别数字按键
- 4.清除
- 5.显示正确率
- 6.手动绘图区域



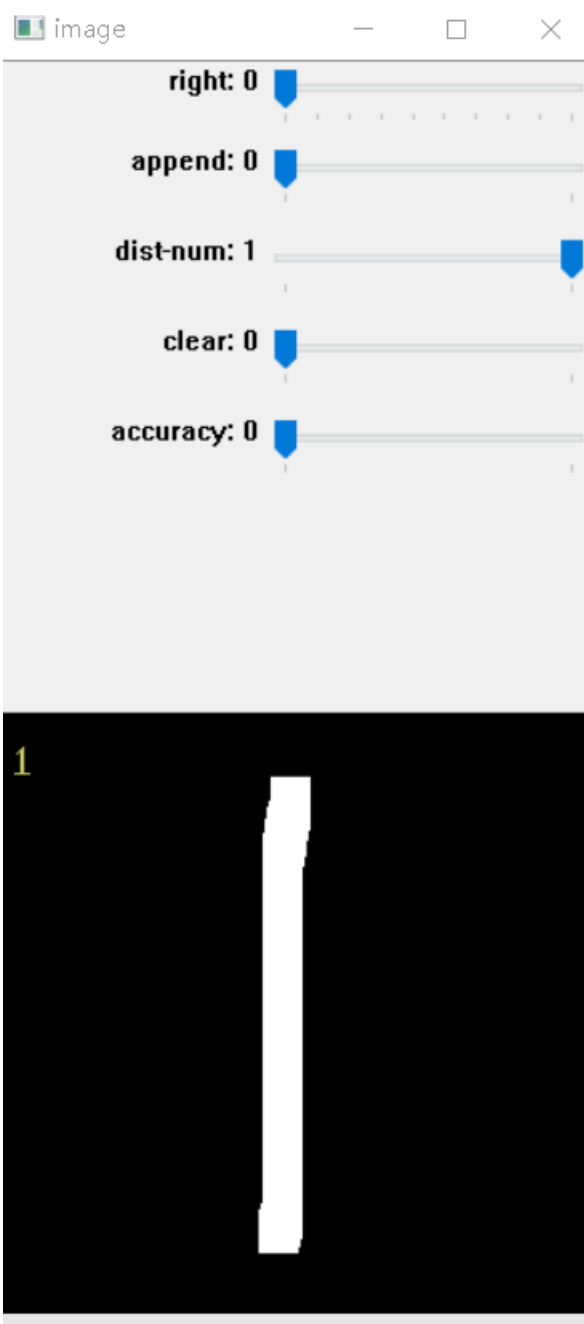
Step:如果没有识别数字，而点击.  
调整特征矩阵按键会提示先输入  
数字：

