Report

Part 0: Imports and Basic Setup (5 Points)

Nothing to report for this part. You will be just scored for finishing the setup.

Part 1: Fully connected neural networks (25 Points)

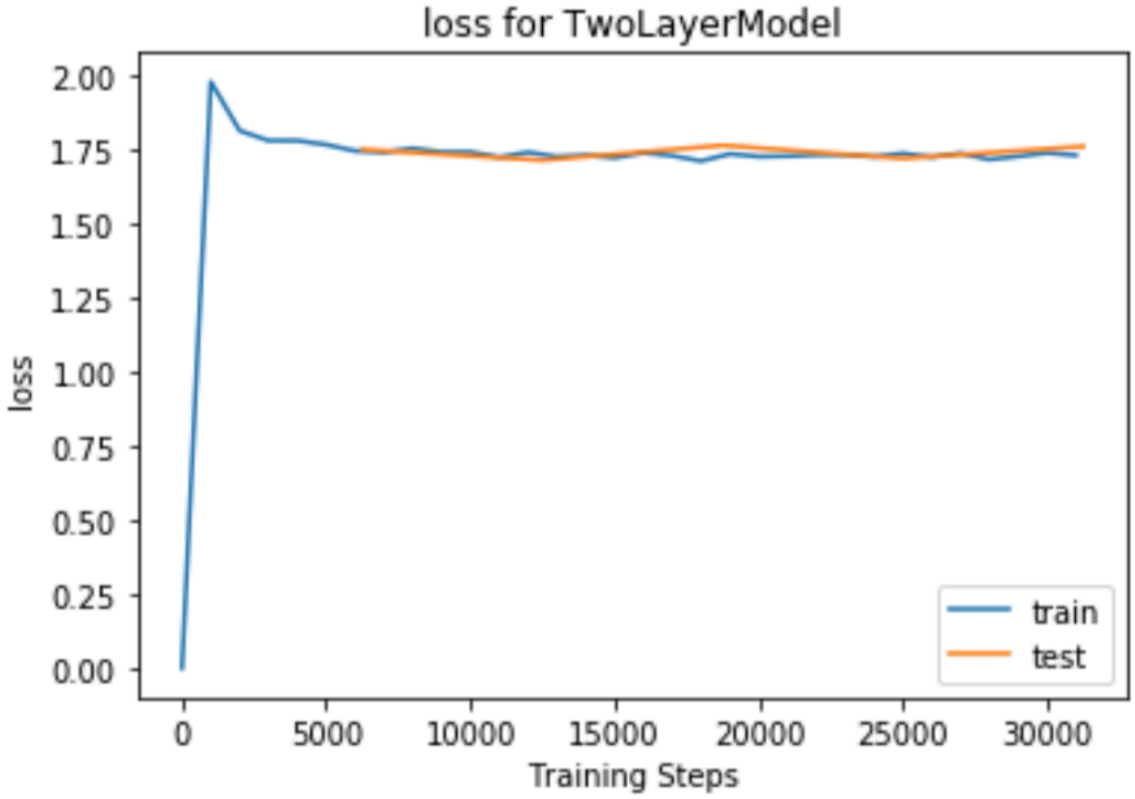
Test (on validation set) accuracy (5 Points): 0.365100

