Deliverable 3

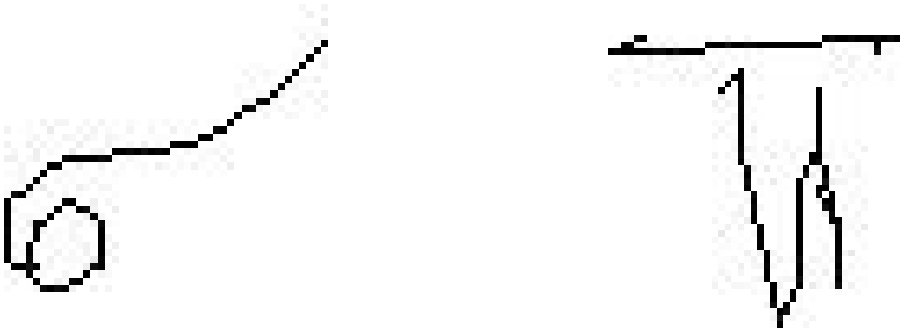
Training Results:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	130
1	0.90	0.97	0.93	1430
2	0.98	0.95	0.97	1436
3	0.98	1.00	0.99	2512
4	1.00	1.00	1.00	3401
5	0.90	0.96	0.93	692
6	0.90	0.94	0.92	2652
7	0.93	0.98	0.96	2615
8	0.95	0.99	0.97	1092
9	0.97	0.94	0.96	741
10	0.82	0.97	0.89	355
11	0.99	0.91	0.95	313
12	0.92	0.96	0.94	292
13	0.99	0.88	0.93	308
14	0.93	0.86	0.90	375
15	0.99	0.99	0.99	1311
16	0.98	0.95	0.97	1238
17	0.96	0.96	0.96	581
18	1.00	0.87	0.93	15
19	0.72	0.78	0.75	170
20	0.88	0.76	0.81	147
21	0.96	0.92	0.94	249
22	0.95	0.95	0.95	1087
23	0.84	0.79	0.82	268
24	0.89	0.70	0.79	142
25	0.89	0.81	0.85	328
26	0.94	0.94	0.94	2660
27	0.97	0.94	0.95	79
28	0.95	0.88	0.91	78
29	0.99	0.90	0.94	256
30	0.96	0.97	0.96	866
31	0.99	0.92	0.95	203
32	1.00	0.99	1.00	300
33	0.97	0.97	0.97	486
34	0.87	0.90	0.88	88

35	0.94	0.90	0.92	301
36	1.00	1.00	1.00	3
37	0.87	0.91	0.89	372
38	0.43	0.14	0.21	21
39	0.83	0.69	0.75	42
40	0.78	1.00	0.88	70
41	0.92	0.81	0.86	27
42	0.97	0.90	0.94	514
43	0.60	0.50	0.55	6
44	0.99	0.90	0.94	179
45	0.89	0.99	0.94	275
46	0.98	0.93	0.95	155
47	0.97	0.93	0.95	308
48	0.97	0.64	0.77	103
49	1.00	0.75	0.86	12
50	1.00	1.00	1.00	62
51	0.97	0.94	0.95	98
52	0.99	0.98	0.99	168
53	0.99	0.98	0.98	201
54	1.00	0.90	0.95	49
55	0.86	0.63	0.73	19
56	1.00	0.98	0.99	57
57	0.33	0.11	0.16	46
58	0.97	0.93	0.95	268
59	0.94	0.92	0.93	36
60	0.93	0.92	0.93	234
61	0.96	0.98	0.97	81
62	0.88	0.63	0.74	123
63	0.95	0.99	0.97	171
64	0.53	0.90	0.67	21
65	0.97	0.99	0.98	430
66	0.97	0.99	0.98	892
67	0.96	0.98	0.97	270
68	0.99	0.98	0.98	245
69	0.92	0.99	0.95	281
70	0.67	0.60	0.63	326
71	0.86	0.94	0.90	128
72	0.91	0.80	0.85	157
73	0.96	0.84	0.90	57
74	0.93	0.97	0.95	934
75	0.95	0.75	0.84	587
76	0.85	0.72	0.78	39
77	0.94	0.79	0.86	39
avg / total	0.95	0.95	0.94	37303