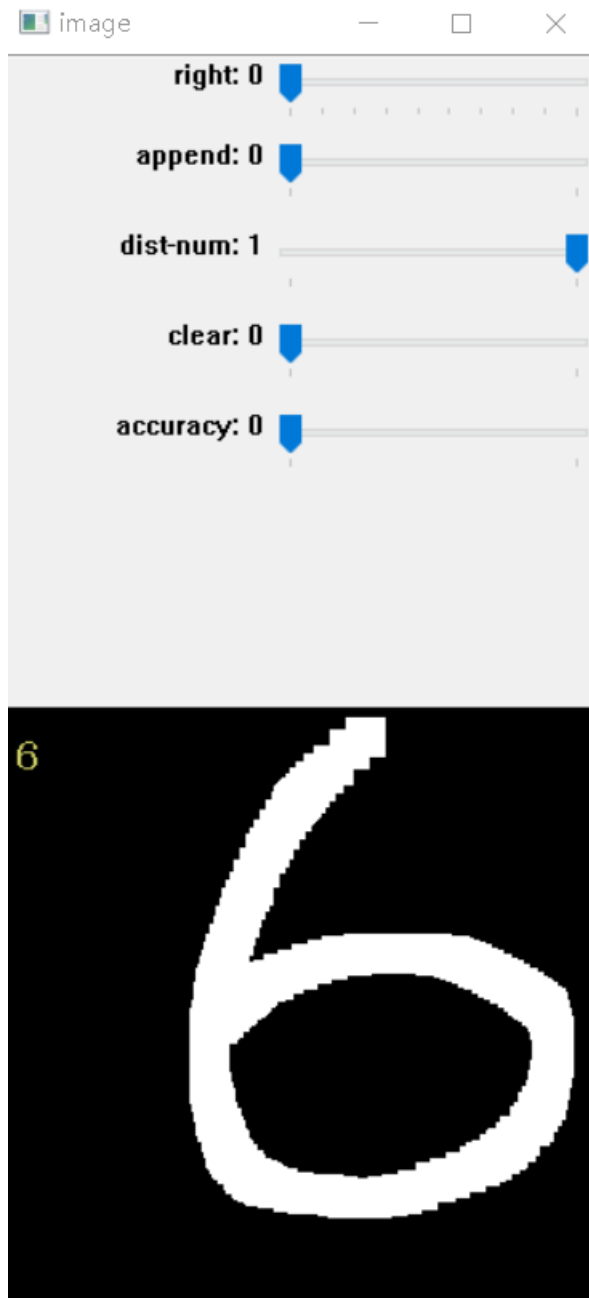
Step:在黑色区域写数字



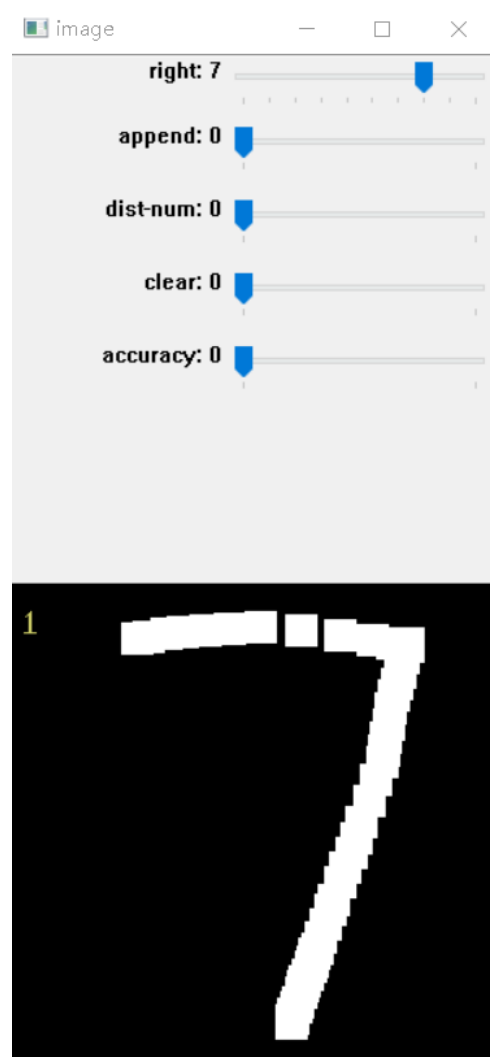
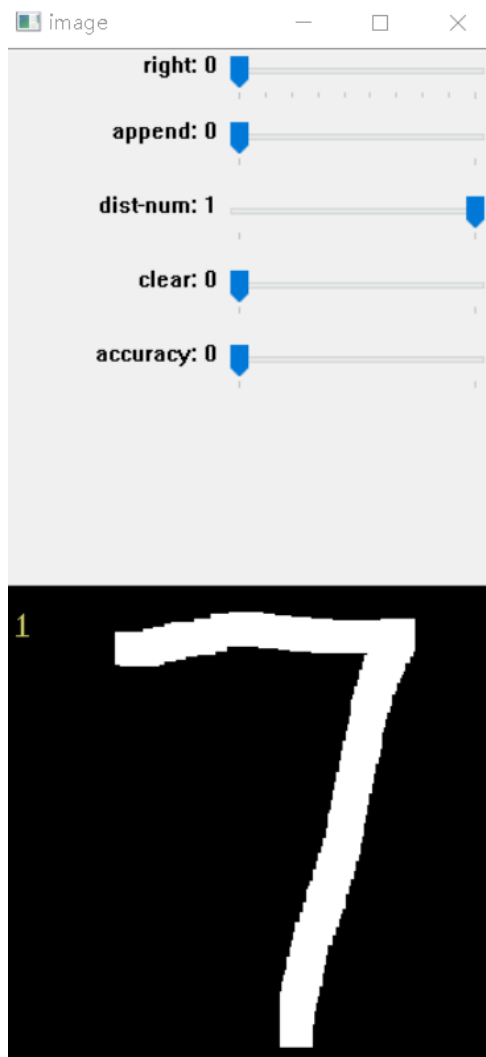
Step:

