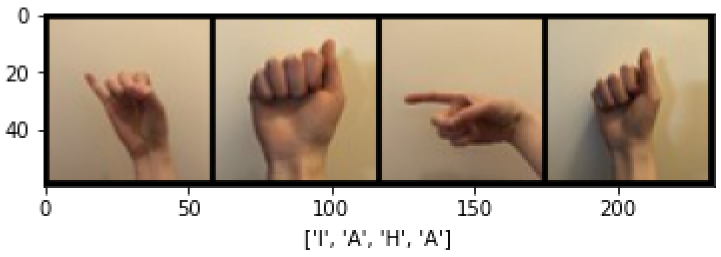
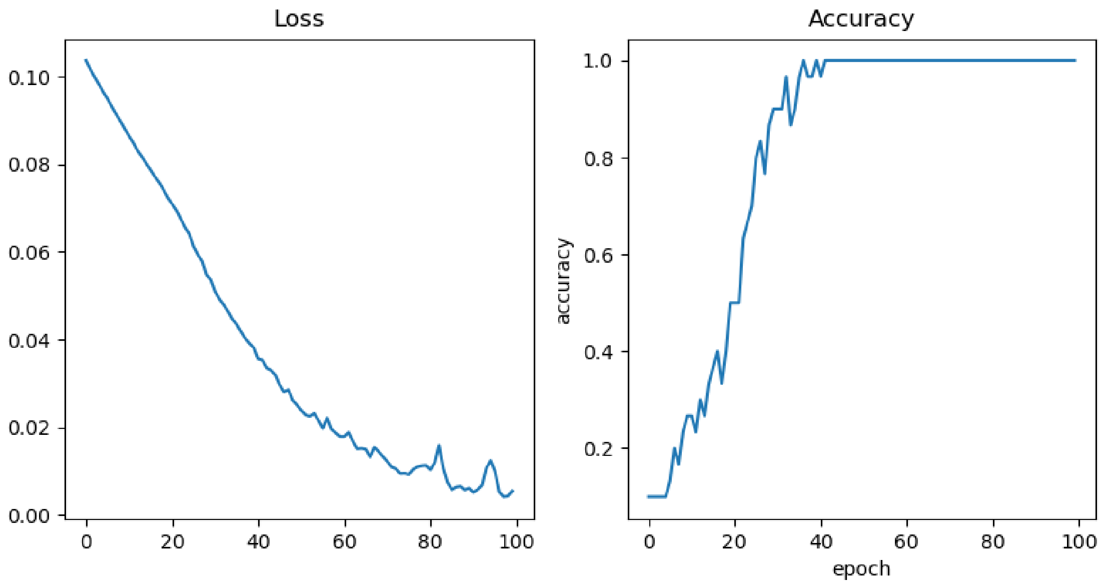
Section 2



Section 3
Using the basic model I was able to reach 100% accuracy on the 37th epoch. With a manual seed of 100, learning rate of 0.05, and a batch size of 10 the following training curves were generated.



The size of the network reported by torchsummary is 98 830 parameters which takes around 0.38 MB to store.

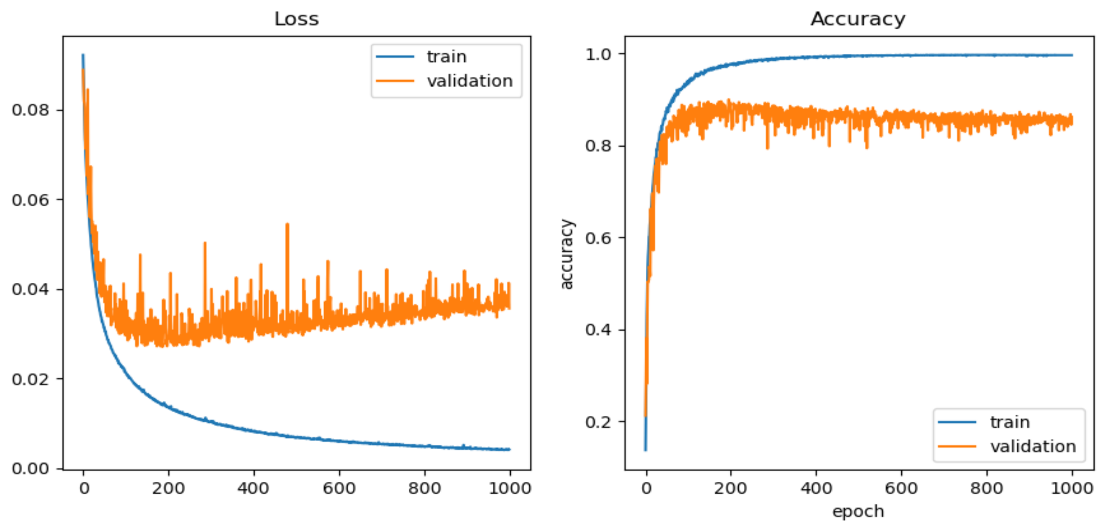
Section 4.1
1. The second option for splitting would likely yield more realistic results. The purpose of the validation set is to give an indication of how generalized the data is and how we expect it to perform “in the real world”. If this network were to be applied it would be seeing data from sources it has never seen before, i.e. from different people. Thus by splitting with the latter option we can expect the network to be able to perform on hands and situations it has never seen before.
2. I used a 70-30 train-validation split. Because we are seperating training and validation data by person, we are decreasing the amount of characteristics for training and thus need a larger amount of data. This is done in the attached `split.py`.
3. mean = tensor([0.6853, 0.6171, 0.5657])
std = tensor([0.1581, 0.1855, 0.1995]))

