CIFAR100 CNN Report

Written by:

Brandon Silva
9/23/18

Parameters Used

A) Starting Learning Rate: 0.001

- I used this as my starting learning rate because of the large amount of different classes to predict. Since there were so many, starting at a small learning rate would help obtain better accuracy. While training, if the validation loss did not improve for two epochs, then the learning rate would automatically decrease by a factor of 0.1 each time this happened. So depending on how well certain runs did, the learning rate got as small as $1*10^{-8}$.

B) Epochs: 30

- I used 30 epochs as it was the amount of time for the network to mostly converge and also leave extra epochs for small increases in accuracy at the end. It also was the right amount to leave training times under an hour each run.

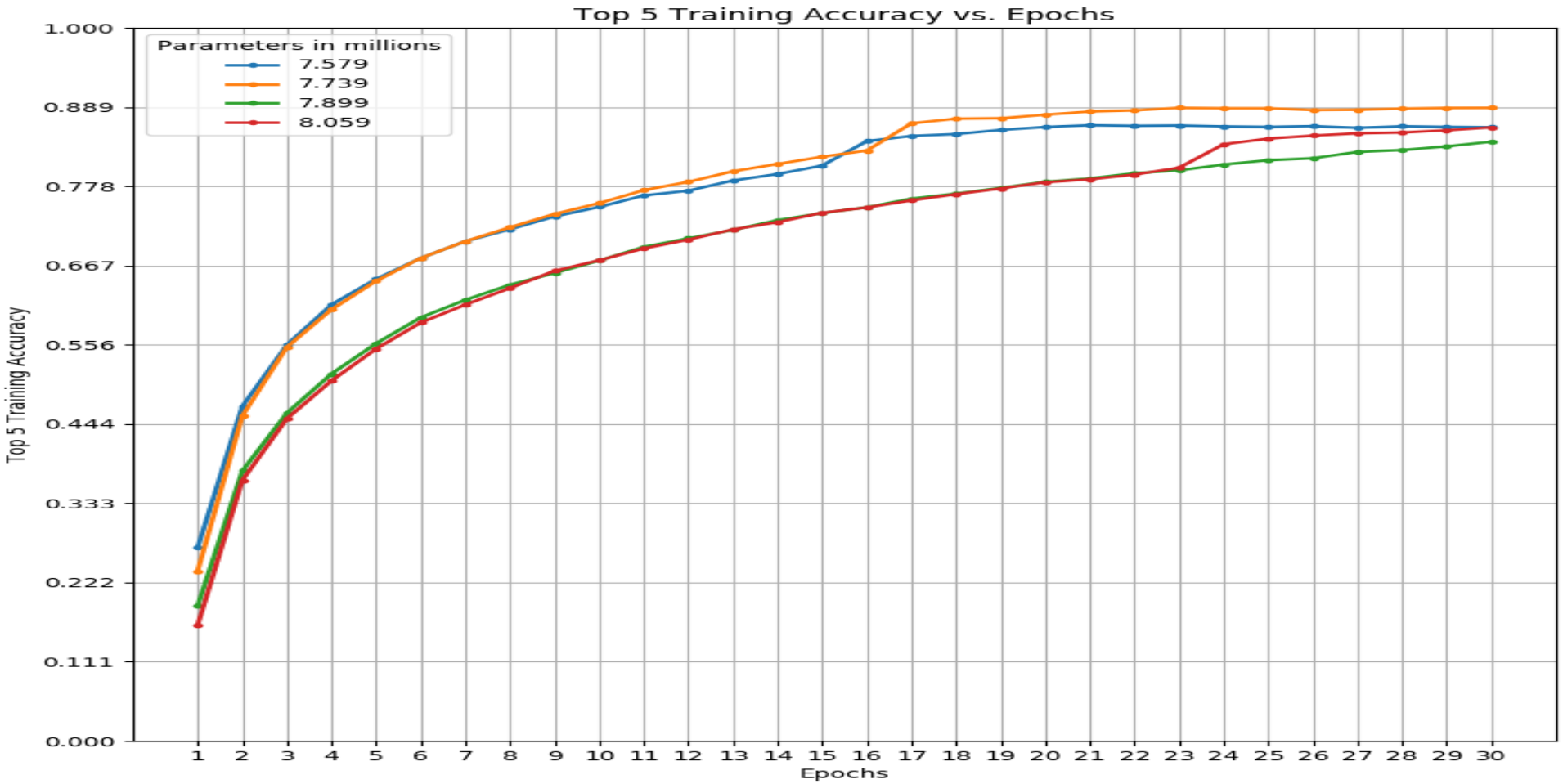
C) Batch Size: 64

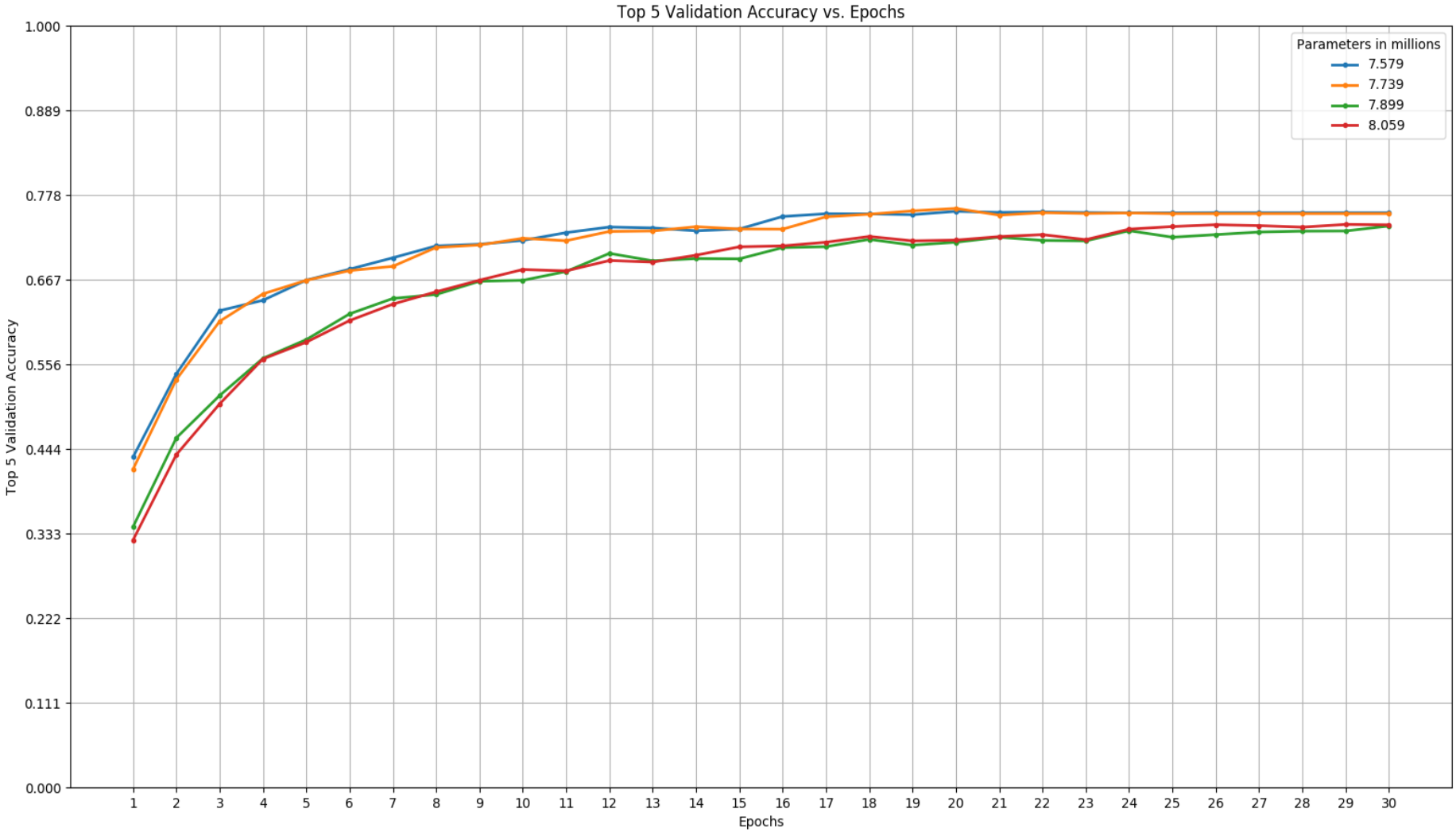
- I used this number since I had enough memory to accommodate the data. 64 is a good size for computers with 16gb of memory and the size of this data.

