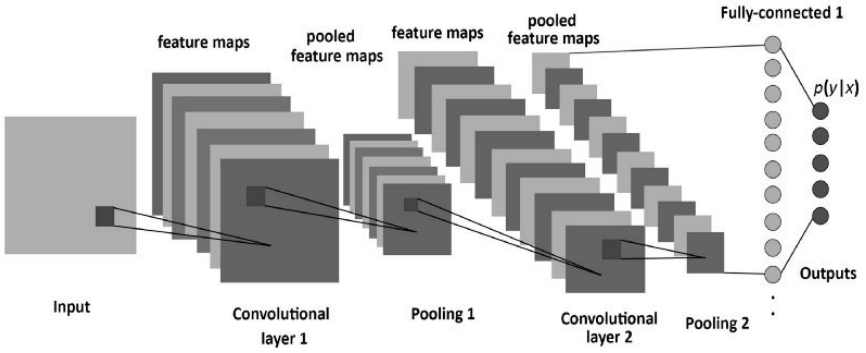
Project Deliverable 2

1. Problem Statement: The goal is to create a web app that will be able to detect, recognize and translate handwritten mathematical symbols. The app will either allow you to upload pictures, or allow you to directly draw into the browser using the cursor, but the final implementation has not been decided yet.

2. Data Preprocessing: The dataset has not been changed since the first proposal. It can be found here: https://www.kaggle.com/xainano/handwrittenmathsymbols. It consists of over 300 000 images of different mathematical symbols. I am contemplating expanding the dataset by adding different symbols from other datasets. For example, the symbols for epsilon, delta, and integral signs are not included. Currently, the dataset consists of over 80 types of symbols. For the training of the neural network, I used ImageDataGenerator from Keras to rescale the images (which were originally 45x45 pixels), to shear, zoom, flip, whiten, and rotate the images to get a more robust training model. I also changed the pictures to grayscale so that the color won't influence the training model. However, after testing on the validation set, I determined that the only preprocessing which was necessary was the rescaling. For data that will be gathered from users, skeletonization will need to be implemented as well to make the data more similar to what the model trained on.

3. Machine Learning Model: The model I implemented is a convolutional neural network, using Keras' Sequential for the implementation. The neural net consists of the following: 2 2D convolutional layers using relu activation function, a pooling layer using MaxPooling2D, a layer using Flatten, another layer using relu for the fully connected layer and finally an output layer using softmax (since the data is multi-label). I chose Keras for its ease of implementation, simplicity, and power. The following visual representation of a convolutional neural network provides a good overview of my own cnn:



source

For the training/validation/test split, I used a standard 0.9/0.1/0.1 split. Since the data set is so large, the exact proportion does not affect the final results in a significant way.

After testing on the validation data, I realised that the preprocessing that I originally had applied was lowering my accuracy. The reason for this is that rotations, sheers and zooms may be helpful for building a robust cnn when the images involved are few and complex, but for mathematical symbols they were destroying the integrity of the data. For example, after a rotation a ‘8’ and infinity become the same, or a forward slash and a ‘1’ become the same. I also threw out some of the symbols which were impossible for the cnn to learn on. For example, there was originally apostrophes and primes, however, without context they are impossible to recognize as they are simply drawn as straight lines, and their relative size and position is usually what allows us to recognize them.

4. After training the model for 50 epochs with 80 steps per epochs, the final accuracy was 0.9981 on the training set and 0.9984 on the test set. It is

interesting to note that after a single epoch, due to the large size of the data set, the accuracy was already at 0.9974 and 0.9978. However, after stopping the training after a single epoch, the model performed poorly on small samples which were randomly chosen from the training set. After 50 epochs, this problem disappeared. The precision was not as high, with a 0.93 precision and 0.93 recall after 50 epochs. This is mostly due to the fact that the precision on a few of the symbols was fairly low, with two of them even being below 0.5. These symbols included alpha and ‘not equal’. A large part of the problem seems to stem from the fact that many of the images are of poorly written mathematical symbols which would be almost impossible even for a human to recognize. However with over 300 000 images, it is impossible for me to manually remove all the bad ones. It could also be that training the cnn for longer might improve the accuracy and precision, since 50 epochs is relatively low. For example, raising the epoch number from 25 to 50 hardly affected accuracy, but raised precision from 0.89 to 0.93. To increase the precision even further, it might be necessary to completely replace the data for certain symbols, either my finding a higher quality dataset or by creating one manually.

The 0.9984 accuracy on the test set blows past the 0.9 baseline I had set in the first deliverable and clearly indicates that the project is feasible Further, the precision of 0.93 is pretty good, especially considering that it is disproportionately being lowered by a few symbols in particular . However, the 0.9 referred to the model’s performance on arbitrary test data gathered from users, not the actual test data from the sample. The greatest difficulty will be implementing localization while still maintaining a high degree of accuracy. Due to the model’s high performance recognizing single characters, the limiting factor will likely be how good it is at localizing the characters in a string.