点击数字识别按钮识别数字:



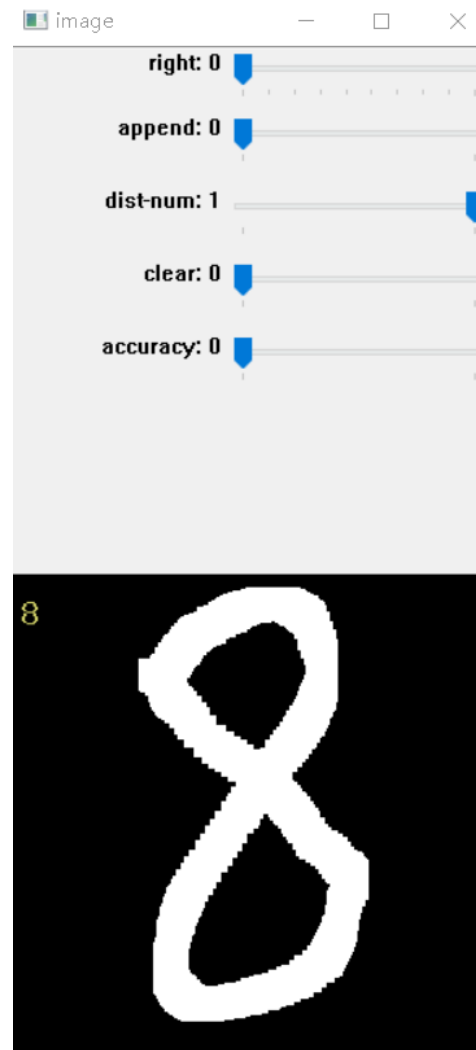
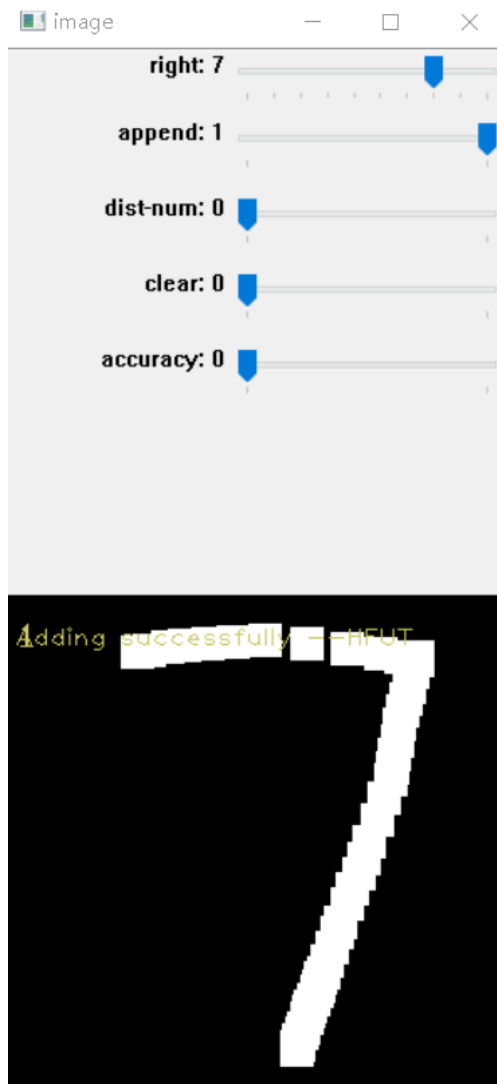