Test loss (5 Points): 1.762166

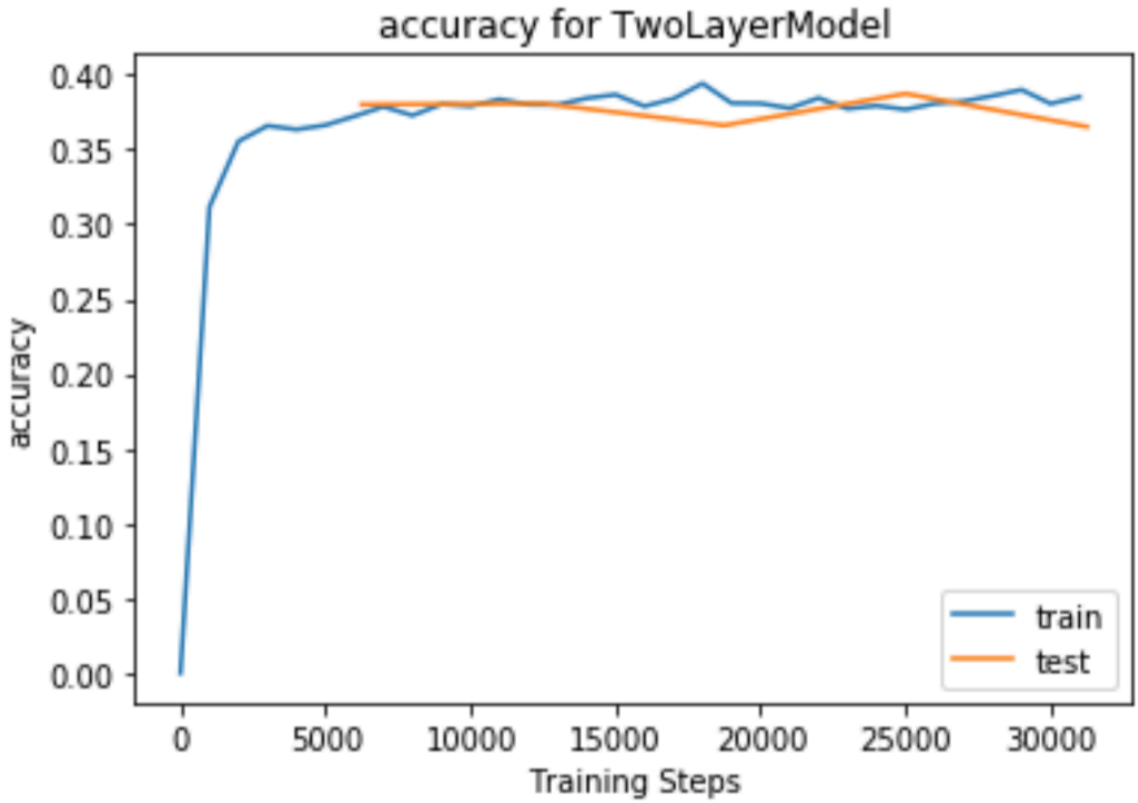
Training time (5 Points): 85.2s

Plots:

- Plot a graph of accuracy on validation set vs training steps (5 Points)



- Plot a graph of loss on validation set vs training steps (5 Points)



Part 2: Convolution Network (Basic) (35 Points)

Tensor dimensions: A good way to debug your network for size mismatches is to print the dimension of output after every layers:

(10 Points)

Output dimension after 1st conv layer: (8, 16, 32,32)

Output dimension after 1st max pooling: (8, 16, 16, 16)

Output dimension after 2nd conv layer: (8, 16, 16, 16)

Output dimension after flatten layer: (8, 4096)

Output dimension after 1st fully connected layer: (8, 64)

Output dimension after 2nd fully connected layer: (8, 10)

