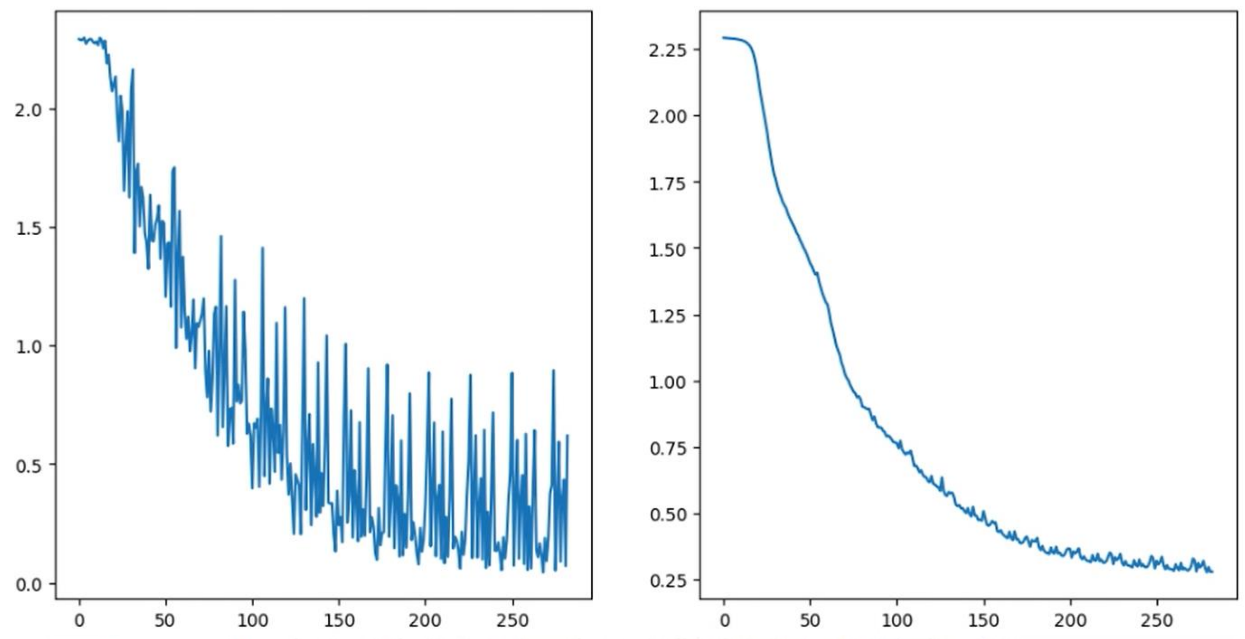
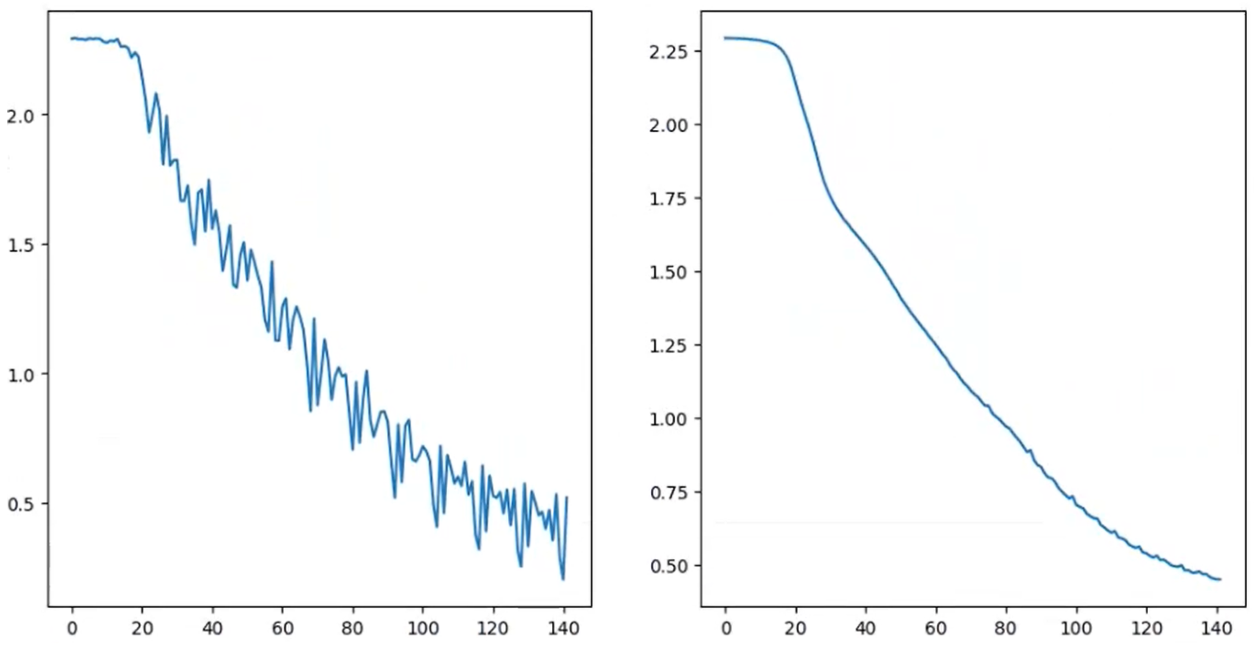
We ran our network on 6 different batch sizes ([16,32,64,128,256,512]) with the same parameters which were (layers = [784, 20, 7, 5, 10], learning rate = 0.009, num epochs = 100, use batchnorm = False) and got the following results:

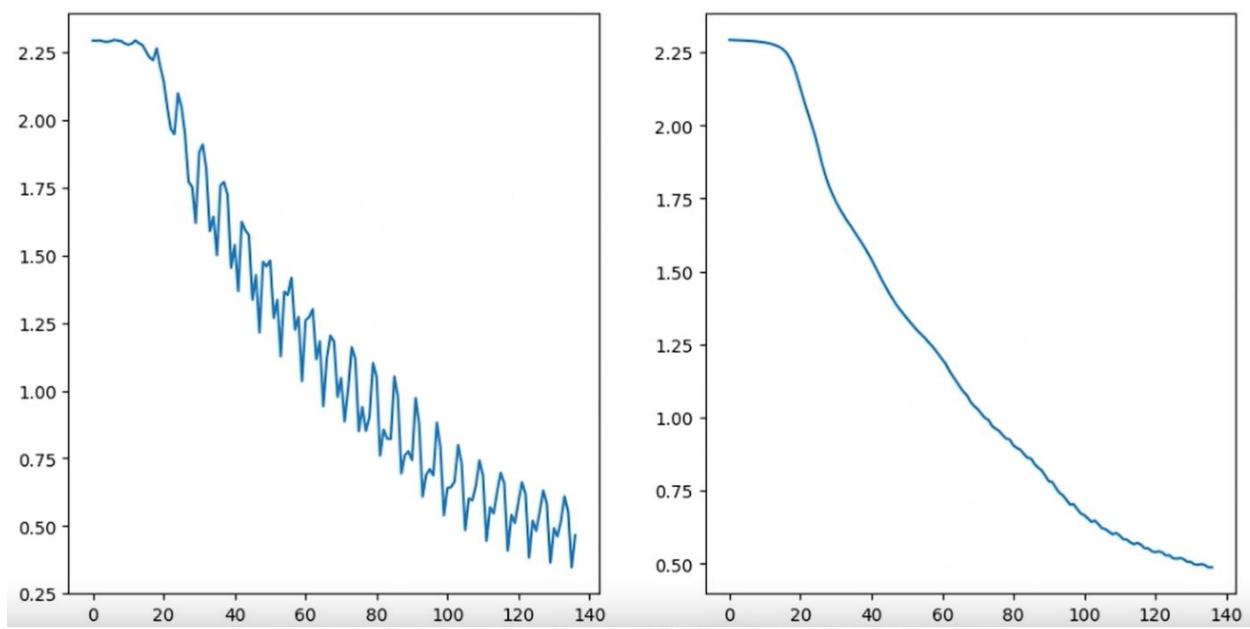
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 16



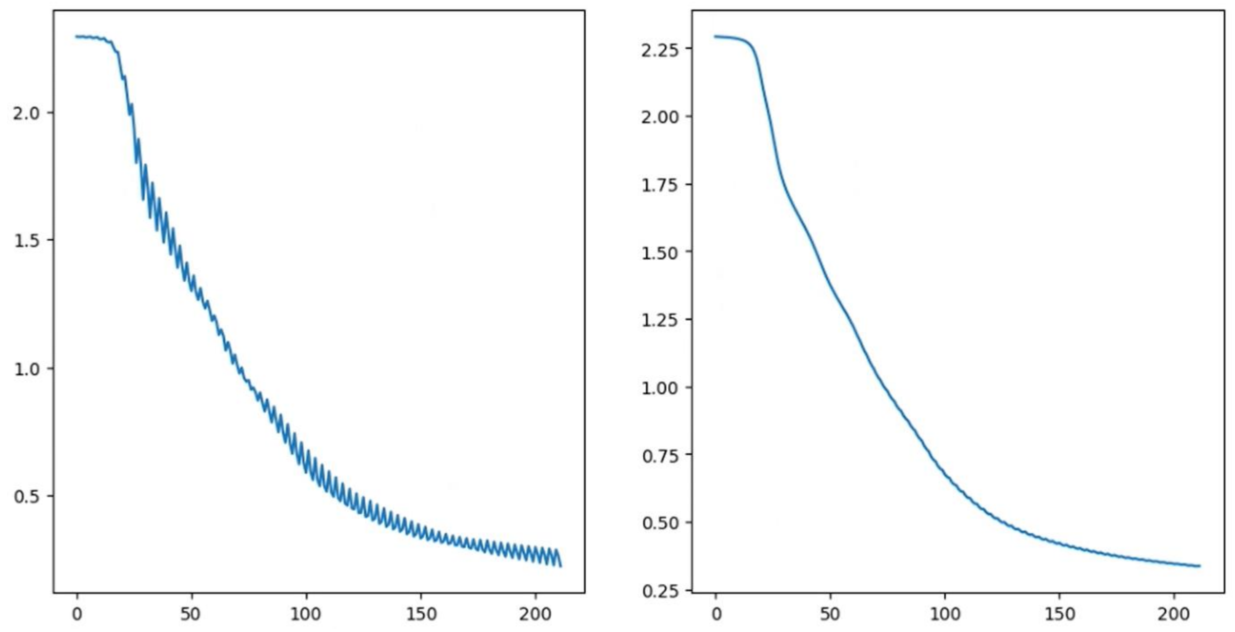
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 32