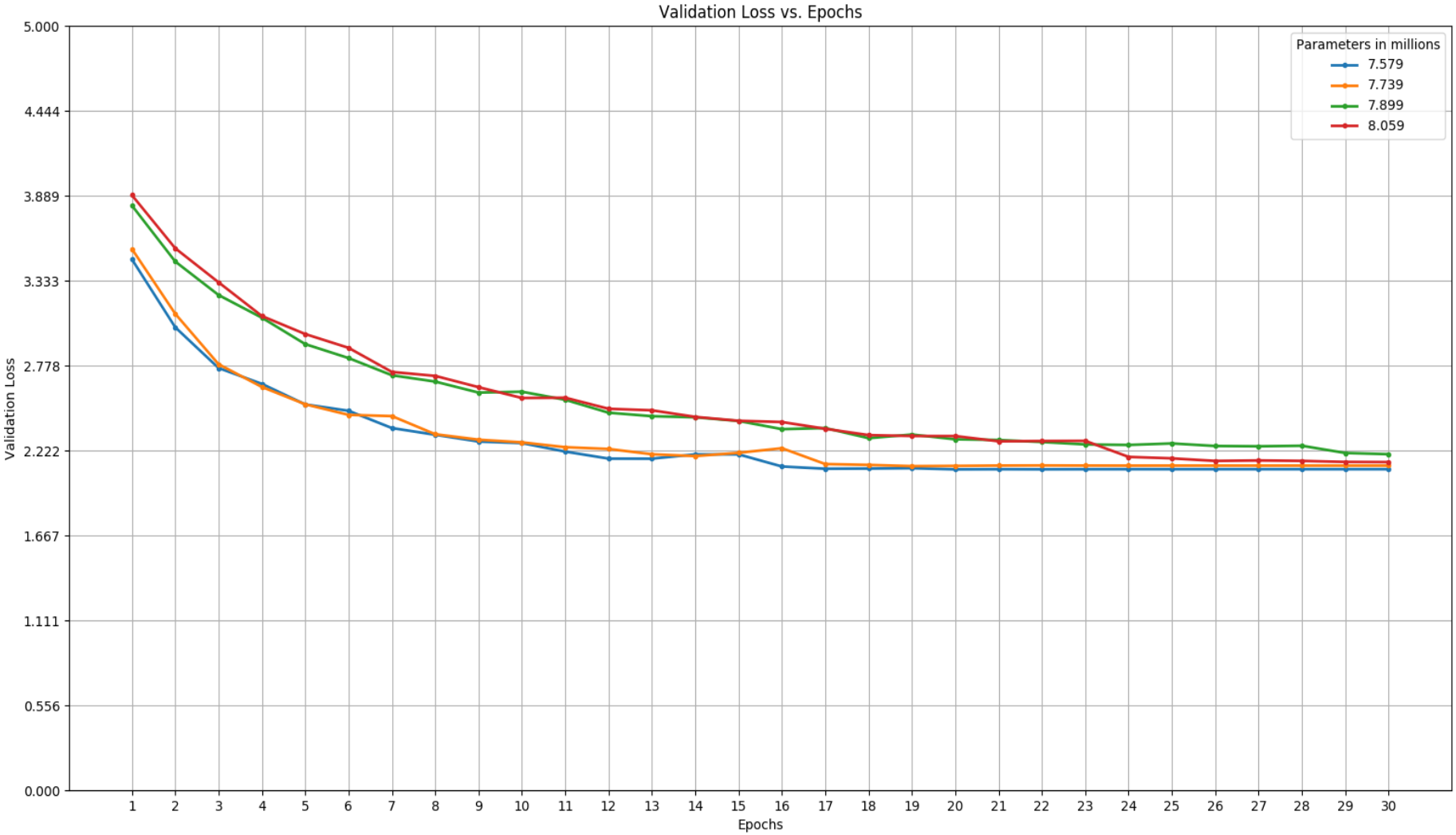
D) Filter size for each Convolution Layer: 200, Kernel Size: 2, Pooling Size: 2, Dense size: 550