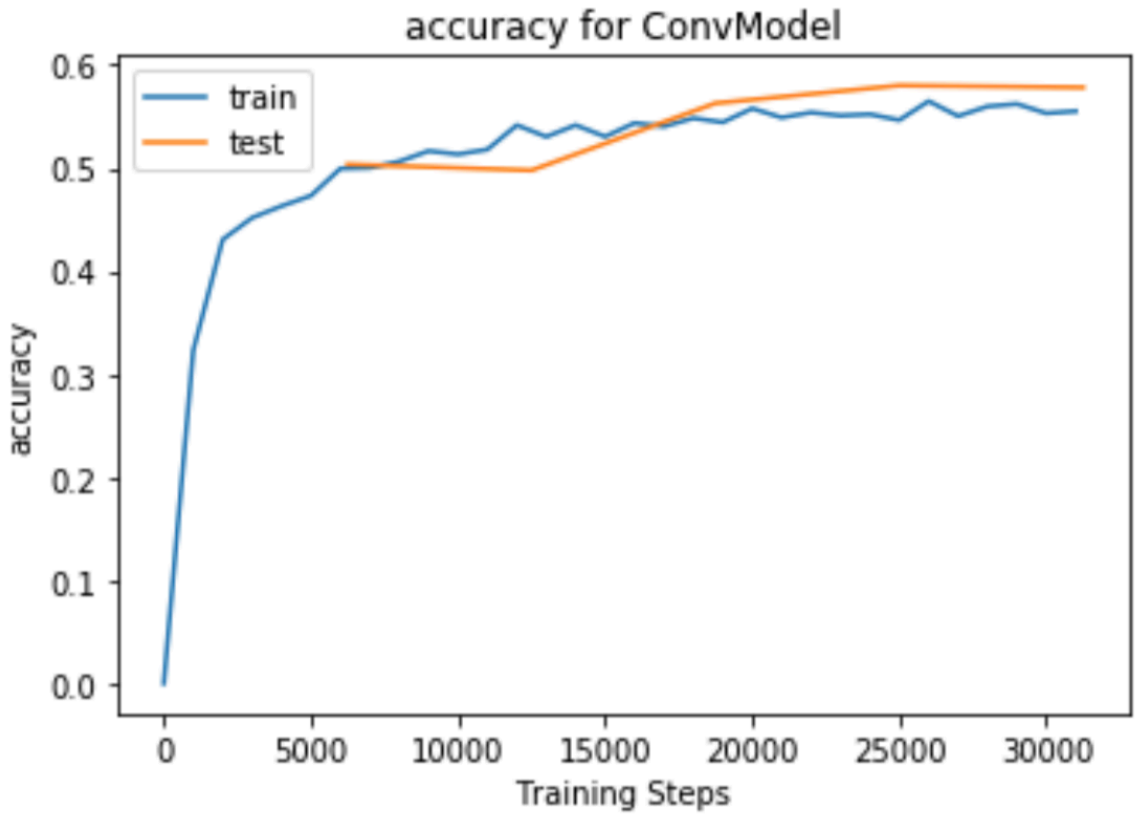
Test (on validation set) Accuracy (5 Points): 0.578300