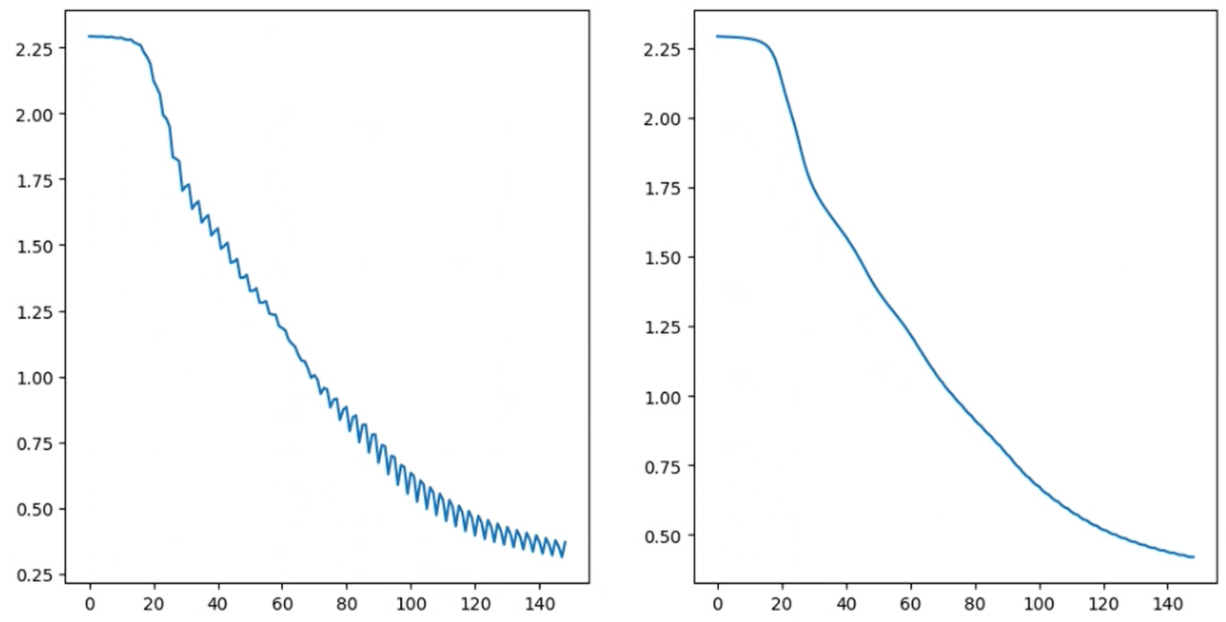
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 64



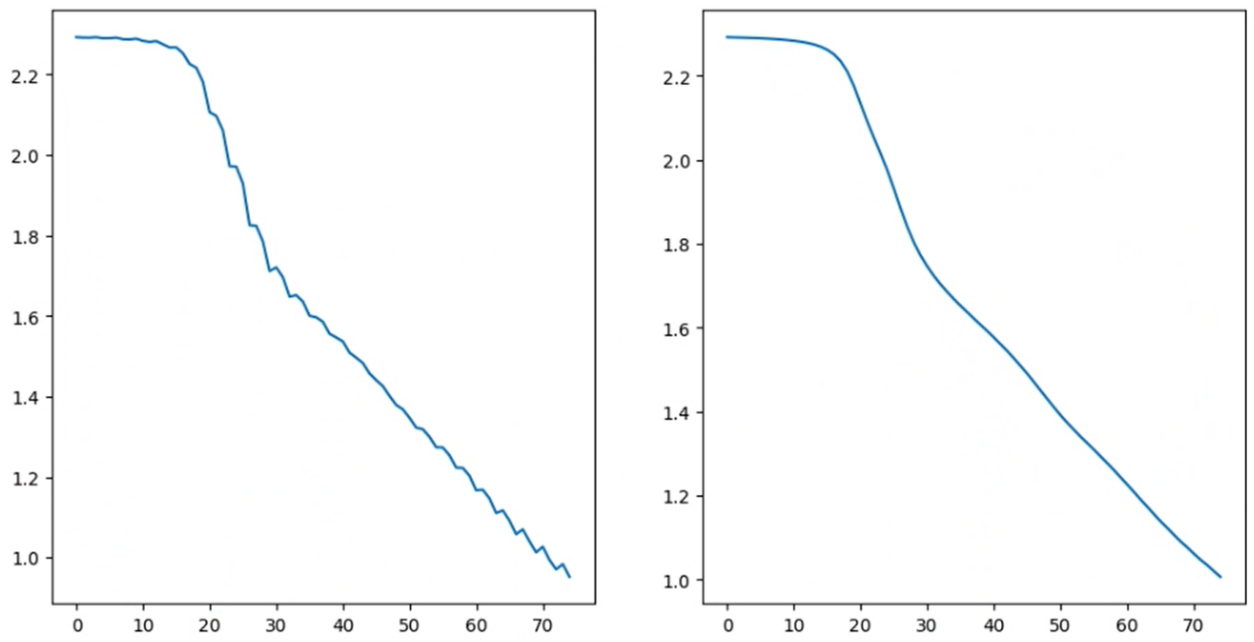
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 128



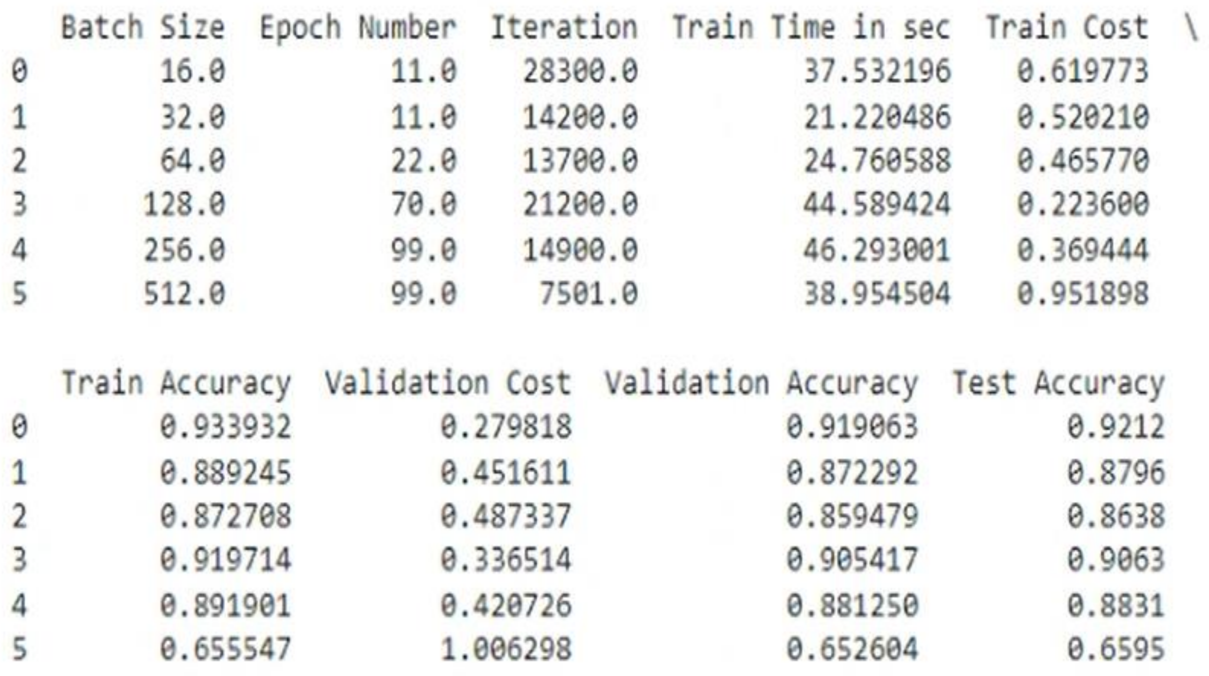
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 256



Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 512



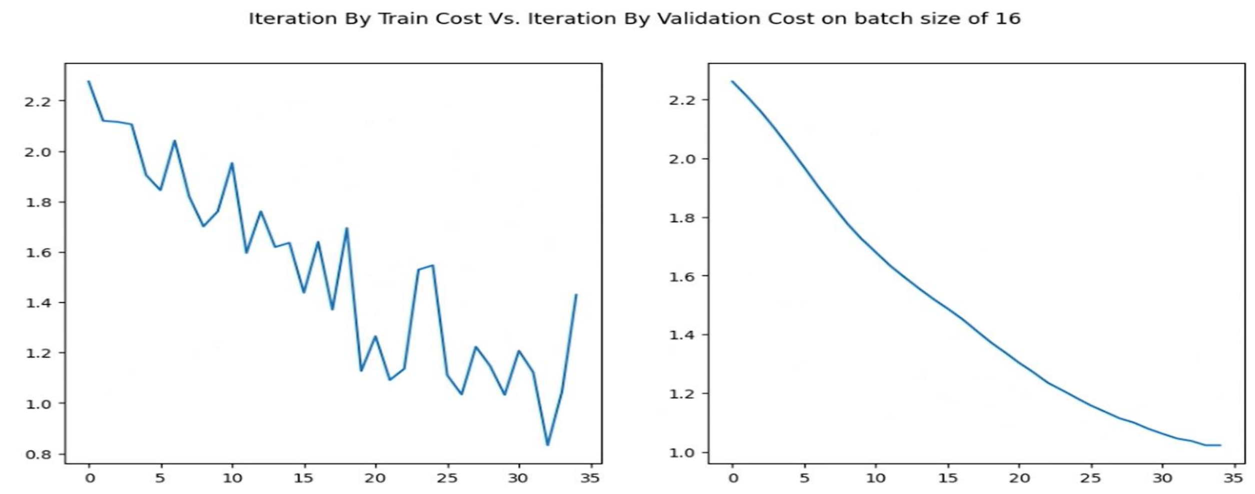
Summary:

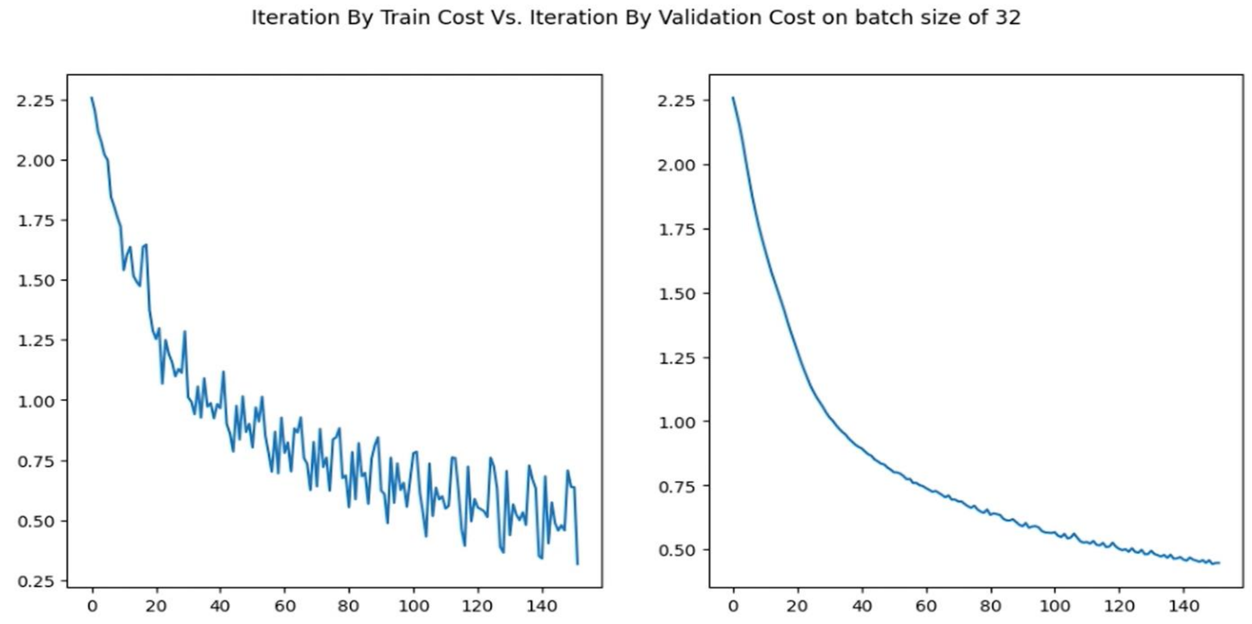


```
   Batch Size  Epoch Number  Iteration  Train Time in sec  Train Cost  \
0        16.0          11.0    28300.0          37.532196    0.619773
1        32.0          11.0    14200.0          21.220486    0.520210
2        64.0          22.0    13700.0          24.760588    0.465770
3       128.0          70.0    21200.0          44.589424    0.223600
4       256.0          99.0    14900.0          46.293001    0.369444
5       512.0          99.0     7501.0          38.954504    0.951898

   Train Accuracy  Validation Cost  Validation Accuracy  Test Accuracy
0        0.933932         0.279818             0.919063         0.9212
1        0.889245         0.451611             0.872292         0.8796
2        0.872708         0.487337             0.859479         0.8638
3        0.919714         0.336514             0.905417         0.9063
4        0.891901         0.420726             0.881250         0.8831
5        0.655547         1.006298             0.652604         0.6595
```

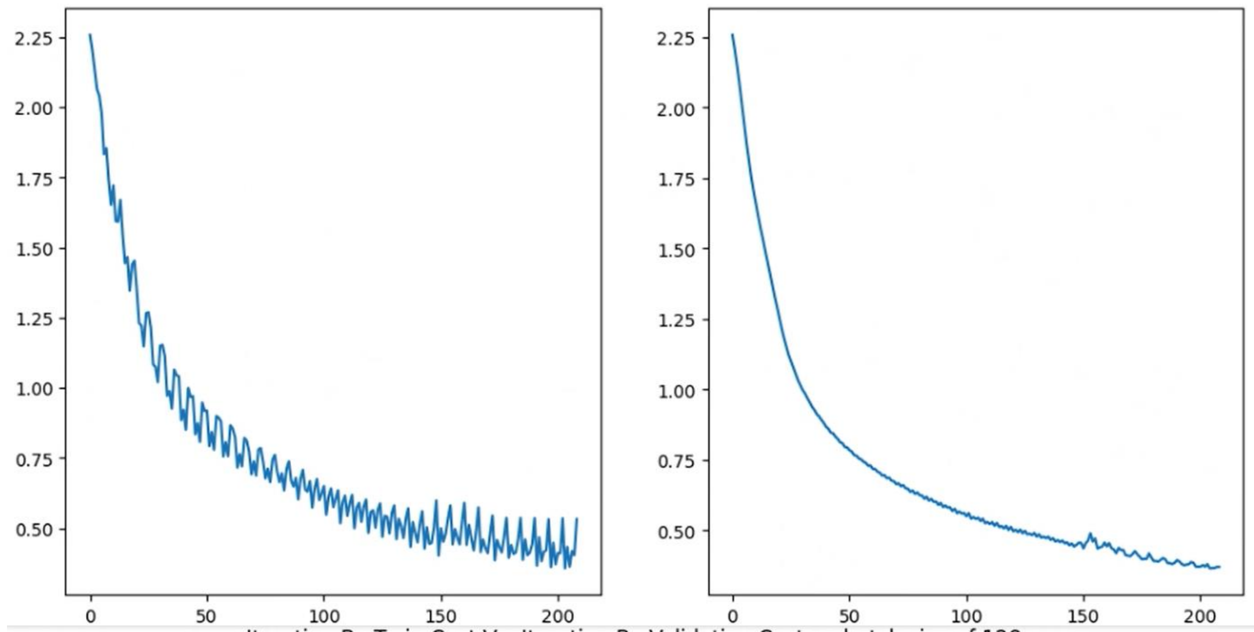
From the following table we found that the best batch size is 16, it received the highest test accuracy of 0.9212 and a runtime of 37.5 sec (note: if we wanted value to runtime more highly, we would choose batch size 32, since it got a similar test accuracy of ~ 0.88 with almost half the running time of 16).
Across all batch sizes, we can see from the graphs that our network isn’t bias to overfitting, but is learning the images in an appropriate way considering the overall accuracies displayed in the table above, we may attribute these accuracy numbers to the good initialization of the weights.

now with batch norm:
We ran our network on 6 different batch sizes ([16,32,64,128,256,512]) with the same parameters which were (layers = [784, 20, 7, 5, 10], learning rate = 0.009, num epochs = 100, use batchnorm = True) and got the following results:

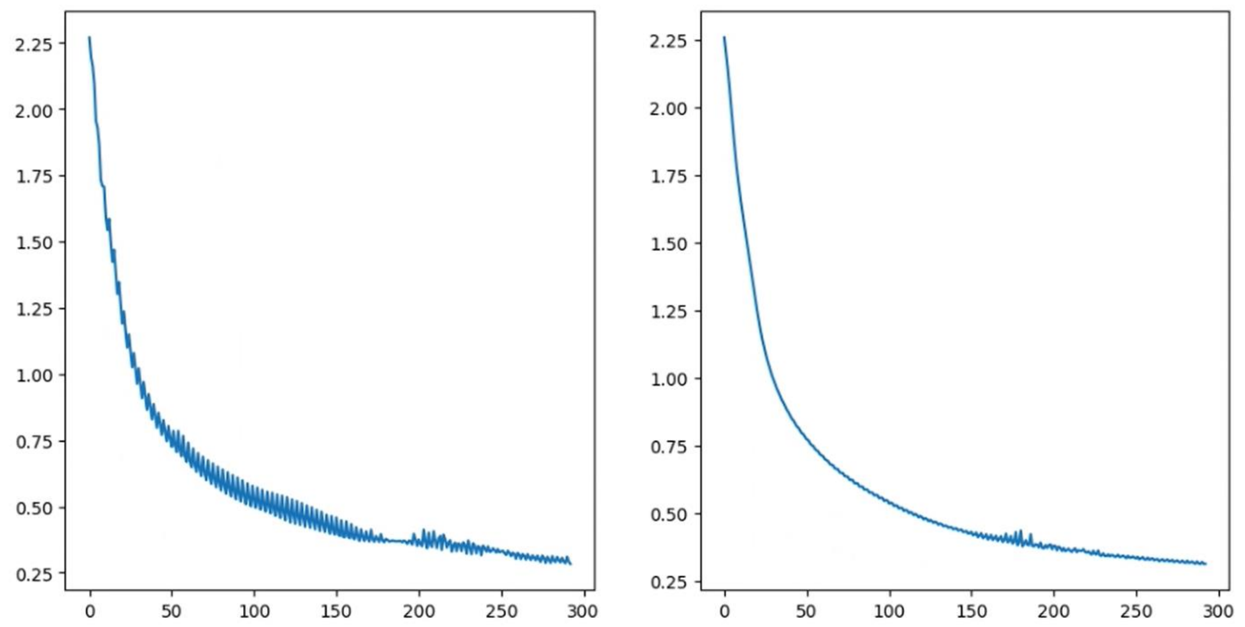




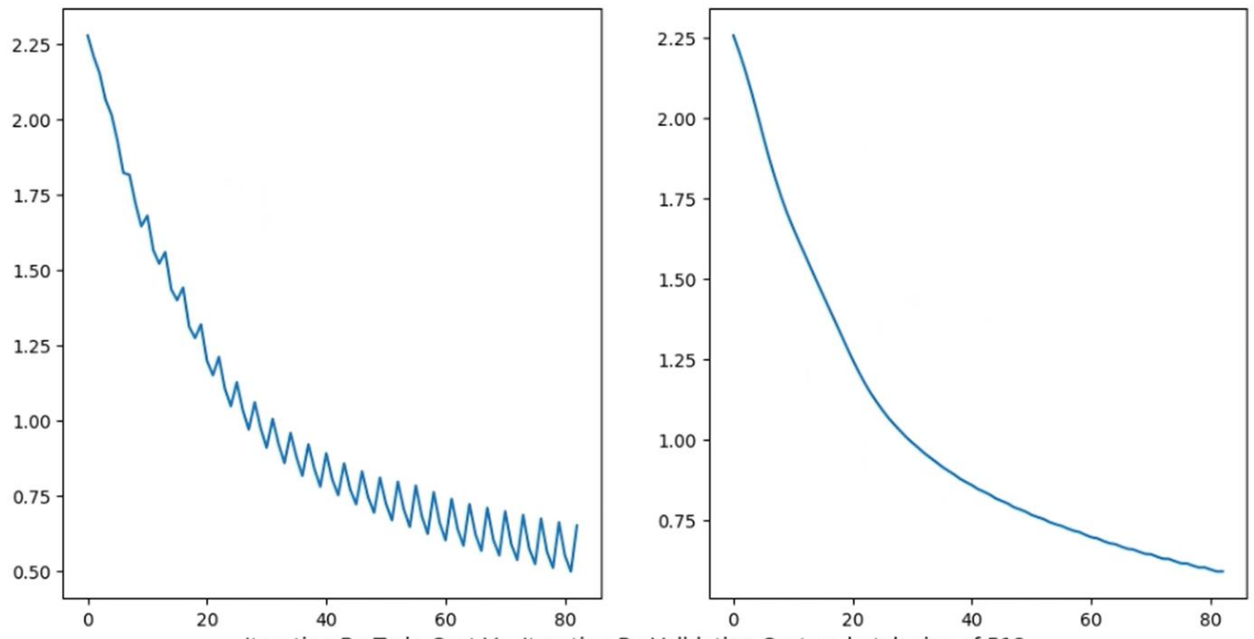
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 64



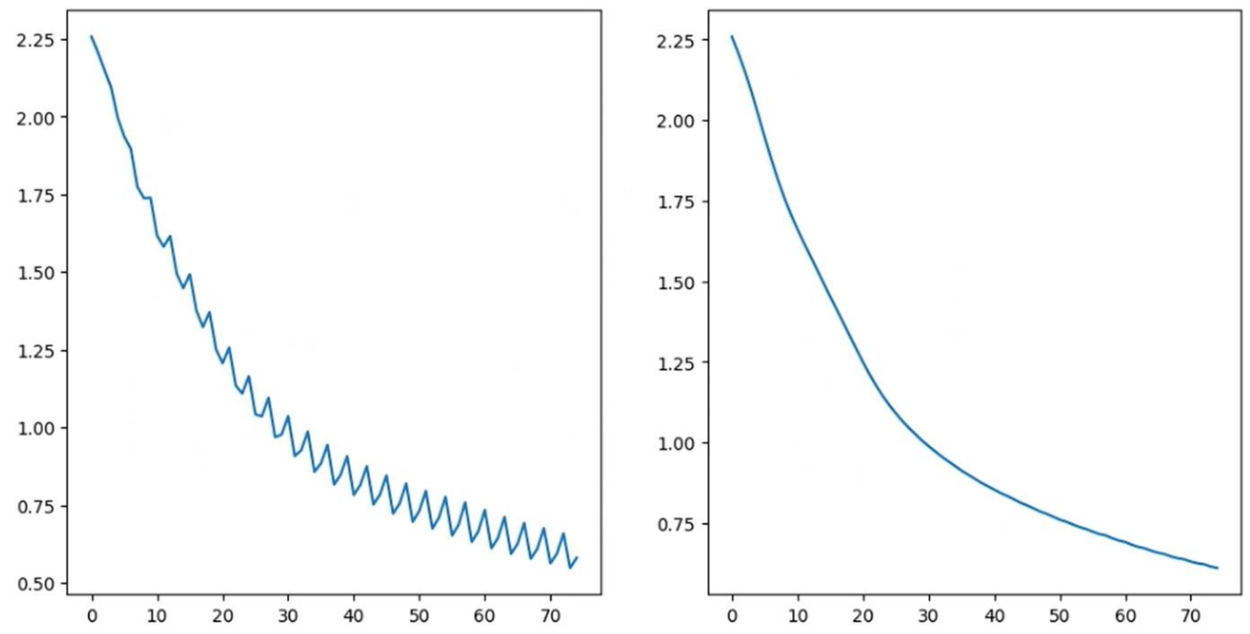
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 128



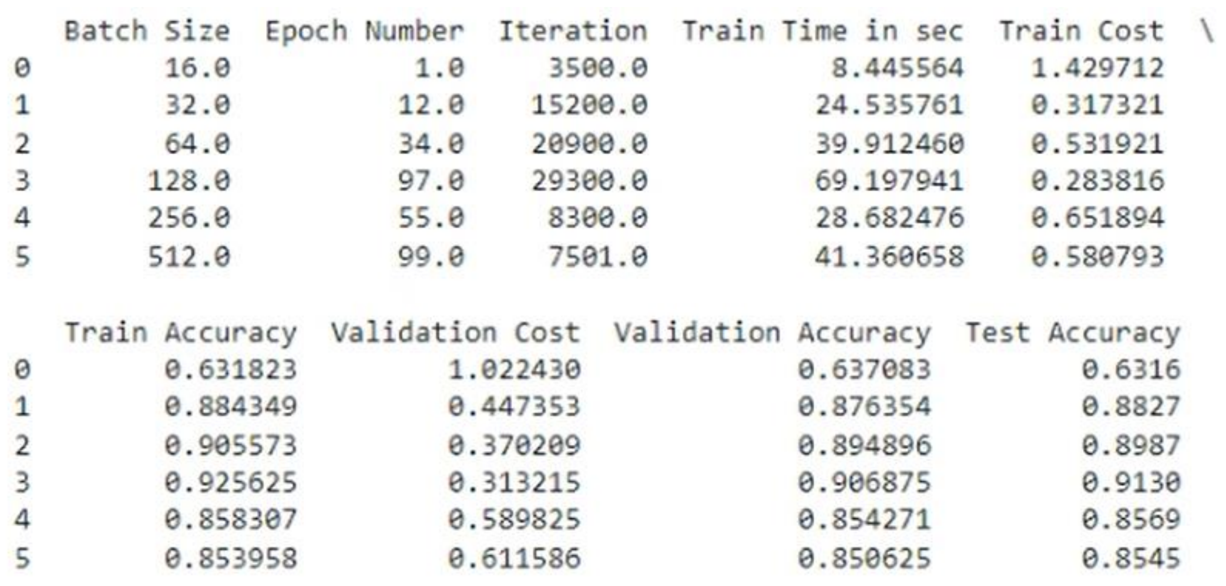
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 256



Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 512



Summary:



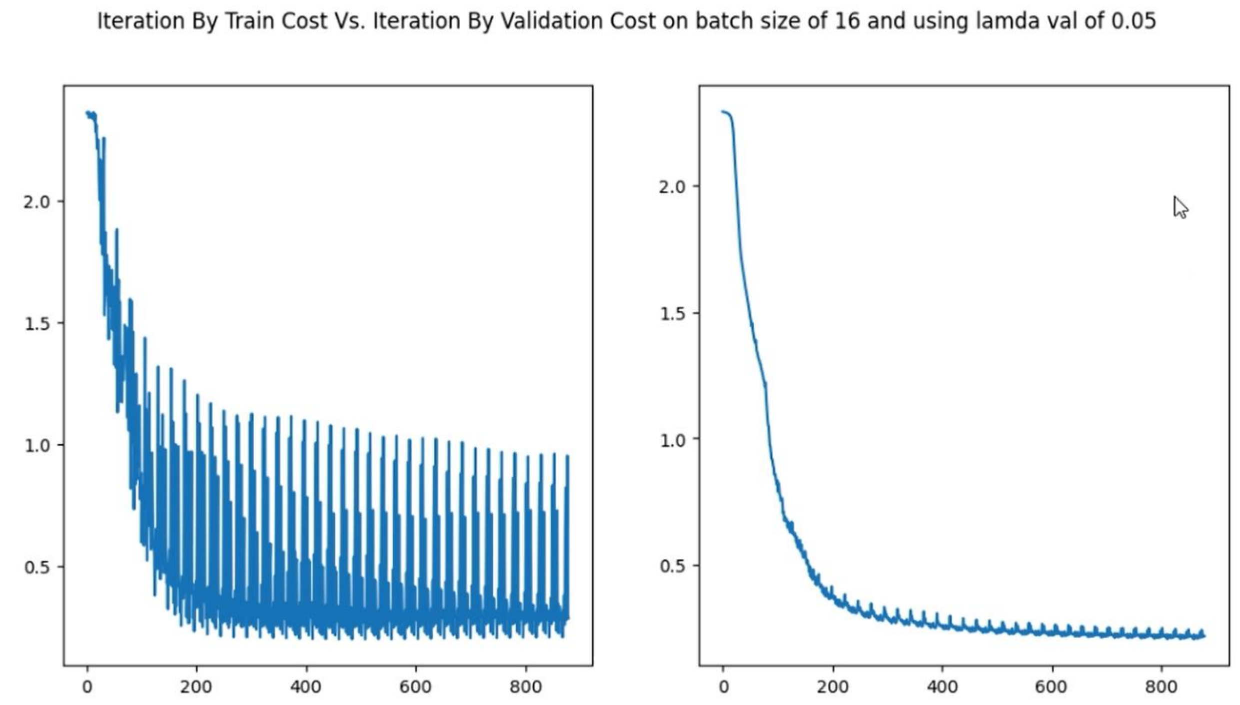
```
   Batch Size  Epoch Number  Iteration  Train Time in sec  Train Cost  \
0        16.0           1.0     3500.0           8.445564    1.429712
1        32.0          12.0    15200.0          24.535761    0.317321
2        64.0          34.0    20900.0          39.912460    0.531921
3       128.0          97.0    29300.0          69.197941    0.283816
4       256.0          55.0     8300.0          28.682476    0.651894
5       512.0          99.0     7501.0          41.360658    0.580793

   Train Accuracy  Validation Cost  Validation Accuracy  Test Accuracy
0        0.631823         1.022430             0.637083         0.6316
1        0.884349         0.447353             0.876354         0.8827
2        0.905573         0.370209             0.894896         0.8987
3        0.925625         0.313215             0.906875         0.9130
4        0.858307         0.589825             0.854271         0.8569
5        0.853958         0.611586             0.850625         0.8545
```