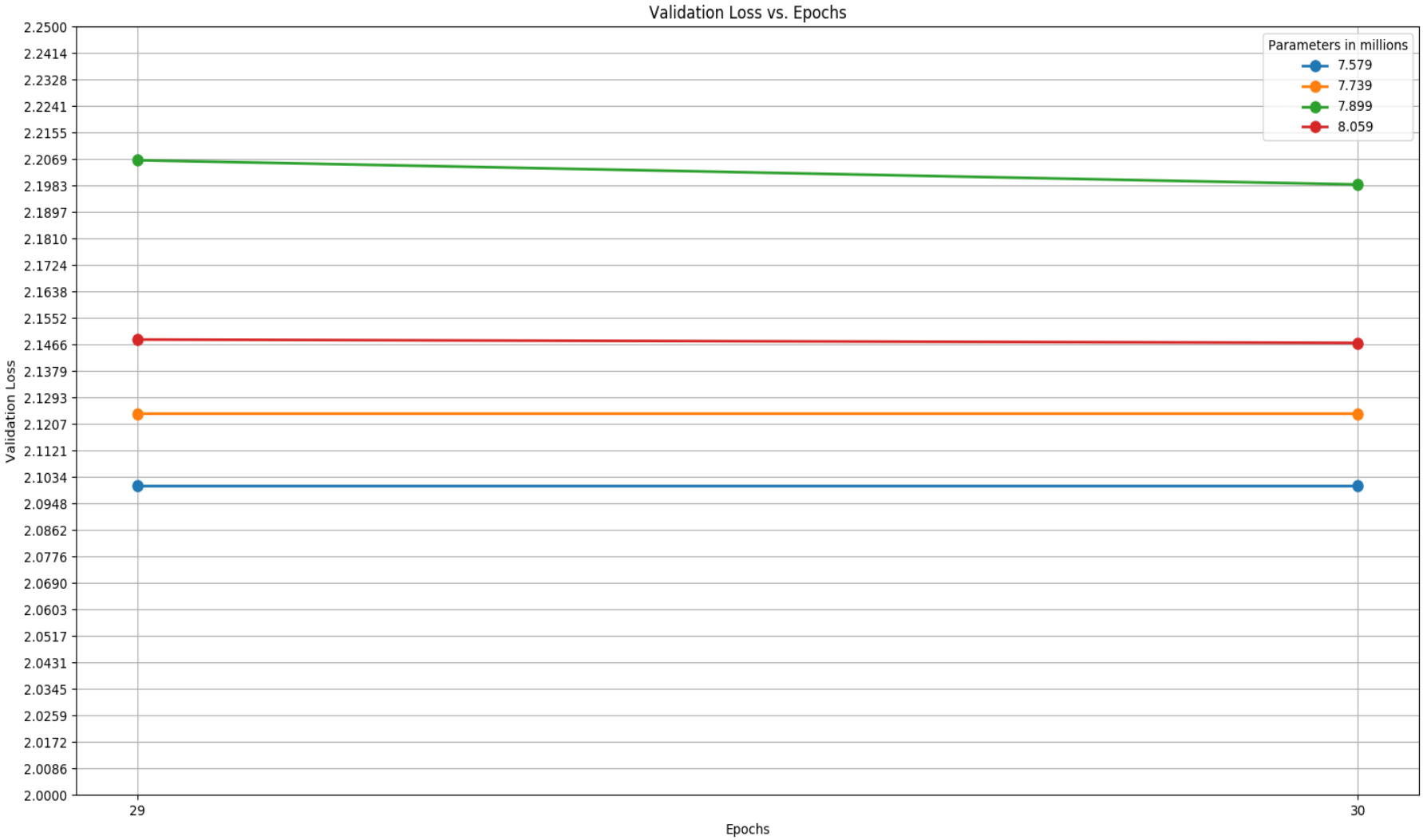
- I wanted to keep the amount of filters the same for each of the layers so I could measure the effect of adding more convolution layers by itself, without having size of the layers contribute. 200 was a good amount for my hardware capabilities and network accuracy.
- I choose the kernel and pooling sizes to be 2 since the data I was working with was pretty small images, 32x32 pixels. Larger kernel sizes would result in more padding and that could introduce more noise into the data, lowering accuracy. Larger pooling sizes would decrease the size of the input data by a large amount that would decrease accuracy of the network.
- I choose 550 nodes as the size for the dense classifier layer because it was the most I could give before computation time increased dramatically. It was a good balance between accuracy and training time.