Step:如果识别错误，将手动调整按键调整至正确数字，点击调整权重矩阵按键，从而优化权重分配：

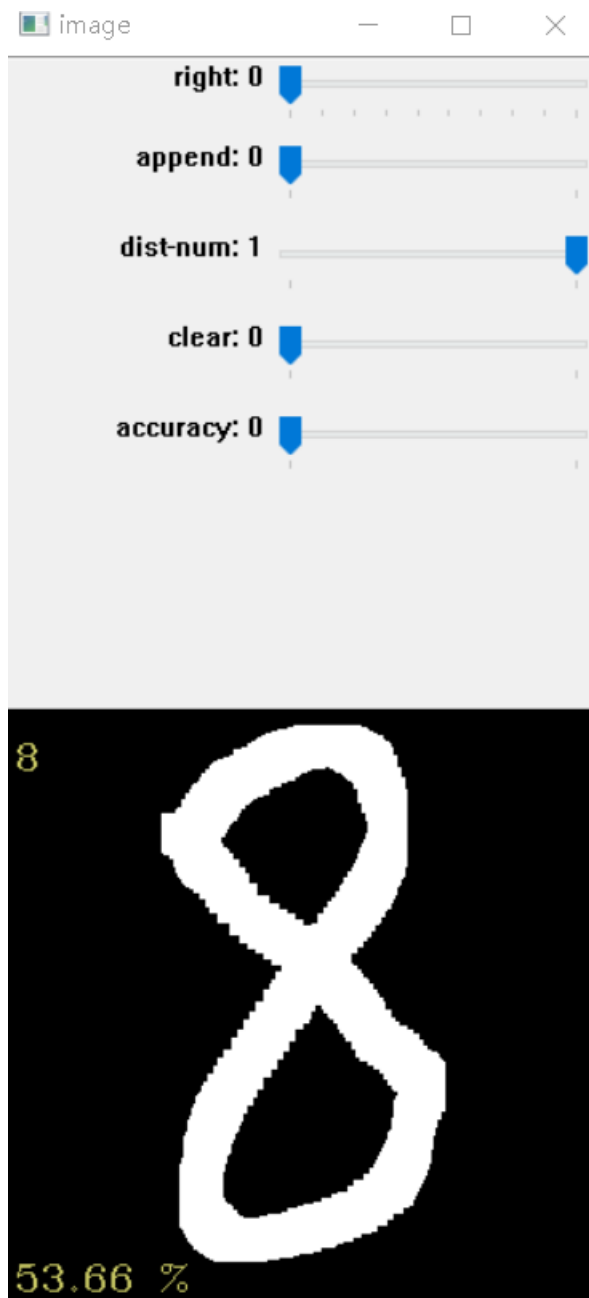


Step:再次识别提高正确率



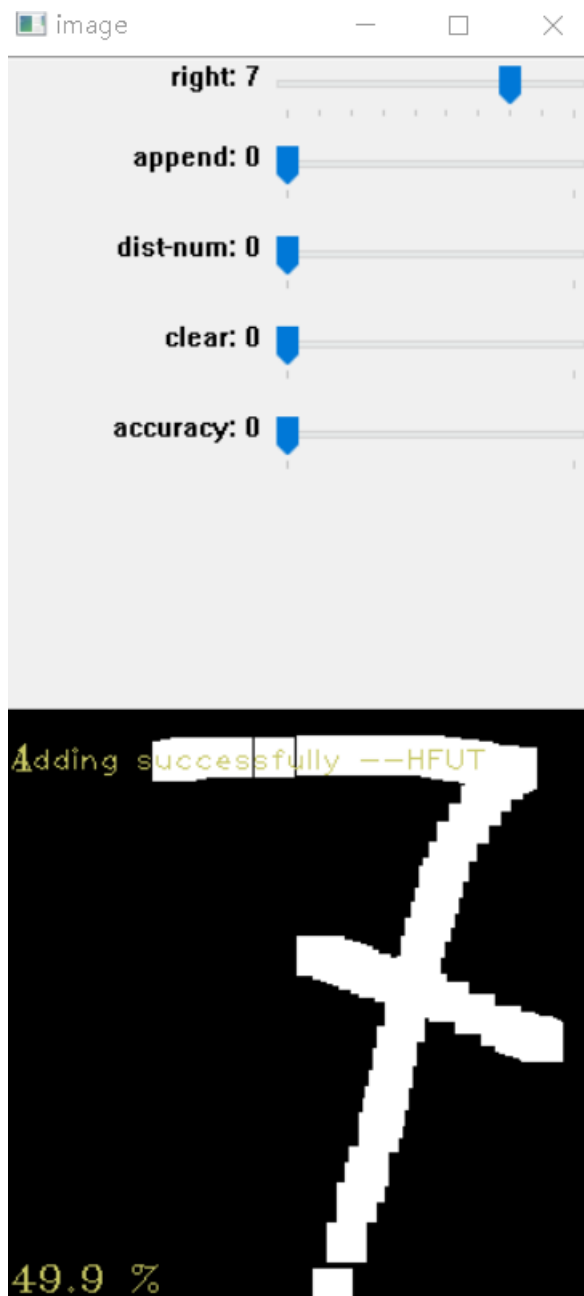


Step: 点击正确率按键显示正确率:



*Note:*

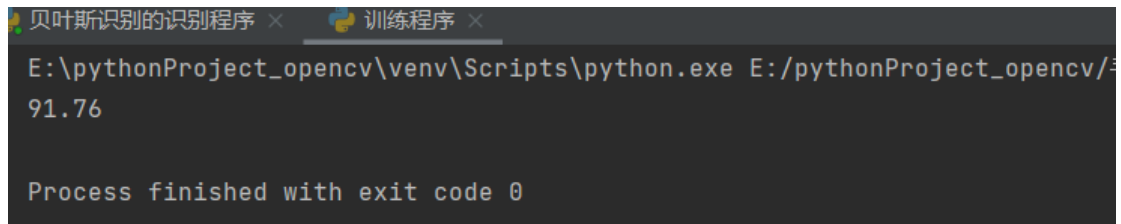
*Hoops! 正确率只有 53.66%*



Note:手动矫正正确率变低

Compare with other program:

### 1.K-近邻算法:



```
贝叶斯识别的识别程序 x 训练程序 x
E:\pythonProject_opencv\venv\Scripts\python.exe E:/pythonProject_opencv/...
91.76
Process finished with exit code 0
```

Note: K-近邻算法的正确率为 91.76%

而本程序的正确率最高只有 54.3%

Note:

但是本算法的优势在于其训练集的大小只有 2KB,

而使用 K-近邻算法的训练集的大小有 1957KB

对于嵌入式平台而言, 如果要识别更加复杂的图形, 如果使用 K-近邻算法其训练集大小将会大于系统内存。

Note:

本算法的处理核心是普通代数加法, 处理的速度远快于 K-近邻算法, 尽管 OpenCV 专门为 K-近邻算法做了优化。

## Improvements:

由于本人的 Python 技术水平有限，理论算法仅仅实现了一小部分，由于对 Numpy 库中矩阵分割的操作不熟练，无法将图形矩阵进行列分块，而仅仅将其作为整体来处理，导致正确率较低。其次，由于时间紧迫，相关参数并未进行详细的调整。导致本人设计的程序对纵向的线性特征不敏感，并且难以重构。

但总而言之，该算法的可行性还是相当高的。

## Note:

本项目将在 GitHub 上开源

Author: Xuzhe\_徐喆\_2021211896

*Reference:*

*1. <https://download.csdn.net/>*