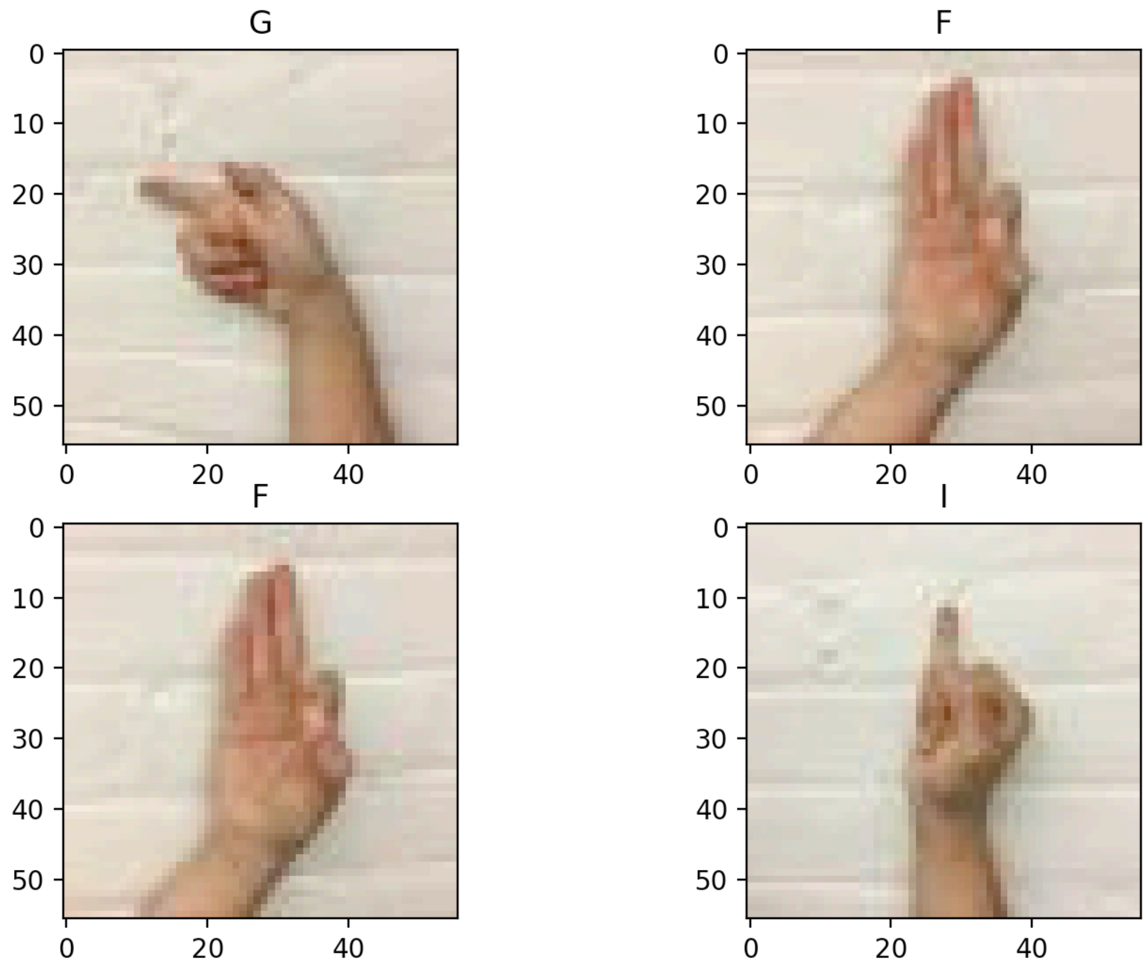
Part 2:

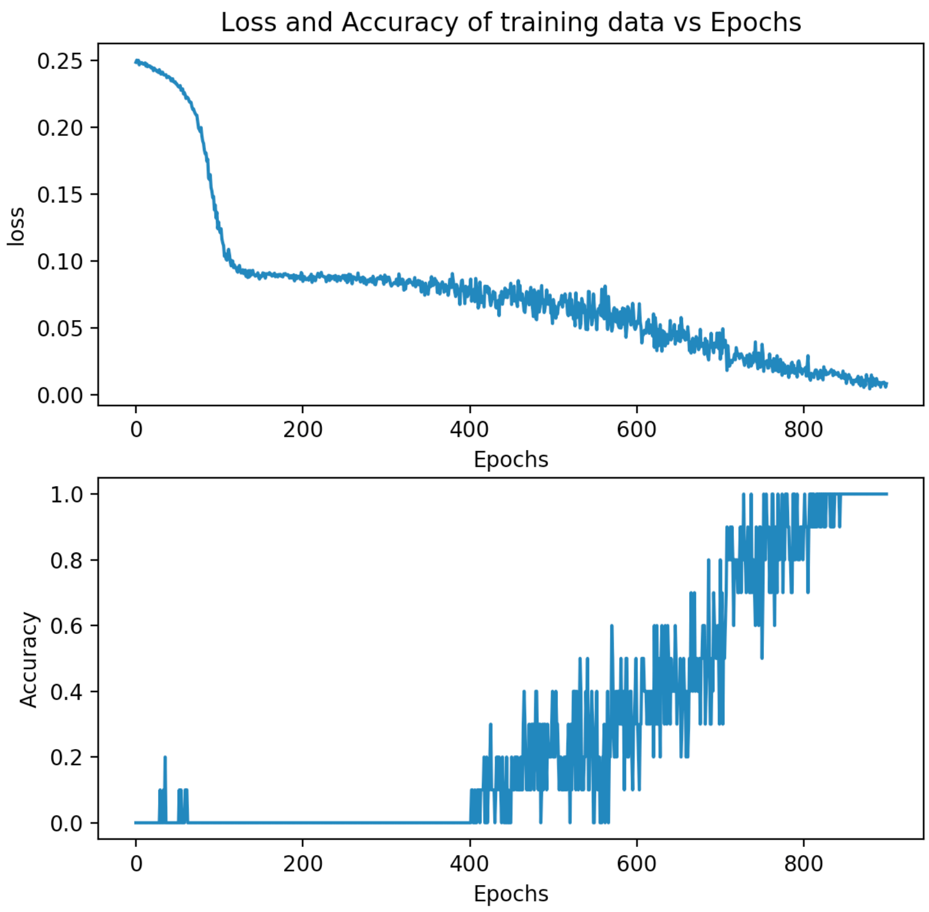
Question:

Figure 1: four images from the dataset



Part 3

Figure 2: Loss and accuracy of training data vs Epochs, with batch size = 10, Learning rate = 0.01, random seed = 2, and NN configuration as specified.



Number of Epoch:
Through experimentation, it is shown that the model only takes about 900 epochs to for the training data to reach an accuracy of 100%.

Run time:
The execution time only took 20.08 seconds.

Size of Network:
Total number of parameters = 98,830
Input size (MB) = 0.04
Forward/backward pass size (MB) = 0.15
Params size (MB) = 0.38
Estimated Total Size (MB) = 0.56

Part 4:

Question 4.1.1: Which Method is better? Why?

I think it's better to put some people's images in the validation set while putting some other's in the training set is better. This way the validation set would be a better indicator of generalizing

Question 4.1.2: What fraction of train/validation split did you use? Why?

I'm using a split of 9 to 1. This is because we only have 135 images for each gesture and having a high ratio of training to validation ensures that there is enough training data for the model.

Question 4.1.3: Report the mean and standard deviation of the image training data that you use.

The standard deviation of the image training data is

Red Channel = 36.87

Green Channel = 45.21

Blue Channel = 48.40

The mean of the image training data is:

Red Channel = 181.75

Green Channel = 160.71

Blue Channel = 144.92

Choice of Combination:

Table1: The 12 sets combination of hyperparameters used

Number	# of Conv Layers	# Kernel per conv layer	# neuron on fc1	LR	batch size
1	2	10	8	0.1	32
2	2	10	32	0.1	32
3	2	30	8	0.1	32
4	2	30	32	0.1	32
5	2	10	8	0.01	32
6	2	10	32	0.01	32
7	4	10	8	0.1	32
8	4	10	32	0.1	32
9	4	30	8	0.1	32
10	4	30	32	0.1	32
11	4	10	32	0.1	4
12	2	10	32	0.1	4

The combinations chosen are shown above.

For the **number of convolution layers**, I wanted to focus on investigate the networks with 2 and 4 layers. The reason is having one layer will only be able to extract information regarding features, but not a collection of features, which might not lead good generalization.

For the **number of kernels per convolution layers**, I'm choosing to explore all the options, however for the 4-layer convolutional layers, I'm focus on investigating the ones with 10 kernels, because having too many kernels might cause the network to over train.

For the **number of neurons on fc1**, I'm also investigating all the options, because they can have a very strong influence on the accuracy.

For **learning rate**, I'm going to focus on investigating LR = 0.1, because from part 3, it was shown that a learning rate of 0.01 will take very long to train, both in terms of epochs and time.

For **batch size** I'm running most 32 except for 2, because I'm hypothesizing that a low batch size might lead to very slow training

Comparison of Results:

Table2: Epochs and time it takes for each combination to reach an accuracy plateau, along with the number of parameter and the memory the NN occupies.

Number	Epoch	Accuracy	Time (s)	# of Param	Memory (MB)	seed
1	220	0.35096154	229.515	12,808	0.42	3
2	200	0.375	220.315	47,632	0.55	3
3	240	0.30288462	673.014	43,628	1.21	3
4	200	0.40384615	539.331	147,572	1.61	3
5	540	0.27403846	532.816	12,808	0.42	3
6	600	0.31730769	505.270	47,106	1.29	3
7	420	0.48076923	594.17287	8,228	0.42	3
8	240	0.52403846	329.790909	23,852	0.48	3
9	280	0.46634615	1088.55995	40,688	1.24	3
10	220	0.48557692	799.517552	87,032	1.42	3
11	60	0.50961538	102.708006	233,922	1.21	3
12	60	0.50961538	97.2673567	701,042	3.54	3

First, I would like to point out that I'm using the second method of splitting my training and validation data, which put some people's data in the training set and other's in the validation set. Which might be the reason why my accuracies are a lot lower than my peers' in class, however if any learning happens, it shows the NN's ability to generalize!

In this exploration of hyper-parameter, I will use combination number 2 as a bench mark and discuss the relative merit and issue with other combinations, as the second combination has an about average performance. (the second combination uses 2 convolutional layers, 32 neurons in the first fully connected layer, a batch size of 32 and a learning rate of 0.1.) I will compare other combinations with each other as well if needed.