Section 4.2
I fixed batch size to 32 and the number of neurons on the first layer to 32 as well. This results in 12 sets of hyperparmeters to explore. I chose 32 neurons for the fully connected layer because the other option

(8) was likely too small since we are classifying 10 letters. Asking the network to condense the 10 letters into 8 neurons and then bring it back up to 10 isn't a good idea especially because this network is quite shallow. I also chose a batch size of 32 because we have 10 classes and using 4 as a batch size seemed a little small. All of these experiments are conducted with a random seed of 100. Execution time was measured as time to complete training over 1000 epochs to allow us to see the maximum accuracy some of the slower training networks can achieve.

Conv Layers	Kernels	Learning Rate	Validation Accuracy	Execution Time	Epochs	Parameters	Memory Required
1	10	0.1	73.50	1761.64	~25	233922	0.89
		0.01	73.31	1557.46	~100	233922	0.89
	30	0.1	12.75	1740.78	~5	701042	2.67
		0.01	76.77	1721.68	~100	701042	2.67
2	10	0.1	81.59	1692.04	~100	47632	0.18
		0.01	81.92	1539.61	~300	47632	0.18
	30	0.1	84.80	1881.94	~50	147572	0.56
		0.01	83.28	1814.85	~200	147572	0.56
3	10	0.1	90.03	1797.07	~125	10462	0.04
		0.01	88.94	1788.54	~950	10462	0.04
	30	0.1	88.85	1910.84	~100	41462	0.16
		0.01	88.80	1963.77	~500	41462	0.16

The training and validation losses and accuracies across the 500 epochs for 9 (best performing in this search) are shown below.



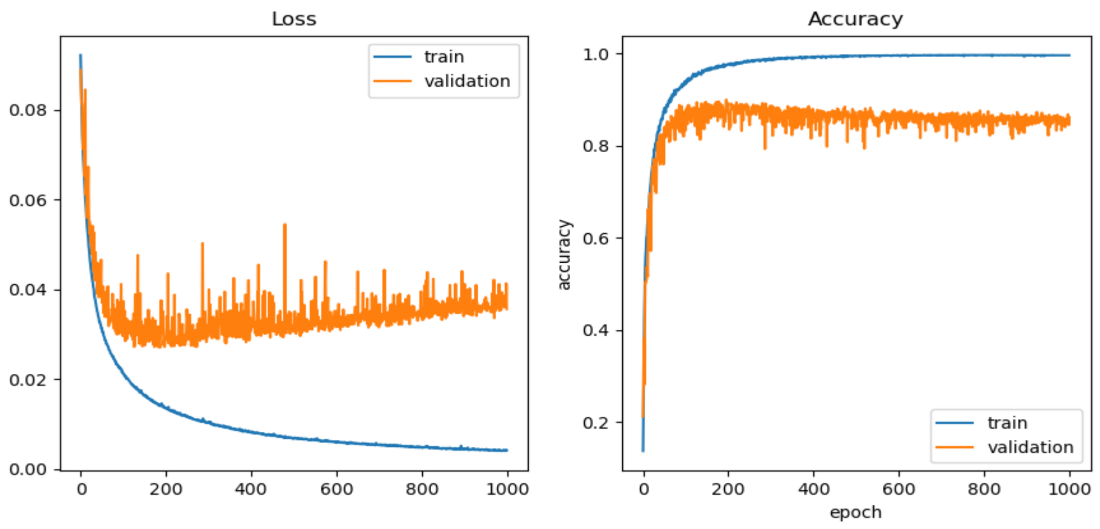
We can see that more convolutional layers with max pooling decrease the number of parameters, making the memory requirements smaller. It also appears that convolution layers take longer to train than fully connected layers. Although the networks with 3 conv layers have less parameters, they take a longer time to train than the networks with 2 conv layers. However, this extra training time appears to be worth it as we're able to get higher accuracies with 3 conv layers. In terms of learning rate, the best choice appears to depend on the specific network as for some 0.1 fails to train but for others it gives the best accuracy.