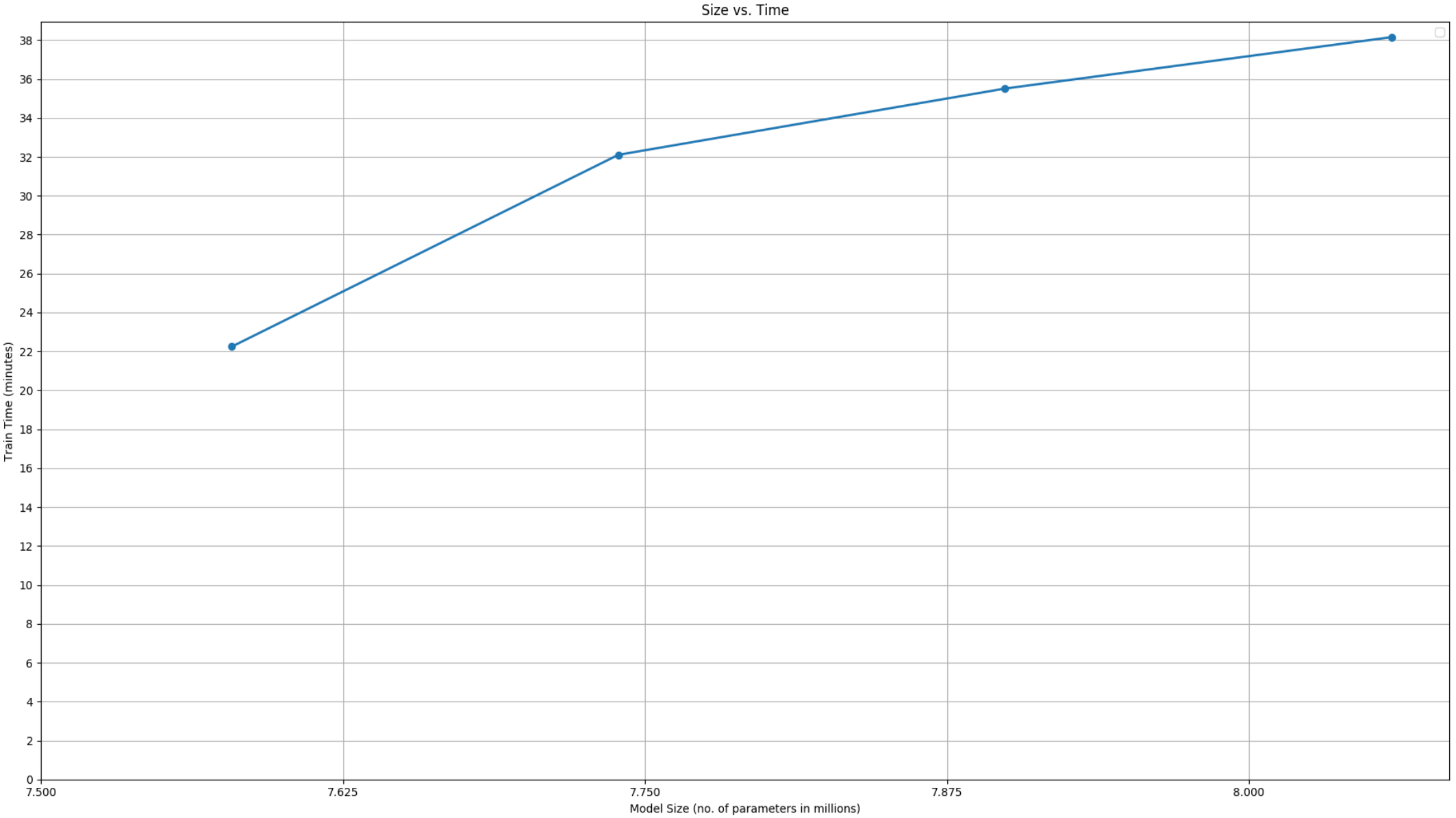
Largest Data Set Size Graphs (largest data-set and network size is in red)