Number 1:
For this combination, all the hyperparameters are the same as number 2, except it only has 8 neurons in the first fully connected layer instead of 32. We can see that it has achieved a slightly lower accuracy, and that the run time and number of epochs taken to reach plateau is not too different form number 2. The lower accuracy could be attributed to the fewer number of neurons in fc (fully connected) layer 1. I would hypothesize that fewer neuron would mean the information processed by the fc layers are less complex and less abstract than a fc layer with more neurons, make the NN worse at generalizing. The similar runtime can be attributed to the fact that torch use a technique called broadcasting when doing tensor operations. The runtime is dependent on the number of tensors being operated on, but not the size of the tensor. Another thing to note is that through it has ¼ of the number of parameter than number 2, it still takes 4/5 memory. This might be attributed to the way pytorch stores parameters, though I'm not sure how

pytorch does it, an inference that can be made from it is that extra neurons don't take a lot of memory.

Number 2:
This is the bench mark combination. Though it only achieved 37.5% accuracy, its way higher than 10%, which is what randomized classification would achieve.

Number 3:
The difference between this and number 2 is that it has 30 kernels in each convolution layers (number 2 has 10), and it has 8 neurons in the first fc layer (number 2 has 32). The accuracy for this model is only about 30%, which is 7% lower than number 2, but it takes 3 times the runtime to reach the plateau. The run time is likely because this model needs to train 3 times as many kernels, and that scanning is very time consuming. The lower accuracy is likely due to the fact that since this model has a lot more kernels, but fewer neurons in the first fc layers, the classifier underfits.
In terms of space, this has about the same number of parameters as number 2. However, it takes about double the memory. This is likely due to the way pytorch store information regarding convolutional layers.

Number 4:
This combination is different from number 3 by that it has more neurons in the first fc layers. It has significantly higher accuracy (40%). This is likely due to the fact that more neurons resolved the issue of underfitting.
The runtime is similar to number 3, which again demonstrated that pytorch is very efficient at handling large fc layers. On the other hand, the memory difference between number 4 and 3 is a lot bigger than the memory difference between number 2 and 1, though both differs only be the number of neurons on the fc layers. This might be due to the fact that having more kernels produces a feature map with more dimensions, and that more parameters are required for each neuron. So that increasing number of neurons have a much larger effect on the total memory required.

Number 5:
This combination is the same as number 1 except it has a learning rate that is 10 times smaller. This model only achieved an accuracy of about 27%, and it takes 3 times as much epochs to reach this plateau. The more epoch is because of the slower learning. This low accuracy might be due to the fact a lower learning rate allows the neural network to fit the training data a lot more closely, which leads to over-fitting. The memory taken is the same as number 1 as no new parameter is introduced

Number 6:
This combination is the same as number 5 except with 32 neurons on the first fc layer. We can see that the runtime is similar with number 5, and the memory usage is similar with number 1. The accuracy is about 32%, which is slightly higher than number 5, which again shows that having more neurons in the fc layer causes the network to generalize better.

Number 7:
This combination uses 4 convolution layers instead of 2. Just like number 1, it has 10 kernels, 8 neurons in the first fc layer, a learning rate of 0.1 and a batch size of 34. We can see that it has achieved an accuracy of 48%, which is a lot higher than number 1. Since this only differ from number 1 by more convolutional layers, it can be observed that more convolutional layers also lead to higher accuracy. I hypothesize this as running more convolutional, allows the kernels to pick up complex combinations of features, which lead to better generalization.
In term of run time, we can see that it takes more epochs than number 1. This can be due to the fact that there is simply more parameters to train.
In terms of memory, one thing to be observed is that this model has fewer parameter than number 1. This can be due to the fact that having more layers reduces the size of the feature map to be a lot smaller, so a lot less parameters are needed for the fc layers.

Number 8:
This model is the same as number 7 except it has 32 neurons in the first fc layer. The accuracy is at 52%, which is the highest out of all the models. The run-time is similar to number 7 and it takes a bit more memory than number 7. This again shows more neurons in the fc layer lead to higher accuracy.

Number 9:
This model has 30 kernels, while other hyperparameters is the same as number 7. It has a lower accuracy than 7, this might be due to the under-fit again. The runtime is significantly higher than number 6, 7. Which demonstrated that convolutional layers has a significant effect on run time.

Number 10:
This model is the same as number 9 except it has more neurons in the fc layer. This have improved accuracy, which can be due to under-fitting being resolved.

Number 11:
This model is the same as number 8, except it has a batch size of 4. This model trains significantly faster than number 8, taking only 20 epochs to reach 46% accuracy and 60 epochs to reach a plateau of 51% accuracy. Which shows that lower batch size can significantly improve training speed. However, each epoch is taking longer time.

Number 12:
This model is the same as number 11 but has more neurons on fc1. The result does not show significant improvement from number 11.

Key Effect of each Hyperparameters:

Number of Convolution layers:
From the combinations, it can be seen that a model with more convolution layers tend to have a higher accuracy, but also lead to significantly longer training time than adding kernels/adding neurons in fc layers.
Another thing is that having more convolution layer reduces the size of the fc layer, which might take a lot of memory.