Section 5

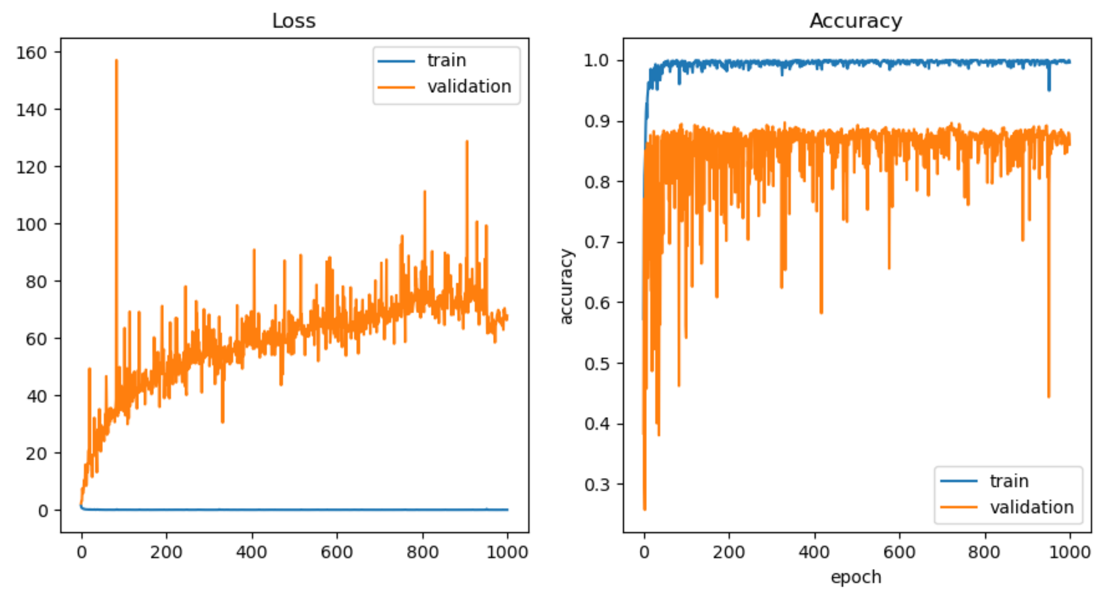
The following table shows the results of the experimentation with batch normalization and cross entropy loss. These were conducted on a model of 3 convolutional layers, each with 10 kernels, a batch size of 32, 2 linear layers with a hidden layer of 32 neurons over 1000 epochs.

Network	Validation Accuracy	Execution Time	Epochs	Parameters	Memory Required
Baseline (Section 4.2)	90.03	1797.07	~125	10462	0.04
Batch Normalization	86.23	1864.60	~30	10586	0.04
Cross Entropy Loss	87.50	1816.93	~200	10462	0.04
BN with Cross Entropy Loss	89.70	1863.24	~50	10586	0.04

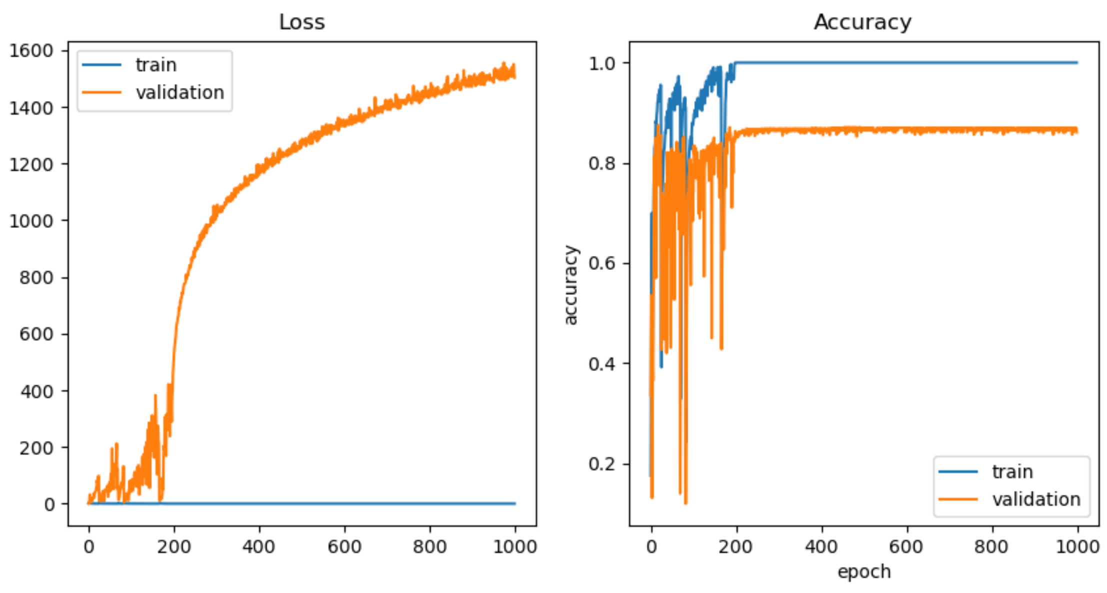
Baseline from Section 4.2:



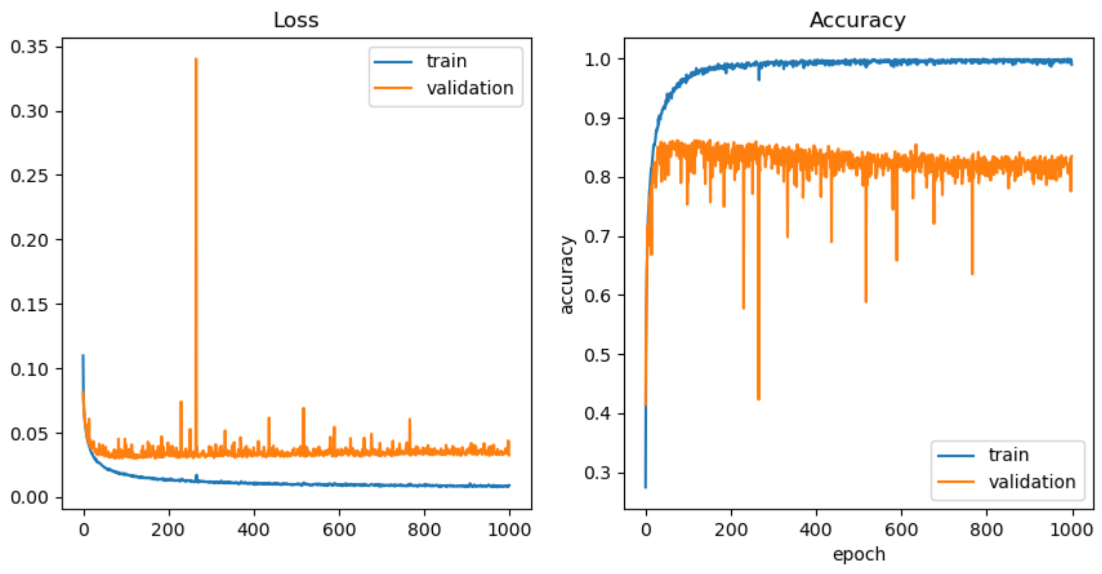
With Batch Normalization using MSELoss:



Baseline with Cross Entropy Loss:



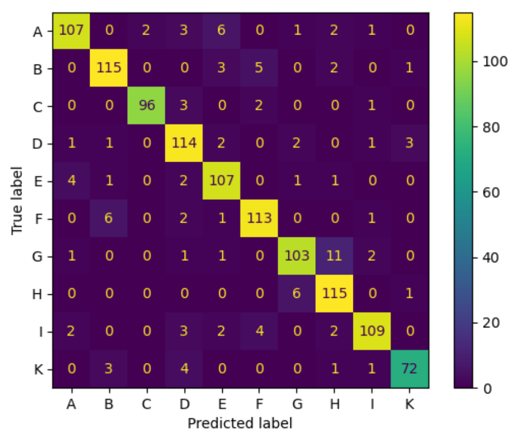
Batch Normalization with Cross Entropy Loss:



Over these 4 scenarios we can see that batch normalization appears to allow the network to reach a stable validation accuracy quicker, while using a few more parameters, and thus a longer execution time. Cross entropy loss appears to add a little bit of execution time also. It also appears that cross entropy loss was able to allow the training accuracy to hit 100% a lot earlier compared to MSELoss, although this does not always translate to higher validation accuracies, or validation accuracies becoming stable earlier.