Test loss (5 Points): 1.188132

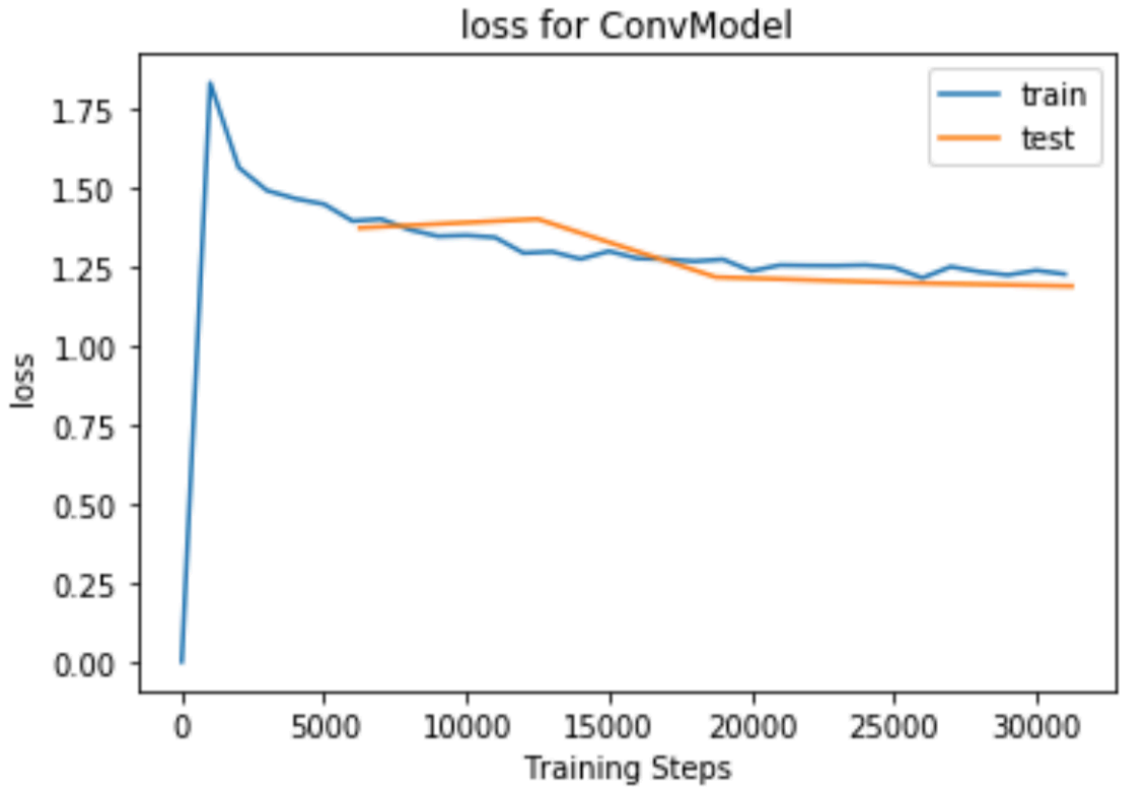
Training time (5 Points): 123.4s

Plots:

- Plot a graph of accuracy on validation set vs training steps (5 Points)



- Plot a graph of loss on validation set vs training steps (5 Points)



Part 3: Convolution Network (Add one or more suggested changes) (35 Points)

Describe the additional changes implemented, your intuition for as to why it works, you may also describe other approaches you experimented with (10 Points):

- Dropout layer

  I add a dropout layer between the two fully connected layers to avoid overfitting. As a result, though the training accuracy of each epoch decreased, the testing accuracy increased, which means it works well. I tried a few parameters of dropout function and 0.3 gives the best result. I think the dropout layer works because in the former model, training accuracy is higher than testing accuracy, which means the model tend to be a little overfitting.

- Batch Normalization

  I add a batch normalization layer at the beginning of the model to help the model converge. But it only helped a little and I think the reason might me the number of layers is not that large so the influence of batch normalization is not very obvious.

- More layers

  I tried to add more than one convolutional layers because more layers mean we could catch more details of the dataset and make the model more accurate, but after attempting, only adding one more convolutional layer gives the best result.

- Change layer size

  I tried some large output channels for convolutional layers like 100, but it did not help. So finally I just did small changes.

- Pooling layers, stride

  I add 2 more pooling layers using the max pooling principle, and tried different strides but 1 still gives the best result.

- Different optimizer

  I tried optim.Adam and optim.SGD with different parameters and different learning rate. SGD with learning rate equaling 0.001 gives the best result. And when I set the learning rate as 0.01, the model result is really bad, though the accuracy increased for a while at the beginning, it soon went down and stop at around 0.3.

- Train for longer

  I use 10 epochs instead of only training 5 times, which increased the accuracy. It helps because longer training gives the model more chances to optimizing itself and find the best parameters, which could give us a better result.