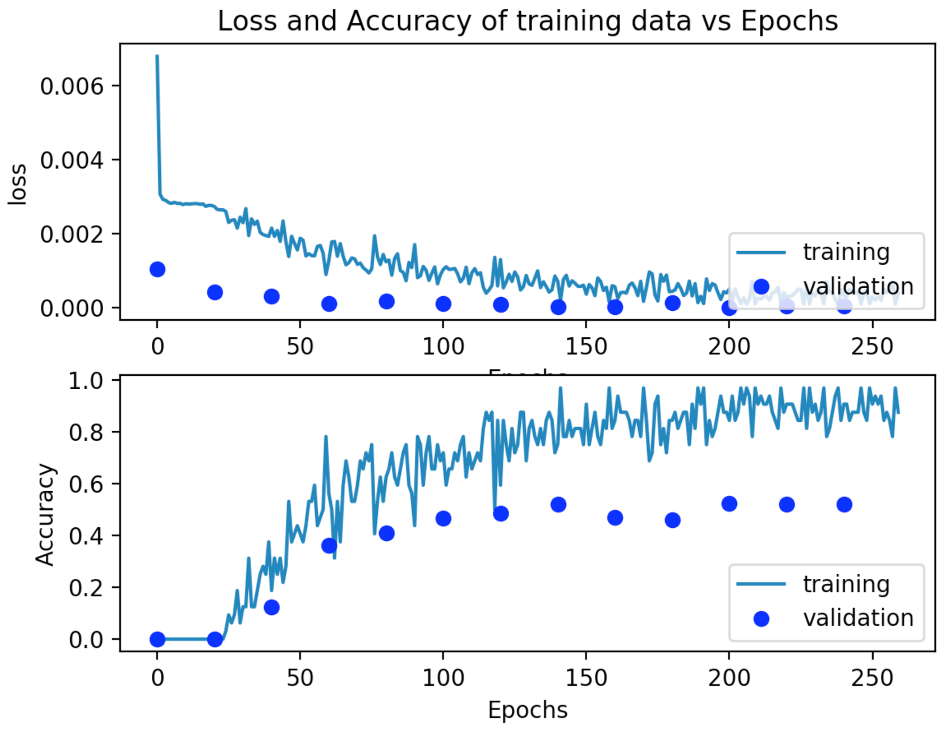
Number of kernels:
In the combinations, it can be seen that adding more kernels does not always leading to higher accuracy. Sometimes it is necessary to add more neurons in the first hidden layer as well otherwise the model might underfit.
In term of runtime, number of kernels have a moderate effect on the overall runtime, as scanning takes a lot of time.

Number of Neurons on FC1:
In the combination tested, increasing the number of neurons always tend to improve the result, and it does not have a visible effect on run time. To achieve the best result, I might increase the number of neurons on FC1 until the network over trains.

Learning Rate:
Learning rate on convolution network has a similar effect as learning rate on MLP. Having a lower learning increases the time it takes to train the model, and in this particular model, decrease the accuracy.

Batch size:
A smaller batch size significantly decreases the time it takes to train the model. However, the comparison between model 12 and 8 shows that a smaller batch size sometimes decreases the accuracy the model plateaus at. The reason might be that since the model adjust its way based on too few training examples, it not always heading at the correct gradient.

Figure 3: Loss and accuracy of training and validation data vs Epochs with combination of hyperparameter number 8.



Part 5

Table 3: Metrics when applying batch normalization and cross entropy loss to the best model:

	Epoch	Accuracy	Time (s)	# of Param	Memory (MB)	seed
BaseLine	240	0.52403846	329.790909	23852	0.48	3
With BN	80	0.3125	155.053289	23996	0.56	3
With CEL	220	0.54326923	327.209973	23852	0.48	3
BN and CEL	140	0.42788462	264.751195	23,996	0.56	3

Figure 4: Loss and accuracy of training and validation data vs Epochs with combination of hyperparameter number 8, with batch normalization

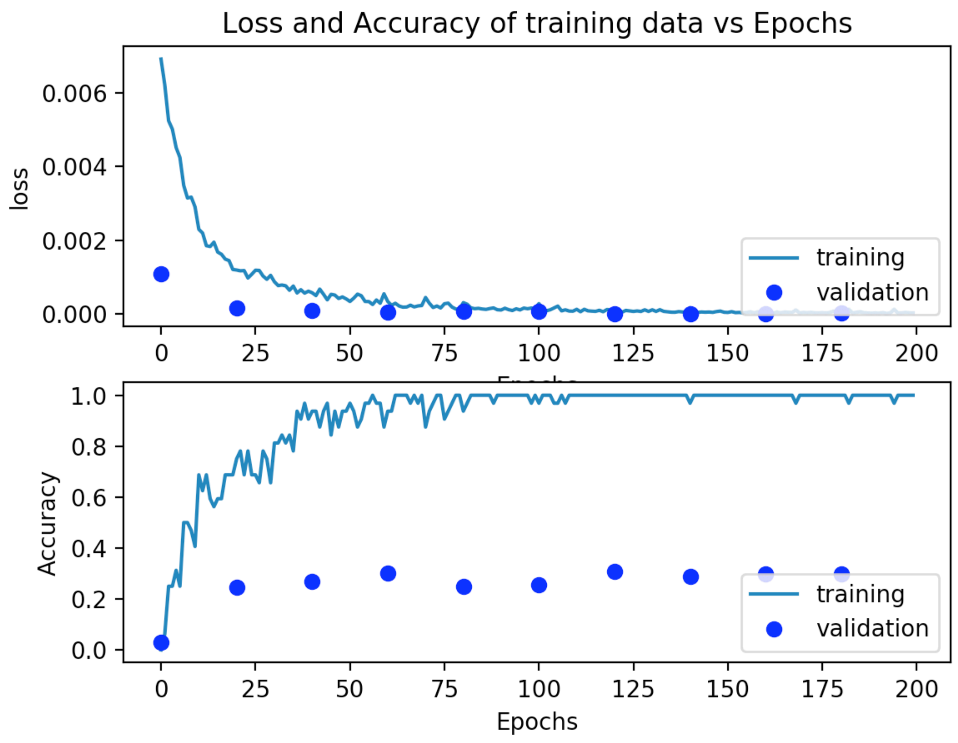


Figure 5: Loss and accuracy of training and validation data vs Epochs with combination of hyperparameter number 8, with cross entropy loss function