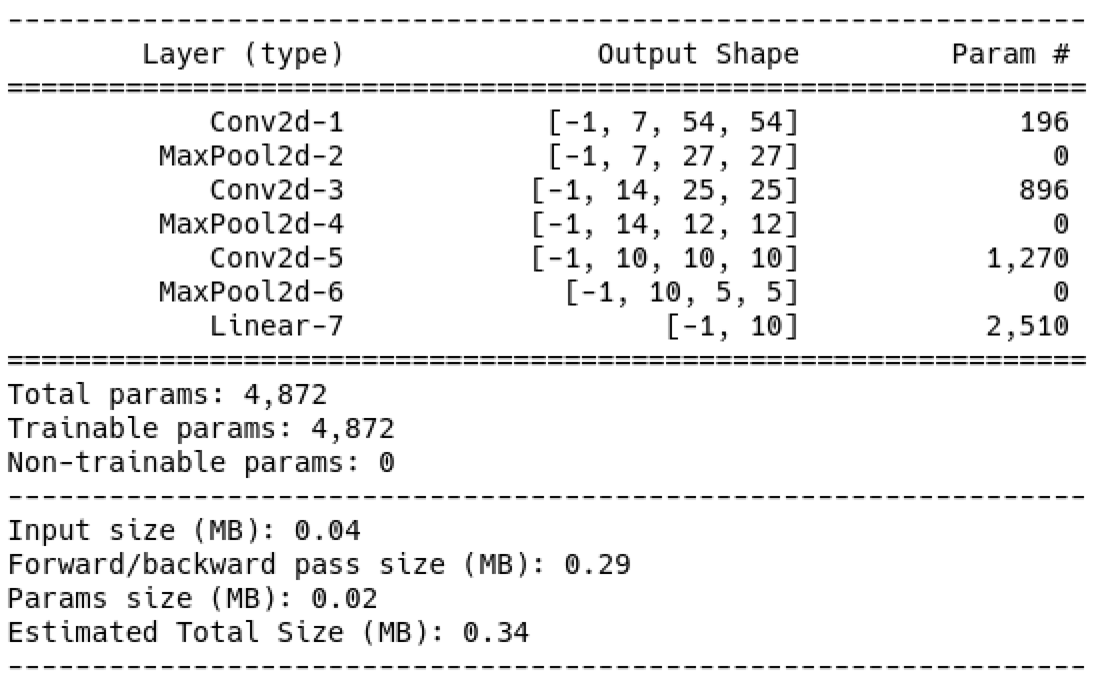
Section 6
The best network I created is named LargeModel in assign2.ipynb. It uses the SGD optimizer, optimizing the Cross Entropy Loss function over 100 epochs, a learning rate of 0.005, and a batch size of 24. A random seed of 100 was used. The network itself can be determined by looking at the LargeModel definition.

The confusion matrix is shown below. Most of the incorrect predictions were between G and H. This is reasonable since the two signs look very similar. With poor lighting and the limited resolution, it is sensible that these two are confused quite often.



	A	B	C	D	E	F	G	H	I	K
A	107	0	2	3	6	0	1	2	1	0
B	0	115	0	0	3	5	0	2	0	1
C	0	0	96	3	0	2	0	0	1	0
D	1	1	0	114	2	0	2	0	1	3
E	4	1	0	2	107	0	1	1	0	0
F	0	6	0	2	1	113	0	0	1	0
G	1	0	0	1	1	0	103	11	2	0
H	0	0	0	0	0	0	6	115	0	1
I	2	0	0	3	2	4	0	2	109	0
K	0	3	0	4	0	0	0	1	1	72

The best small network I created is named SmallModel in assign2.ipynb. It uses the SGD optimizer, optimizing the Cross Entropy Loss function over 100 epochs, a learning rate of 0.01, and a batch size of 24. A random seed of 100 was used. The summary of the network is shown below:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 7, 54, 54]             196
         MaxPool2d-2            [-1, 7, 27, 27]               0
            Conv2d-3           [-1, 14, 25, 25]             896
         MaxPool2d-4           [-1, 14, 12, 12]               0
            Conv2d-5           [-1, 10, 10, 10]           1,270
         MaxPool2d-6             [-1, 10, 5, 5]               0
            Linear-7                   [-1, 10]           2,510
================================================================
Total params: 4,872
Trainable params: 4,872
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.04
Forward/backward pass size (MB): 0.29
Params size (MB): 0.02
Estimated Total Size (MB): 0.34
----------------------------------------------------------------
```

Further details about each layer can be determined from the definition of the network in the provided files.