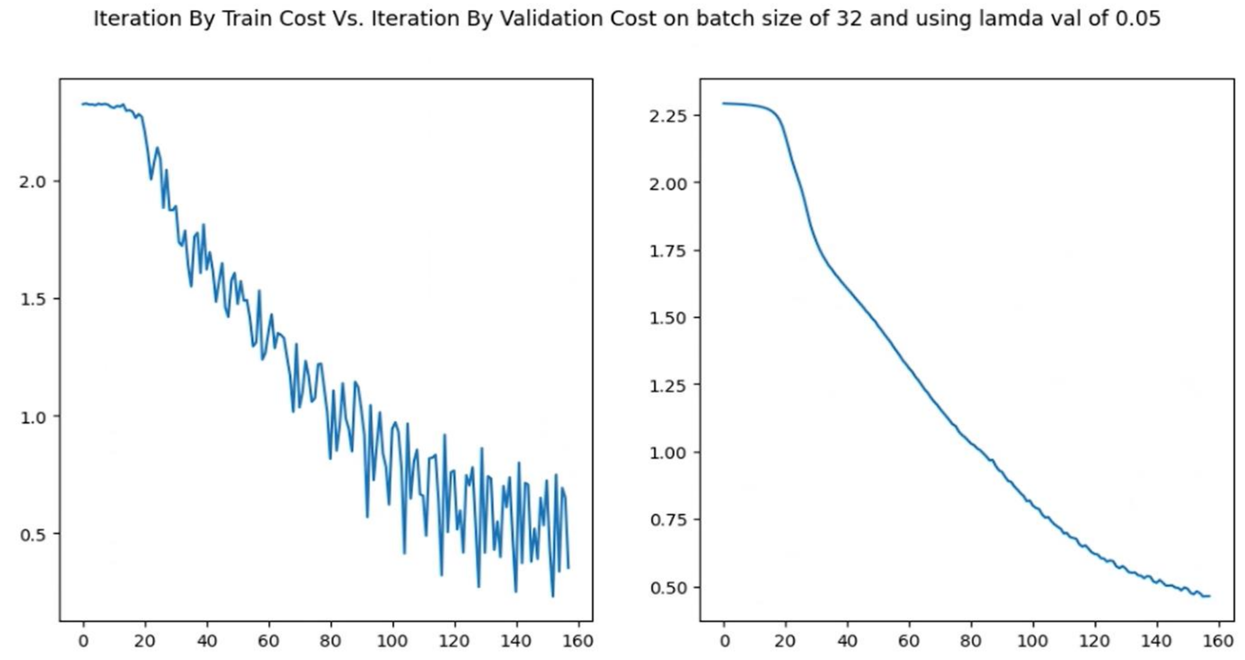
From the summary table above, we concluded that the optimal batch size to use with batch_norm = True is 128 since it has the highest accuracy of 0.913 with a runtime of 69.2 seconds.
If we are deciding by the runtime, then we would recommend using the batch size 32, it has similar accuracy of 0.8827 (difference of 0.03 in accuracy) with a much faster runtime of 24.5 which is 2.8 times faster than batch size 128.
The difference in runtime is attributed to the fact that we do batch normalization, the bigger the batch, the longer our model needs to normalize it.
Note: The main contributor to the runtime values is the epoch number.

Now: using the implemntation and the power of L2 Rgulization.

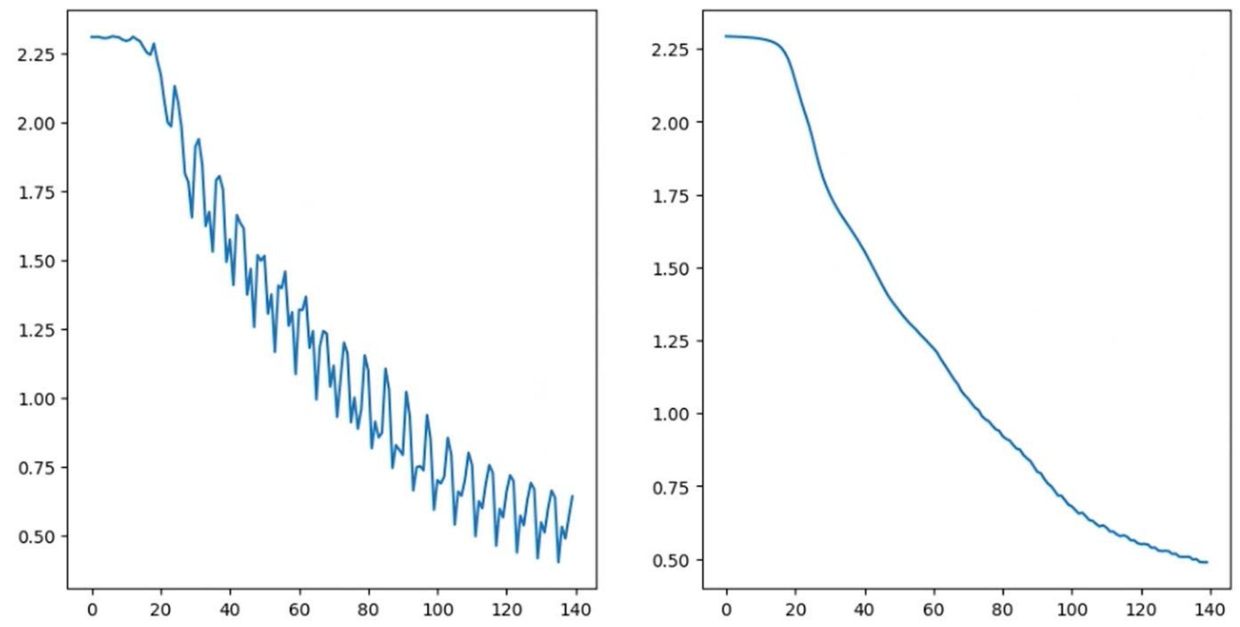
Note: we tried multiple lambda values ([0.2, 0.15, 0.1 0.05]) and decided to go with 0.05 since it gave us the best results.
(The reason why 0.05 is our optimal value is because our network before L2 wasn’t bias to overfitting, so using L2 optimization didn’t improve our performance that much. That’s why using higher lambda values backfired, and we received lower performances)

Part 1 (without batch norm):
We ran our network with the same parameters as before, this time with lambda = 0.05 and batch norm = False