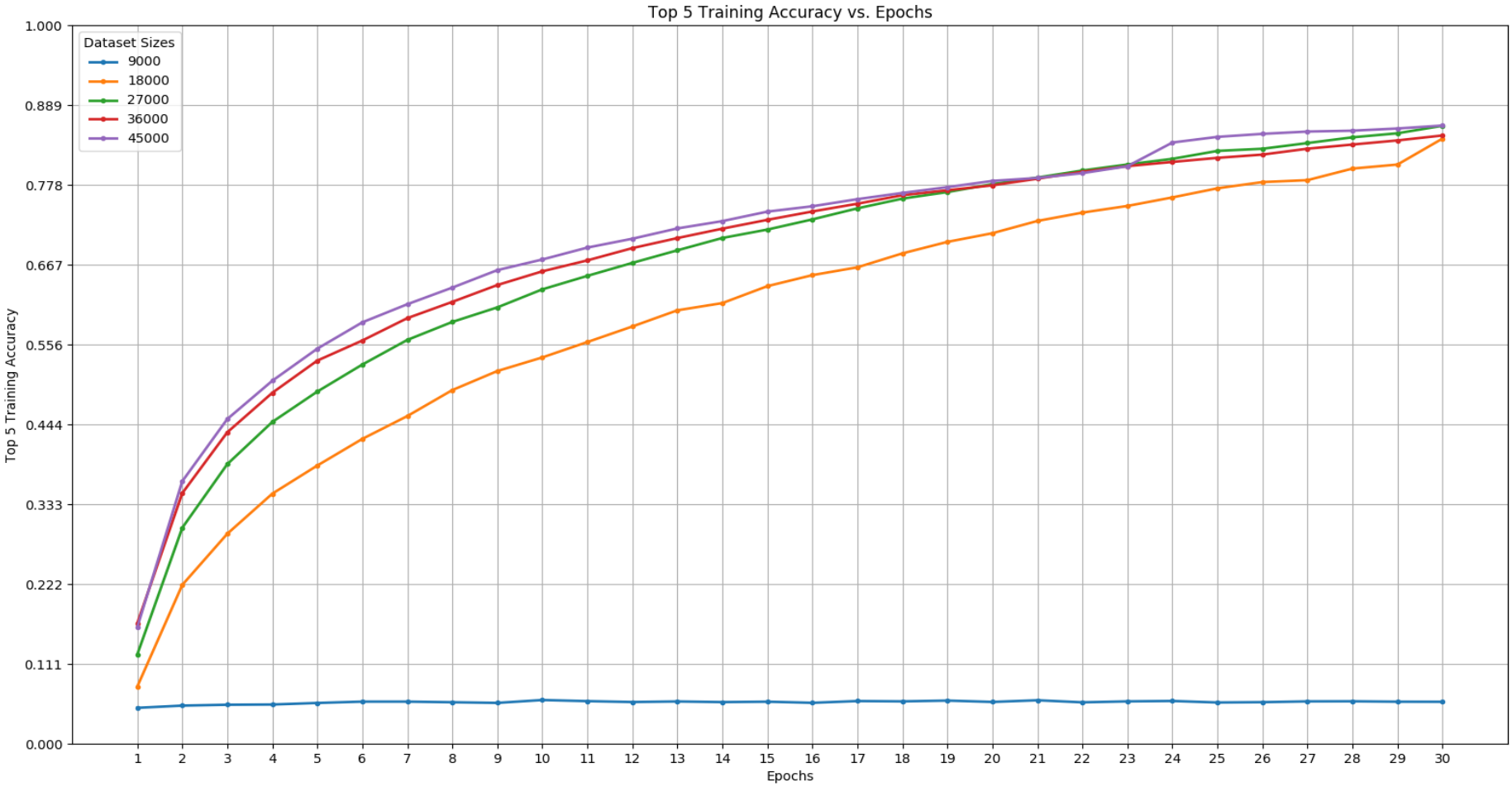

Top 5 Validation Accuracy vs. Epochs
Top 5 Validation Accuracy
Epochs
Parameters in millions
7.579
7.739
7.899
8.059
1.000
0.889
0.778
0.667
0.556
0.444
0.333
0.222
0.111
0.000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