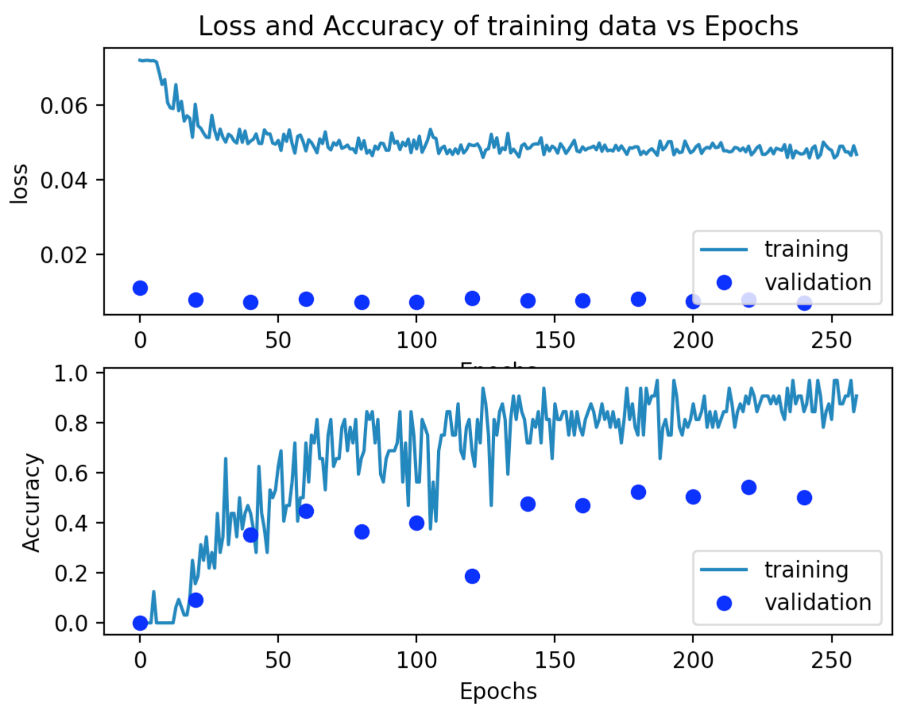
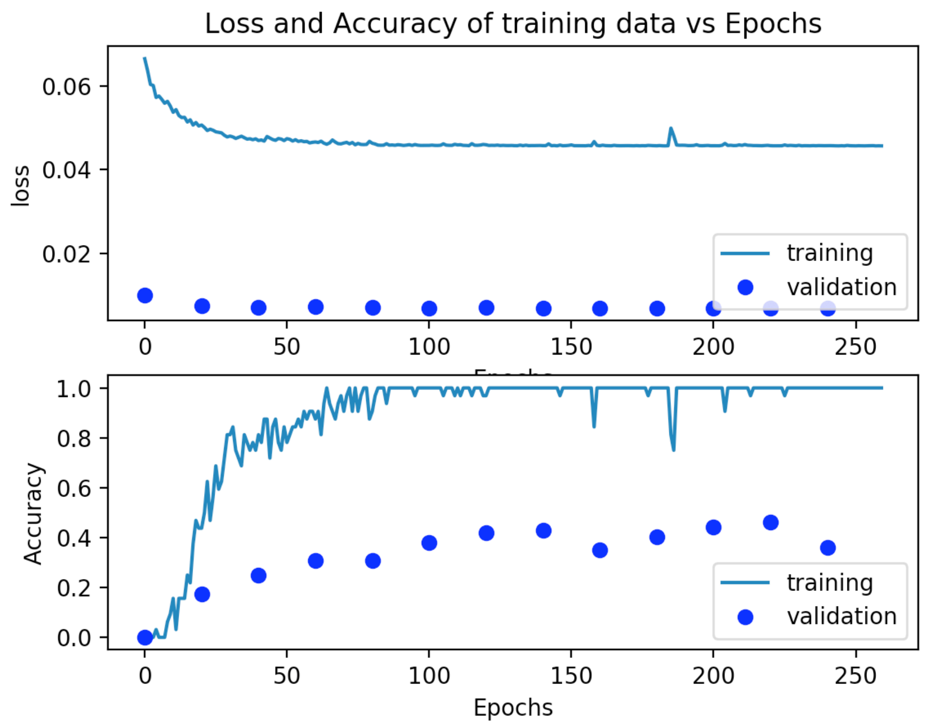


Figure 6: Loss and accuracy of training and validation data vs Epochs with combination of hyperparameter number 8, with cross entropy loss function and batch normalization



Discussion:

Batch normalization:
From the batch normalization graph, we can see that the loss of the training data decreases a lot smoother, and the training accuracy approaches 1 and oscillate a lot less when there. However, the validation accuracy is much lower than the original. This shows that though the batch normalization is able to assist the model to train, it over fit the data. It might be useful in cases where the model trains on a very large dataset, and the data it need to predict have high resemblance with elements from the training dataset.