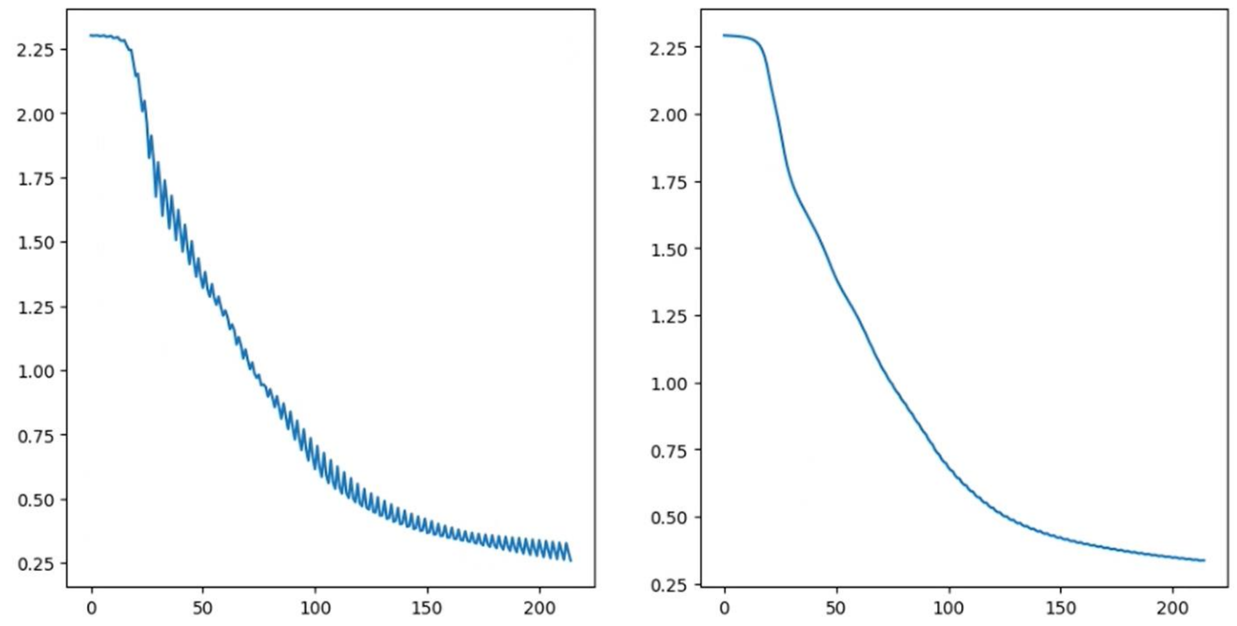




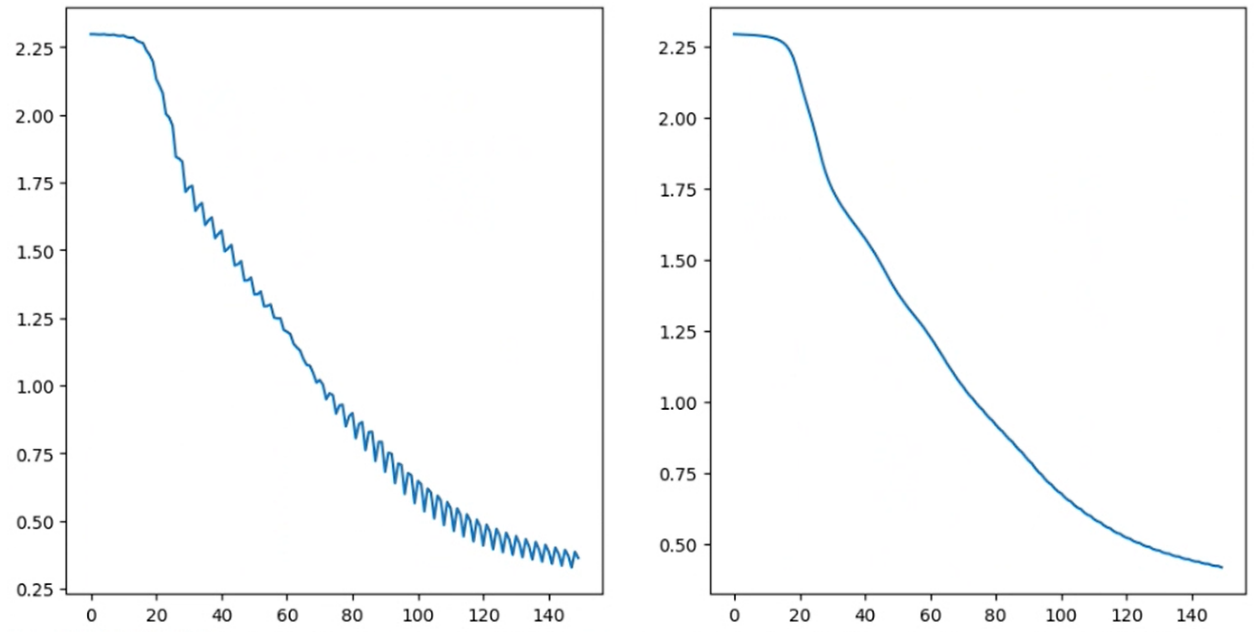
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 64 and using lamda val of 0.05



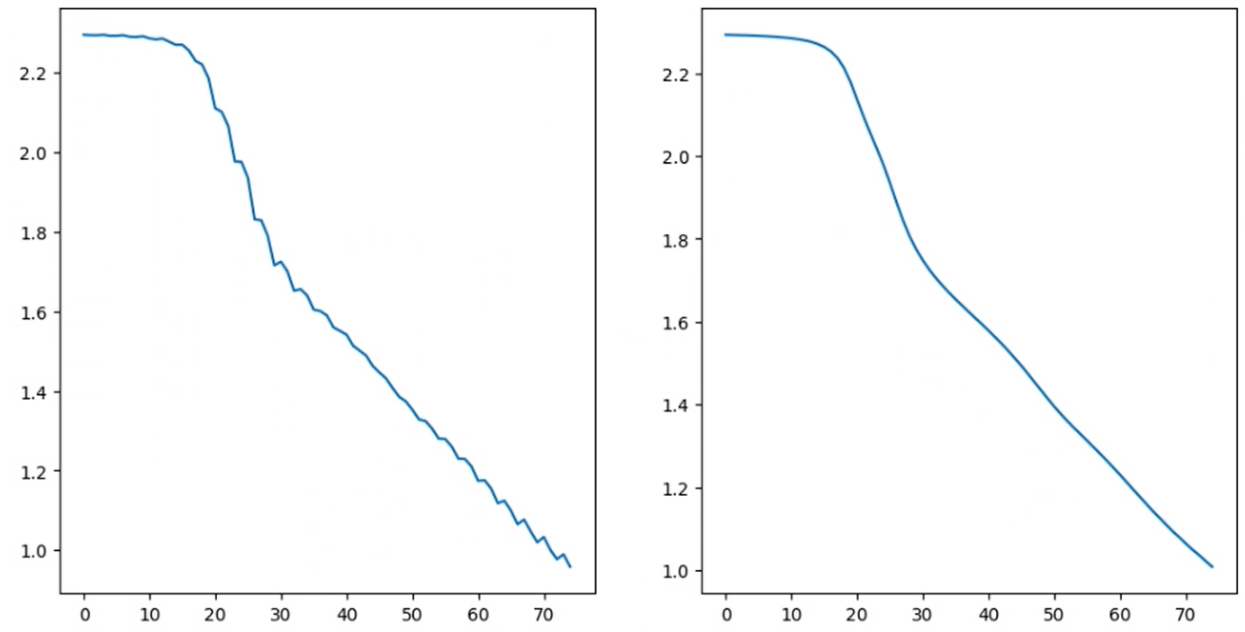
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 128 and using lamda val of 0.05



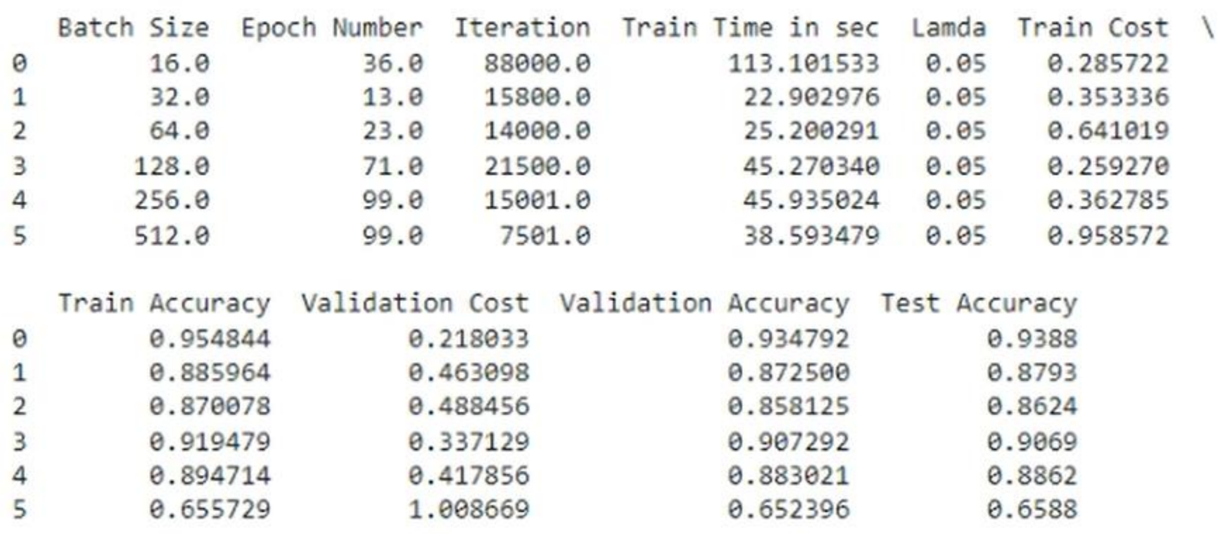
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 256 and using lamda val of 0.05



Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 512 and using lamda val of 0.05



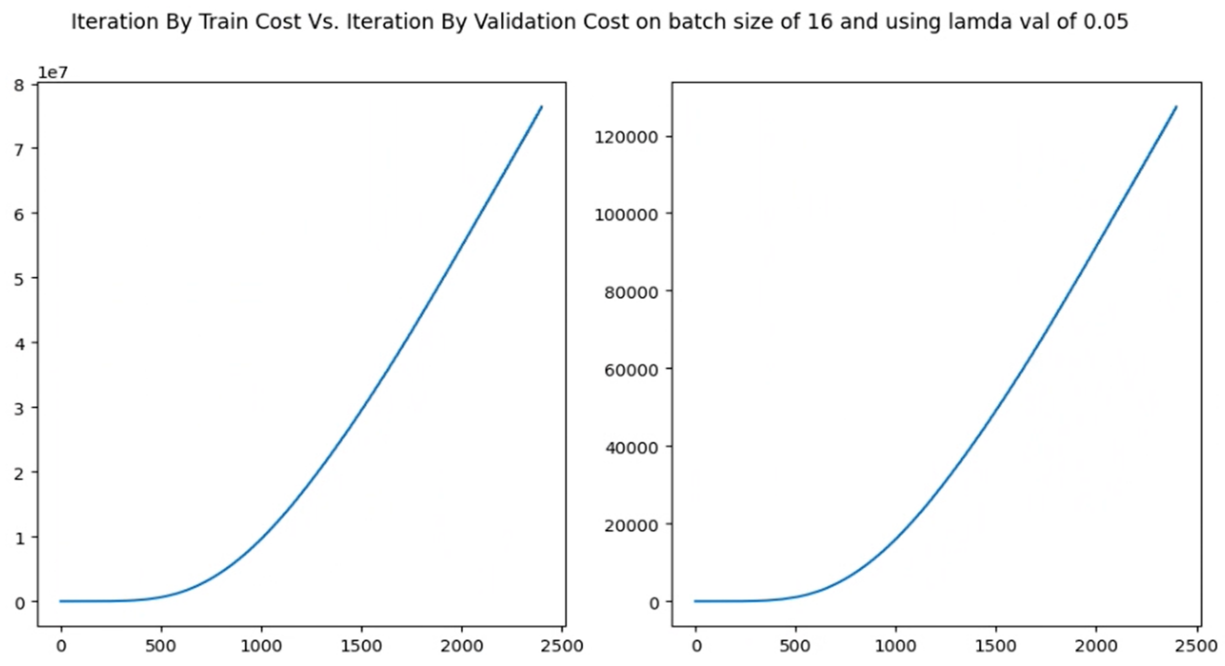
Summary:



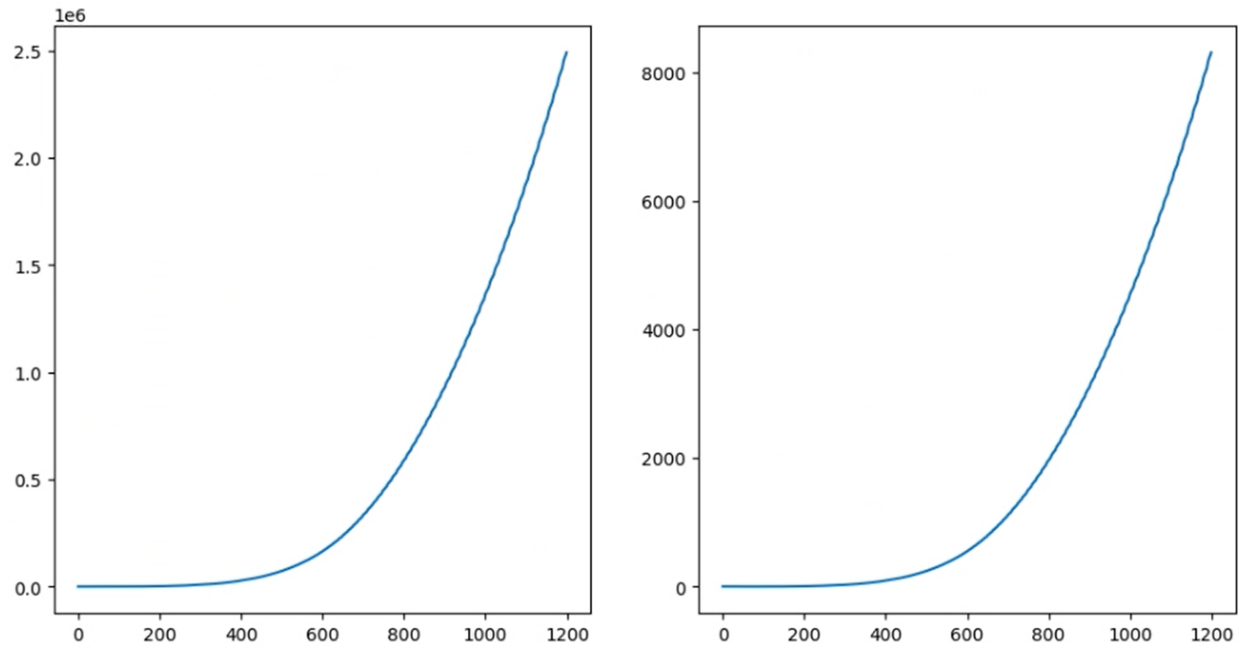
```
   Batch Size  Epoch Number  Iteration  Train Time in sec  Lamda  Train Cost  \
0        16.0          36.0    88000.0         113.101533   0.05    0.285722
1        32.0          13.0    15800.0          22.902976   0.05    0.353336
2        64.0          23.0    14000.0          25.200291   0.05    0.641019
3       128.0          71.0    21500.0          45.270340   0.05    0.259270
4       256.0          99.0    15001.0          45.935024   0.05    0.362785
5       512.0          99.0     7501.0          38.593479   0.05    0.958572

   Train Accuracy  Validation Cost  Validation Accuracy  Test Accuracy
0        0.954844         0.218033             0.934792         0.9388
1        0.885964         0.463098             0.872500         0.8793
2        0.870078         0.488456             0.858125         0.8624
3        0.919479         0.337129             0.907292         0.9069
4        0.894714         0.417856             0.883021         0.8862
5        0.655729         1.008669             0.652396         0.6588
```

The optimal value for batch size for the test accuracy is easily 16, we got an accuracy of almost 0.94 but it has a very big runtime of 113 seconds, compared to batch size 128 which has an accuracy of 0.907 with a much shorter runtime of 45 seconds (more than twice as fast).

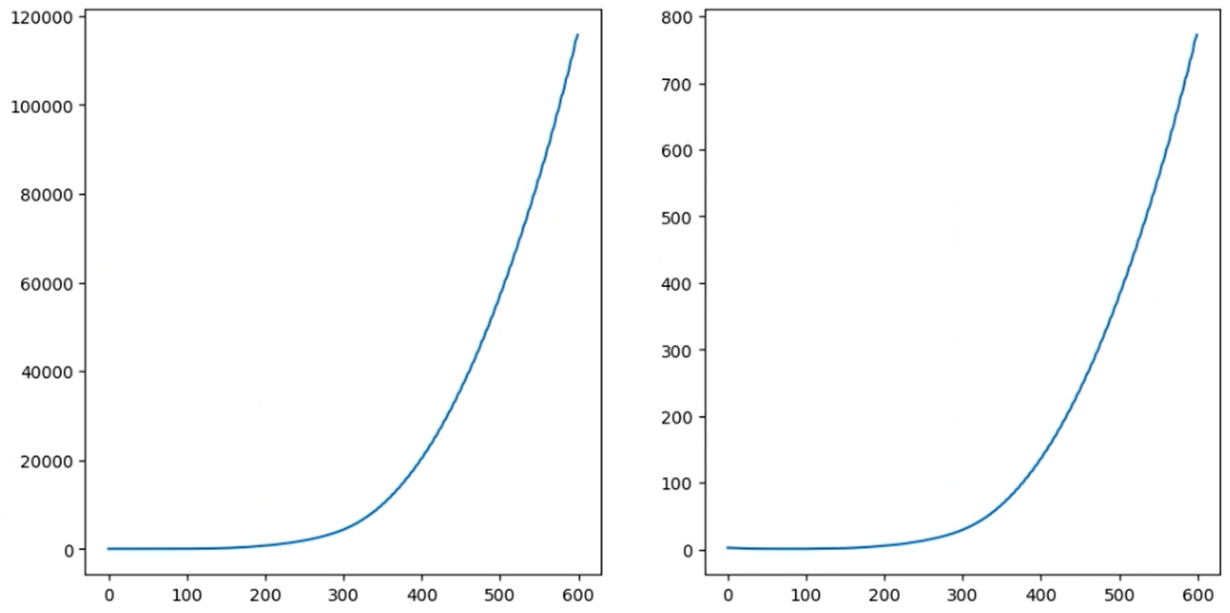
Part 2 (with batch norm):
We ran our network with the same parameters, this time with batch norm = True and with lambda = 0.05 (same as before)



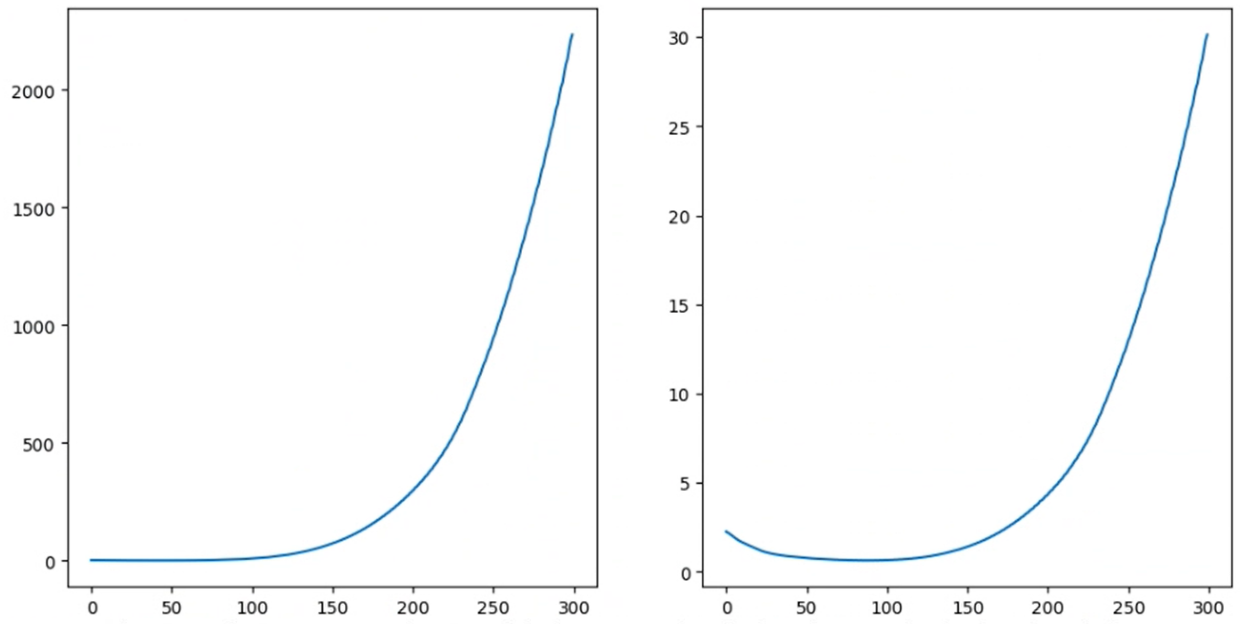
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 32 and using lamda val of 0.05