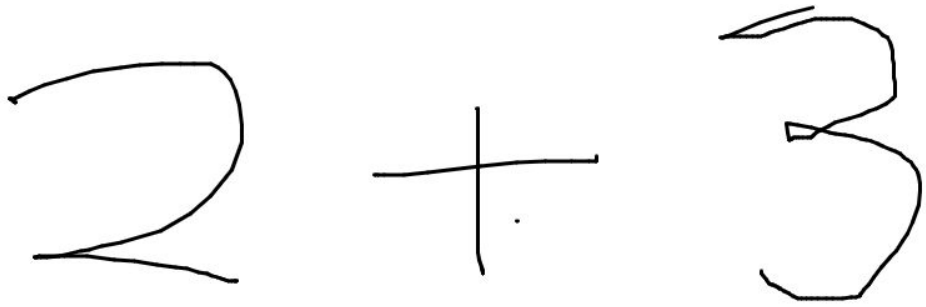
Accuracy: >0.998

The accuracy and precision have both improved slightly from deliverable 2. These results were obtained after training for 65 iterations. The final model will be trained on over 100 iterations, which should improve both fields slightly more. The precision is very low for some of the symbols. However, I will probably discard some of these symbols since the data itself is flawed:

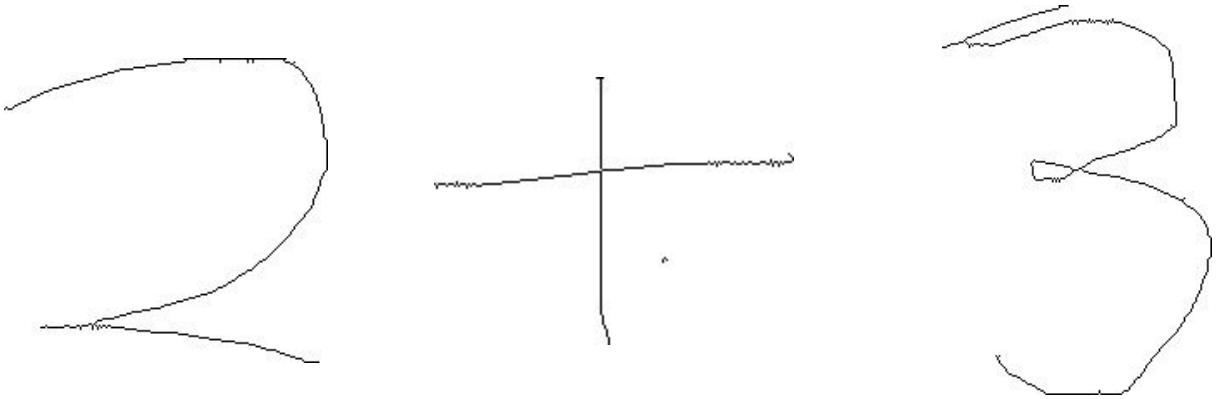


From right to left: sigma and pi

I believe that the reason the data is so strange is that it was skeletonized, which has some strange effects. I replicated the skeletonization process hoping that it would improve results on user generated data since the data would be more similar to what the model trained on. However, after testing, it actually made the result significantly worse. Consider the following simple example:

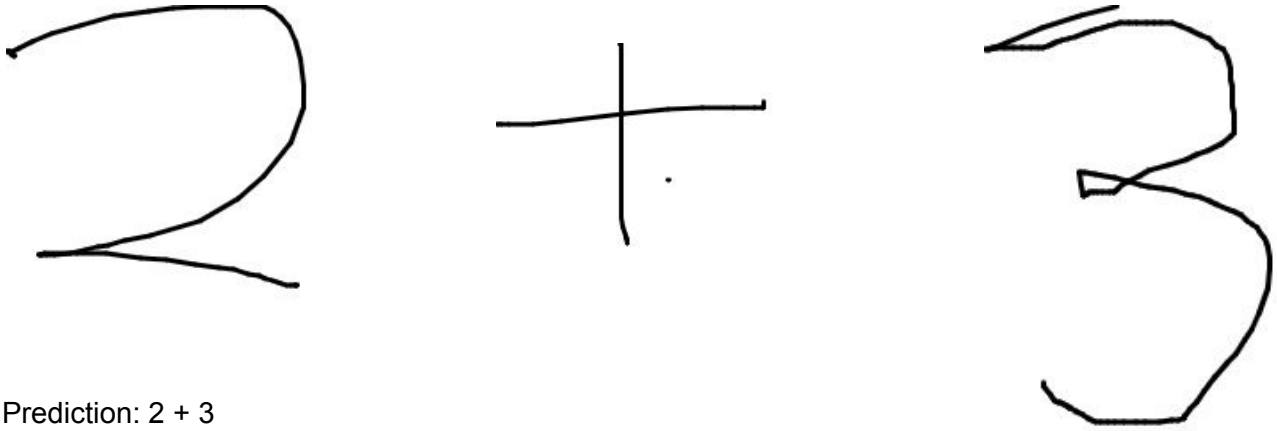


After segmentation and skeletonization we get:



Prediction: = = =

After only segmentation we get:



Prediction: 2 + 3

Clearly the skeletonized version causes difficulties for the model, despite being more similar to what it trained on. Note that when I applied the same test for typed symbols, both models performed similarly. I therefore believe that the way the pixels are being saved by the drawing app I created makes the skeletonization process slightly unpredictable. In short, what this does is it seeks to represent the image such that it has a single pixel width but so that the pixels are

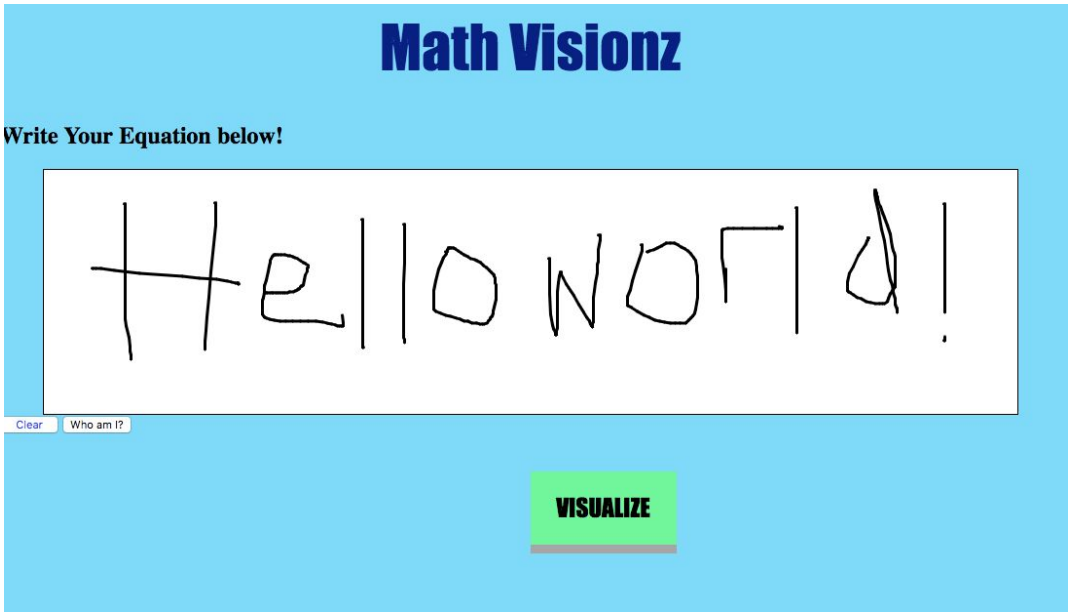
connected. As we can see from the examples, this doesn't always cause smooth results. In some places, the result is rather a zigzag effect which I believe is the source of the problem.

I have included the skeletonization code on github, however I will not be using it.

I have also included the segmentation code which is working almost perfectly. The only flaw is that it is seperating = sign into two seperate symbols. To solve this I plan on using morphological closing which should merge the two globs together into a single symbol. Interestingly it does not have the same issue when it comes to 'i's and 'j's because of the comparative size difference.

Final Project

I have decided to create a webapp where the user can draw on a canvas, download the image and upload it if they wish for it to be translated into math symbols. I would also like it to output the latex code for the input. The reason for the download is so that the user can keep their own version if they please, and they can even use it as a drawing board for other purposes. I have implemented a rudimentary version using JavaScript and HTML. It is on the project github. Here is a sample (this is NOT the final version of the interface).



The clear button simply calls refresh and resets the canvas. The “who am I” button simply outputs some information about the program and eventually the visualise button will be for uploading the image. The reason I decided to go with this approach rather than have users input hand drawn symbols is that it gives me more control of how thick the stroke width will be amongst other things. While in principle, I could overcome the other difficulties, in practice things turned out much more complicated (see the whole skeletonization fiasco above). Since OCR is still a relatively open problem, I feel as if taking on a simpler problem is more realistic. In addition, it’s pretty fun to play with it. I will add more features later (eraser, colors, pen types, paint, etc.) on to make it into a legit drawing board as well as a ML project. Canvas is an amazing tools that makes it fairly easy to do so.The only drawback is that in its current version, my JS code is not compatible with mobile browsers. I also cannot make any guarantees for Internet Explorer users:(.

I liked the name Math Visionz so much that I bought the domain name mathvisionz.com (for $0.99!)
I plan on hosting it using a free web hosting service.

I have very little (read: none) experience when it comes to integrating ML applications into a webapp, so I will rely on online tutorials, following a minimalist approach.