This is the classification report obtained from scikit-learn:

	precision	recall	f1-score	support
0	1.00	0.97	0.98	130
1	0.91	0.97	0.94	1430
2	0.97	0.98	0.98	1436
3	0.96	1.00	0.98	2512
4	0.99	1.00	0.99	3401
5	0.89	0.94	0.92	692
6	0.90	0.92	0.91	2652
7	0.95	0.96	0.96	2615
8	0.97	0.97	0.97	1092
9	0.95	0.97	0.96	741
10	0.91	0.87	0.89	355
11	0.88	0.95	0.91	313
12	0.95	0.87	0.91	292
13	0.99	0.87	0.92	308
14	0.91	0.78	0.84	375
15	0.98	0.99	0.99	1311
16	0.93	0.97	0.95	1238
17	0.95	0.93	0.94	581
18	0.86	0.80	0.83	15
19	0.77	0.57	0.66	170
20	0.79	0.70	0.74	147
21	0.86	0.93	0.90	249
22	0.95	0.93	0.94	1087
23	0.83	0.84	0.83	268
24	0.69	0.73	0.71	142
25	0.83	0.73	0.77	328
26	0.93	0.92	0.92	2660
27	0.87	0.94	0.90	79
28	0.83	0.99	0.90	78
29	0.90	0.90	0.90	256
30	0.47	0.30	0.37	135
31	0.92	0.95	0.94	866

32	0.85	0.99	0.91	203
33	0.98	0.99	0.98	300
34	0.98	0.93	0.96	486
35	0.86	0.89	0.87	88
36	0.91	0.89	0.90	301
37	1.00	1.00	1.00	3
38	0.87	0.87	0.87	372
39	0.75	0.60	0.67	5
40	0.75	0.29	0.41	21
41	0.77	0.40	0.53	42
42	0.89	0.81	0.85	70
43	0.94	0.59	0.73	27
44	0.96	0.91	0.93	514
45	1.00	0.67	0.80	6
46	0.89	0.88	0.88	179
47	0.95	0.92	0.93	275
48	0.96	0.86	0.91	155
49	0.95	0.91	0.93	308
50	1.00	0.51	0.68	103
51	0.80	0.33	0.47	12
52	1.00	0.98	0.99	62
53	0.98	0.92	0.95	98
54	1.00	0.94	0.97	168
55	0.98	0.98	0.98	201
56	0.66	0.98	0.79	49
57	0.62	0.42	0.50	19
58	1.00	0.96	0.98	57
59	0.33	0.02	0.04	46
60	0.96	0.93	0.94	268
61	0.86	0.89	0.88	36
62	0.95	0.93	0.94	234
63	0.97	0.91	0.94	81
64	0.60	0.76	0.67	123
65	1.00	0.95	0.98	171

66	0.63	0.57	0.60	21
67	0.99	0.98	0.98	430
68	0.99	0.99	0.99	892
69	0.93	0.96	0.94	270
70	0.98	0.99	0.99	245
71	0.94	0.95	0.95	281
72	0.58	0.64	0.61	326
73	0.80	0.82	0.81	128
74	0.94	0.78	0.85	157
75	0.98	0.77	0.86	57
76	0.93	0.94	0.94	934
77	0.88	0.84	0.86	587
78	0.70	0.54	0.61	39
79	0.84	0.69	0.76	39
avg / total	0.93	0.93	0.93	37443

And here are the labels:

{0: '!', 1: '(', 2: ')', 3: '+', 4: '-', 5: '0', 6: '1', 7: '2', 8: '3', 9: '4', 10: '5', 11: '6', 12: '7', 13: '8', 14: '9', 15: '=', 16: 'A', 17: 'C', 18: 'Delta', 19: 'G', 20: 'H', 21: 'M', 22: 'N', 23: 'R', 24: 'S', 25: 'T', 26: 'X', 27: '[', 28: ']', 29: 'alpha', 30: 'ascii_124', 31: 'b', 32: 'beta', 33: 'cos', 34: 'd', 35: 'div', 36: 'e', 37: 'exists', 38: 'f', 39: 'forall', 40: 'forward_slash', 41: 'gamma', 42: 'geq', 43: 'gt', 44: 'i', 45: 'in', 46: 'infty', 47: 'int', 48: 'j', 49: 'k', 50: 'l', 51: 'lambda', 52: 'ldots', 53: 'leq', 54: 'lim', 55: 'log', 56: 'lt', 57: 'mu', 58: 'neq', 59: 'o', 60: 'p', 61: 'phi', 62: 'pi', 63: 'pm', 64: 'q', 65: 'rightarrow', 66: 'sigma', 67: 'sin', 68: 'sqrt', 69: 'sum', 70: 'tan', 71: 'theta', 72: 'times', 73: 'u', 74: 'v', 75: 'w', 76: 'y', 77: 'z', 78: '{', 79: '}'}

5. The next step, as mentioned, is to implement localization, meaning the ability for the program to take a string as input and to properly get the characters from it. Once this is accomplished, all that will remain is implementing the model into a web app, and especially deciding on how the user will input their data, either by photo or by drawing in the browser with a cursor. I am also considering having the final model output the latex code for the given input. Further, some common symbols such as integral sign are missing from the data set and if I find a good source for such symbols, I may add merge them with the current dataset.