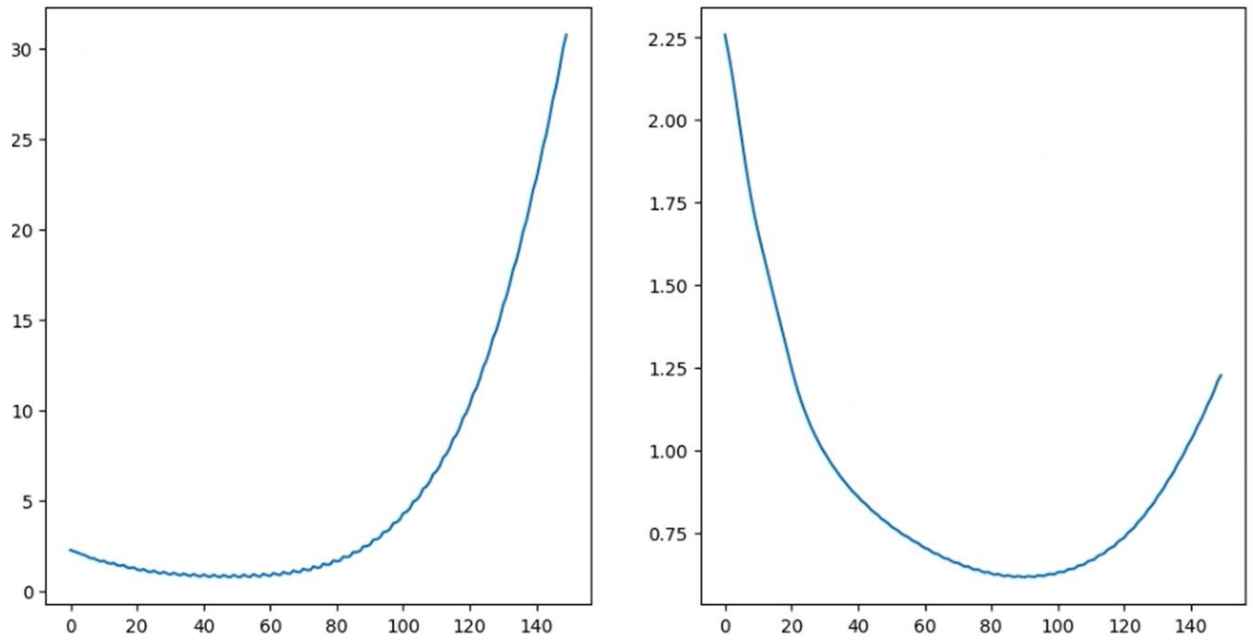
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 64 and using lamda val of 0.05

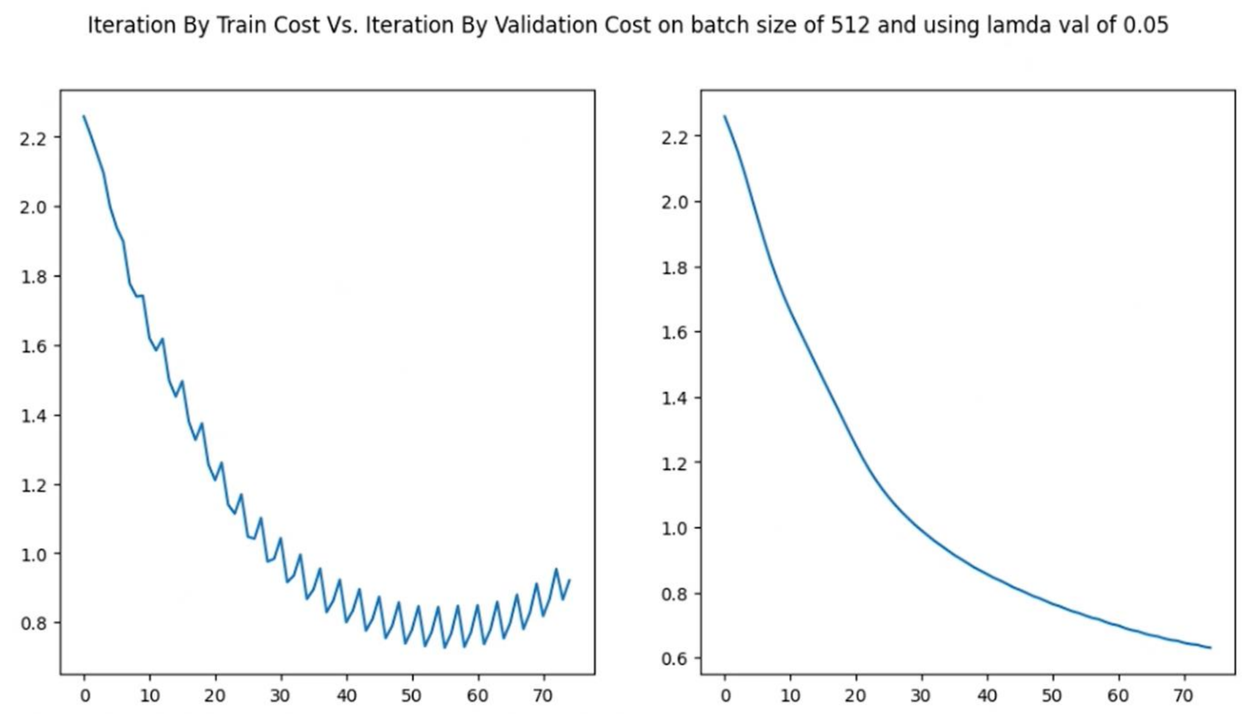


Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 128 and using lamda val of 0.05



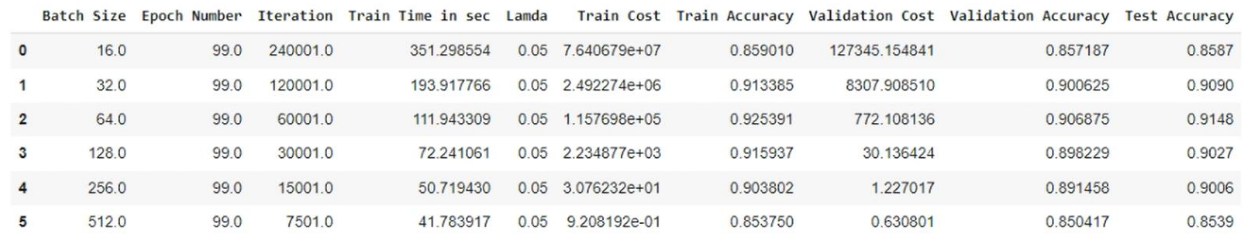
Iteration By Train Cost Vs. Iteration By Validation Cost on batch size of 256 and using lamda val of 0.05





Note: a possible reason why some of the graphs are increasing in cost instead of decreasing is because by normalizing the activations, it can sometimes lead to changes in the scale of the loss. The regularization effects of L2 regularization and the normalization effects of batch normalization might be interacting in a way that causes the raw loss values to increase while still improving generalization.

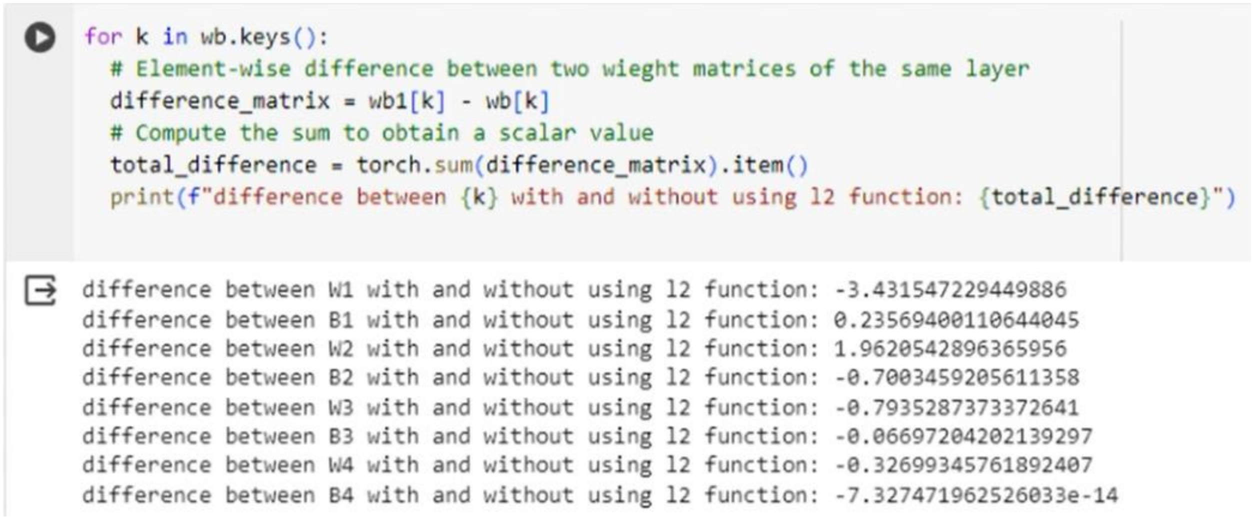
Summary:



	Batch Size	Epoch Number	Iteration	Train Time in sec	Lamda	Train Cost	Train Accuracy	Validation Cost	Validation Accuracy	Test Accuracy
0	16.0	99.0	240001.0	351.298554	0.05	7.640679e+07	0.859010	127345.154841	0.857187	0.8587
1	32.0	99.0	120001.0	193.917766	0.05	2.492274e+06	0.913385	8307.908510	0.900625	0.9090
2	64.0	99.0	60001.0	111.943309	0.05	1.157698e+05	0.925391	772.108136	0.906875	0.9148
3	128.0	99.0	30001.0	72.241061	0.05	2.234877e+03	0.915937	30.136424	0.898229	0.9027
4	256.0	99.0	15001.0	50.719430	0.05	3.076232e+01	0.903802	1.227017	0.891458	0.9006
5	512.0	99.0	7501.0	41.783917	0.05	9.208192e-01	0.853750	0.630801	0.850417	0.8539

Looking at the test accuracy we can say that the optimal batch size is 64 since it got the highest accuracy of 0.915 with a runtime of 112 seconds.

Weight comparison with/without L2:



```
for k in wb.keys():
  # Element-wise difference between two wieght matrices of the same layer
  difference_matrix = wb1[k] - wb[k]
  # Compute the sum to obtain a scalar value
  total_difference = torch.sum(difference_matrix).item()
  print(f"difference between {k} with and without using l2 function: {total_difference}")
```

```
difference between W1 with and without using l2 function: -3.431547229449886
difference between B1 with and without using l2 function: 0.23569400110644045
difference between W2 with and without using l2 function: 1.9620542896365956
difference between B2 with and without using l2 function: -0.7003459205611358
difference between W3 with and without using l2 function: -0.7935287373372641
difference between B3 with and without using l2 function: -0.06697204202139297
difference between W4 with and without using l2 function: -0.32699345761892407
difference between B4 with and without using l2 function: -7.327471962526033e-14
```

Looking at the difference between the weights, we can see that there is a minimal difference between them, which can be attributed to the fact that our network wasn't bias to overfitting and thus didn't improve that much upon regularization.