Cross Entropy:
Comparing to the original one without modification, we can see that the validation accuracy for the cross-entropy function started increasing much faster. In the original the accuracy didn't change for the first 20 epochs, while the one for cross entropy increased to about 10% after the first 20. This is expected as cross entropy punishes high loss, which is present at the beginning of the training.
Compare to batch normalization, the addition of the cross-entropy loss function resulted in a much higher loss, and the learning also appear to be a bit slower. However, the accuracy ended to be higher than that of just using batch normalization. The high loss indicates that the model does not over train, which is probably why the model is able to achieve a higher accuracy on the validation set.

Both Cross Entropy Loss and Batch Normalization:
When both cross entropy and batch normalization are used, the training loss is kept at the same level as the model with only the cross-entropy loss, however the trend for loss is a lot smoother. Since the loss is kept relatively high, I'm guessing this prevents the model from overfitting, which potentially lead to the higher validation accuracy.
Another thing to note is that the training accuracy and validation accuracy of the model with only cross entropy separated 40 epochs later than the model with both cross-entropy loss and batch normalization. This again showed that batch normalization tends to overfit.

Best parameter:

The best parameter I was able to obtain is the one found in part 4 with the following set of parameters. Relu as the activation function, and the cross-entropy loss function. The final model reached an accuracy of 52%. The model used is:

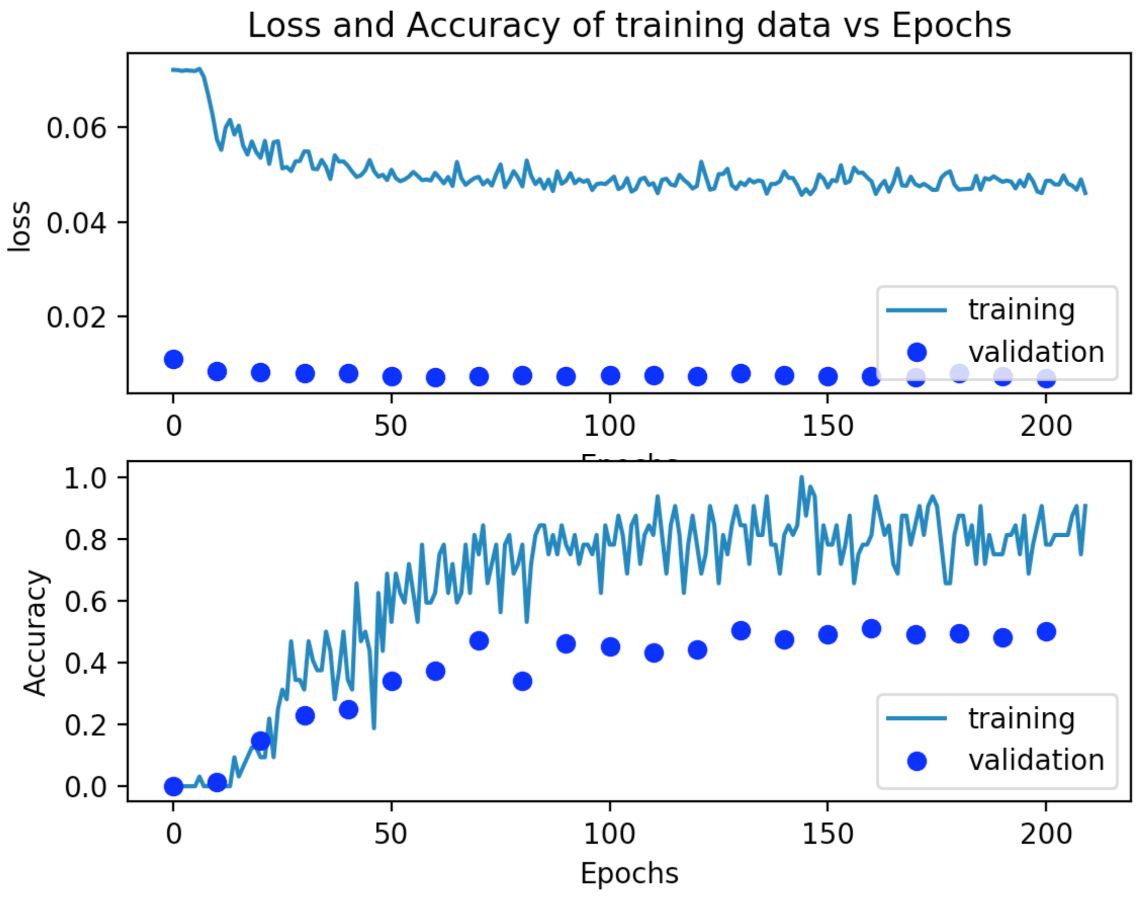
```
ConvolutionalNeuralNetwork4Layers()
```

From the model.py file

Table 4: Best set of hyperparameter

# of Conv Layers	# Kernel per conv layer	# neuron on fc1	LR	batch size	Epoch
4	10	32	0.1	32	210

Figure 7: Loss and accuracy of training and validation data vs Epochs with the best model trained.