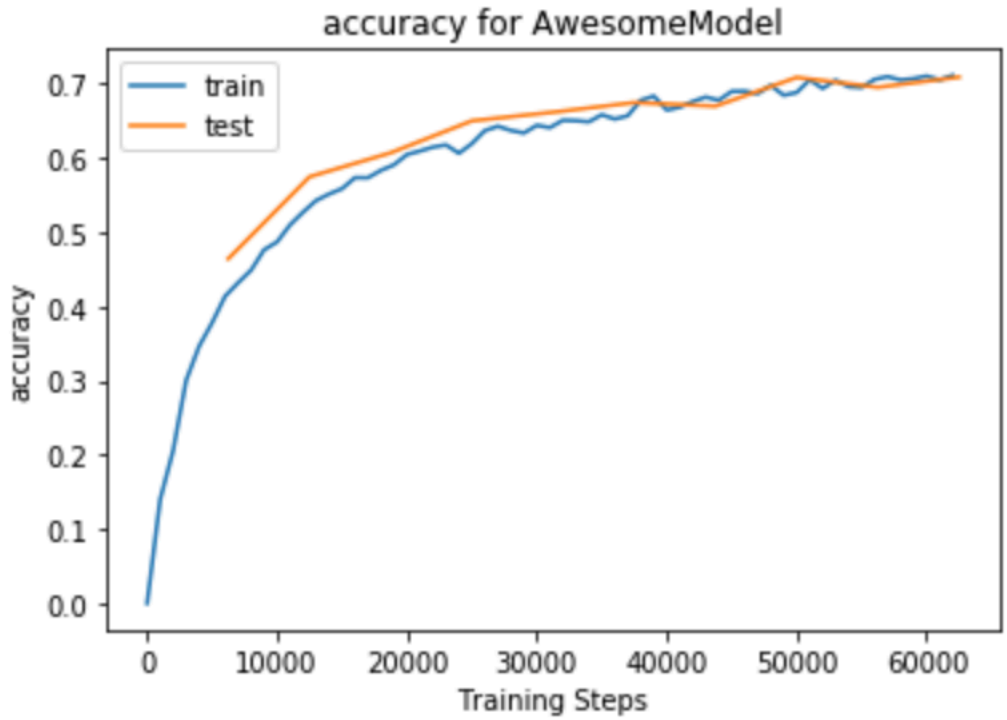
Test (on validation set) Accuracy (5 Points): 0.709100

Test loss (5 Points): 0.835373

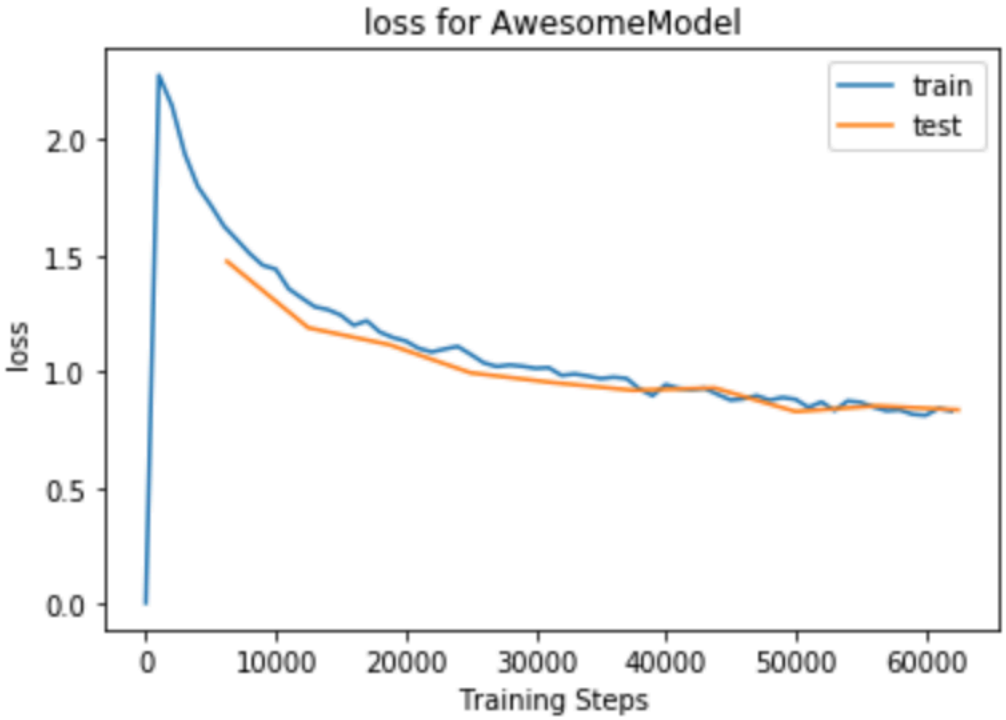
Training time (5 Points): 274.0s

Plots:

- Plot a graph of accuracy on validation set vs training steps (5 Points)



- Plot a graph of loss on validation set vs training steps (5 Points)



10 bonus points will be awarded to top 3 scorers on leaderboard (in case of tie for 3rd position everyone tied for 3rd position will get the bonus)

Result: 72.4 / 100.0