The smallest working set of parameters is to use more max pools to reduce the size of the feature map, the hyper parameters are shown below. The model used 200 epochs to each 41% accuracy. The model used is

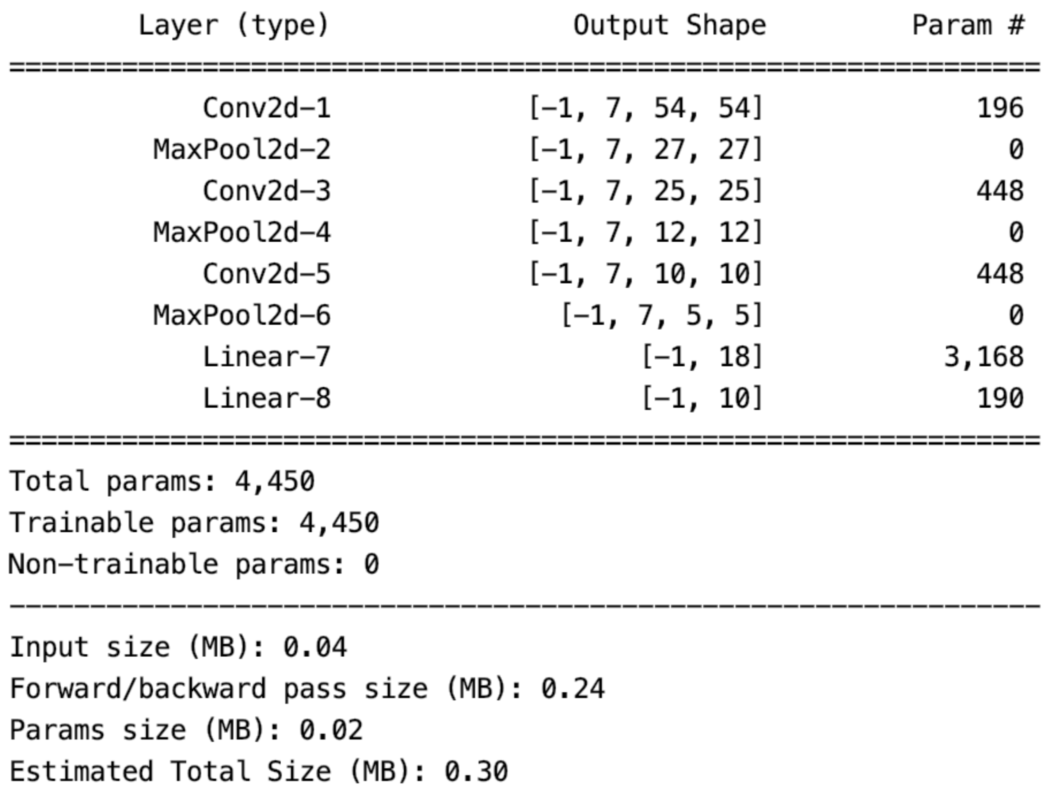
```
SmallConvolutionalNeuralNetwork()
```

From model.py

Table 4: Best set of hyperparameter for small model

# of Conv Layers	# Kernel per conv layer	# neuron on fc1	LR	batch size	Epoch
3	7	18	0.1	32	200

Figure 8: Summary of the small model.



```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 7, 54, 54]             196
         MaxPool2d-2            [-1, 7, 27, 27]               0
            Conv2d-3            [-1, 7, 25, 25]             448
         MaxPool2d-4            [-1, 7, 12, 12]               0
            Conv2d-5            [-1, 7, 10, 10]             448
         MaxPool2d-6              [-1, 7, 5, 5]               0
            Linear-7                   [-1, 18]           3,168
            Linear-8                   [-1, 10]             190
================================================================
Total params: 4,450
Trainable params: 4,450
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.04
Forward/backward pass size (MB): 0.24
Params size (MB): 0.02
Estimated Total Size (MB): 0.30
```

Figure 8: Loss and accuracy of training and validation data vs Epochs with the best small model trained.

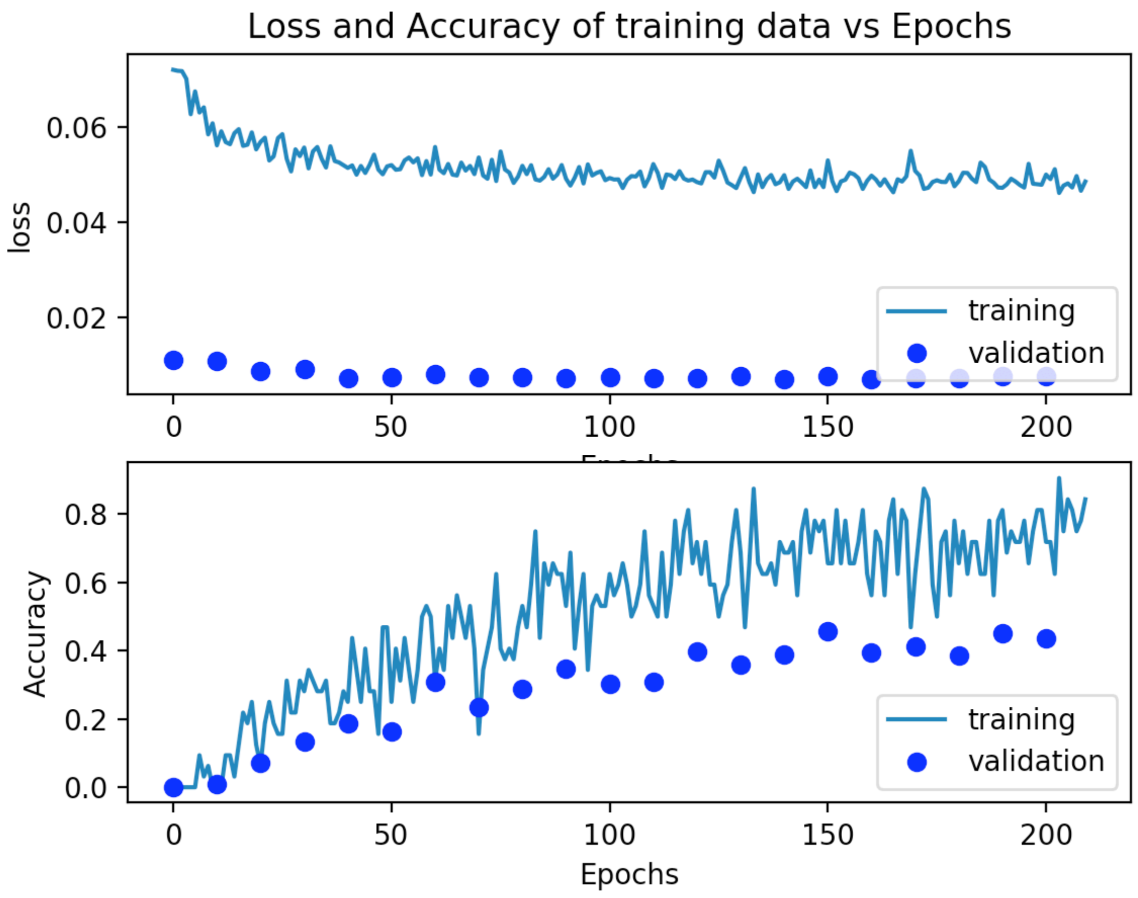
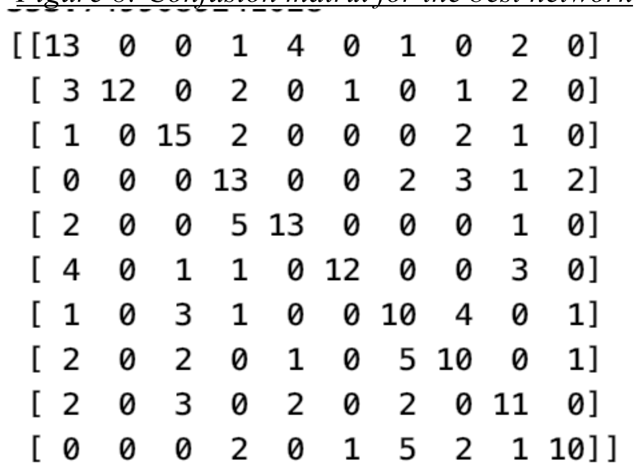
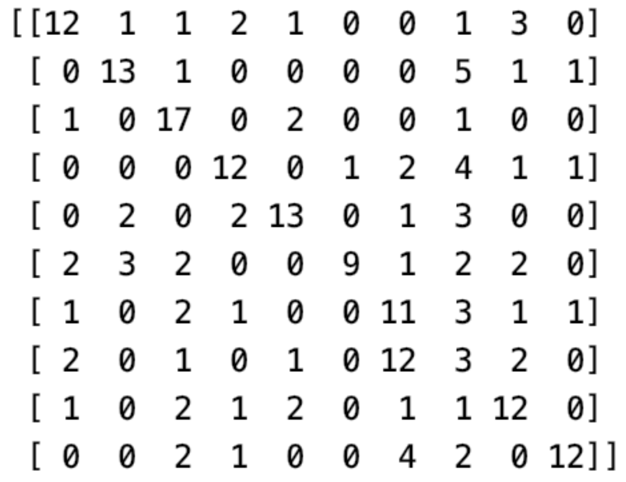


Figure 8: Confusion matrix for the best network



```
[[13  0  0  1  4  0  1  0  2  0]
 [ 3 12  0  2  0  1  0  1  2  0]
 [ 1  0 15  2  0  0  0  2  1  0]
 [ 0  0  0 13  0  0  2  3  1  2]
 [ 2  0  0  5 13  0  0  0  1  0]
 [ 4  0  1  1  0 12  0  0  3  0]
 [ 1  0  3  1  0  0 10  4  0  1]
 [ 2  0  2  0  1  0  5 10  0  1]
 [ 2  0  3  0  2  0  2  0 11  0]
 [ 0  0  0  2  0  1  5  2  1 10]]
```

Figure 9: Confusion matrix for the best small network



```
[[12  1  1  2  1  0  0  1  3  0]
 [ 0 13  1  0  0  0  0  5  1  1]
 [ 1  0 17  0  2  0  0  1  0  0]
 [ 0  0  0 12  0  1  2  4  1  1]
 [ 0  2  0  2 13  0  1  3  0  0]
 [ 2  3  2  0  0  9  1  2  2  0]
 [ 1  0  2  1  0  0 11  3  1  1]
 [ 2  0  1  0  1  0 12  3  2  0]
 [ 1  0  2  1  2  0  1  1 12  0]
 [ 0  0  2  1  0  0  4  2  0 12]]
```

In the confusion matrix, we can see that the large CNN often yield false positive for letter A, C, D and I. Which might be due to the fact the 4 looked quick similar, as the sign for these all involve a closed fist. Therefore, the network probably learned to recognize closed fist and is confused between all of these. We can also see that the large CNN have a lot of false negative at these letters as well. From this observation I would hypothesize that a high false positive at certain classes would often correlate to high false negative for those classes as well.

The small CNN has a lot of confusion at A, C, H and I. Which again might have to due to the fact that these letter has a sign of a closed fist.