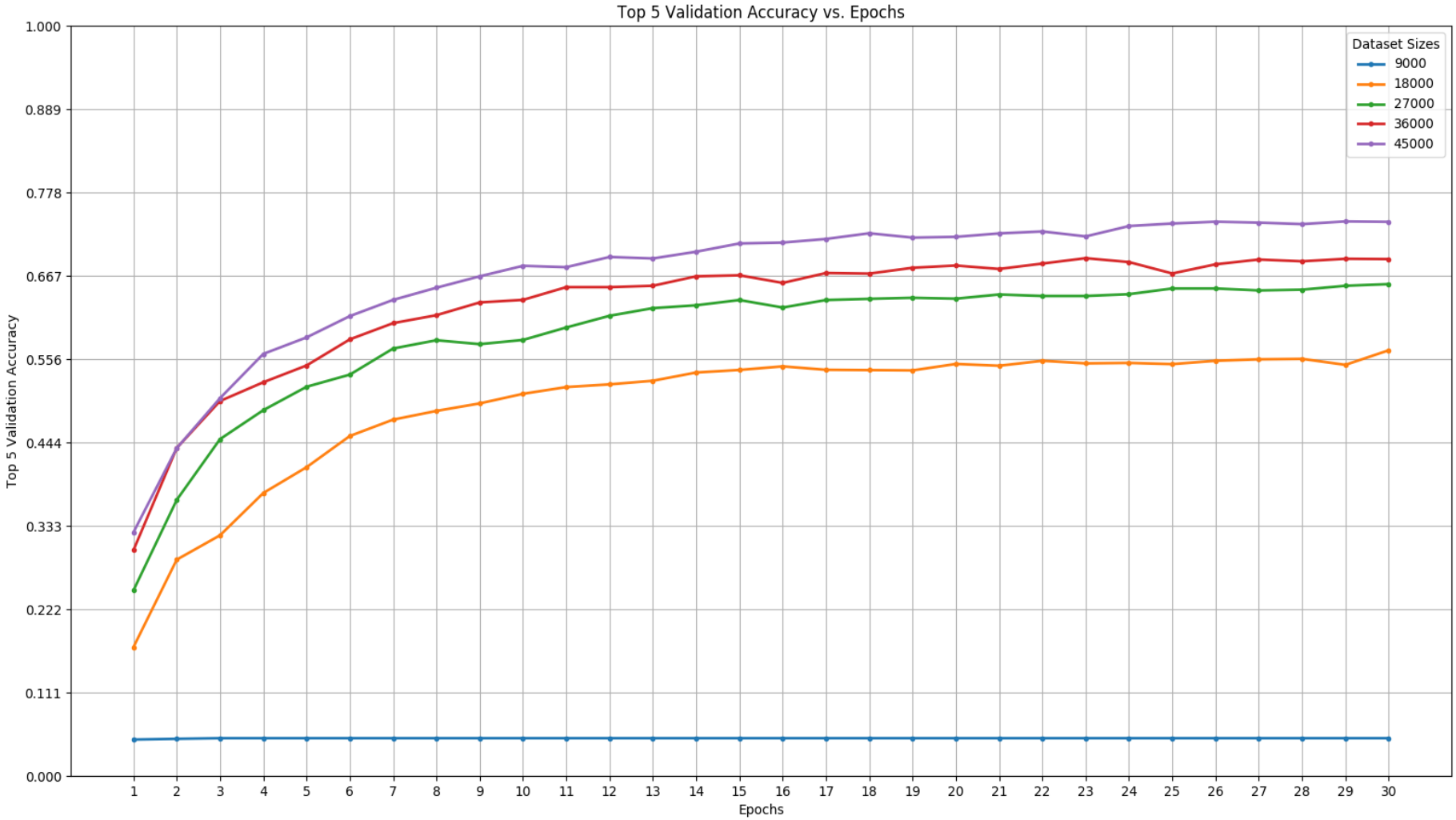

Validation Loss vs. Epochs
Validation Loss
Epochs
Parameters in millions
7.579
7.739
7.899
8.059
5.000
4.444
3.889
3.333
2.778
2.222
1.667
1.111
0.556
0.000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

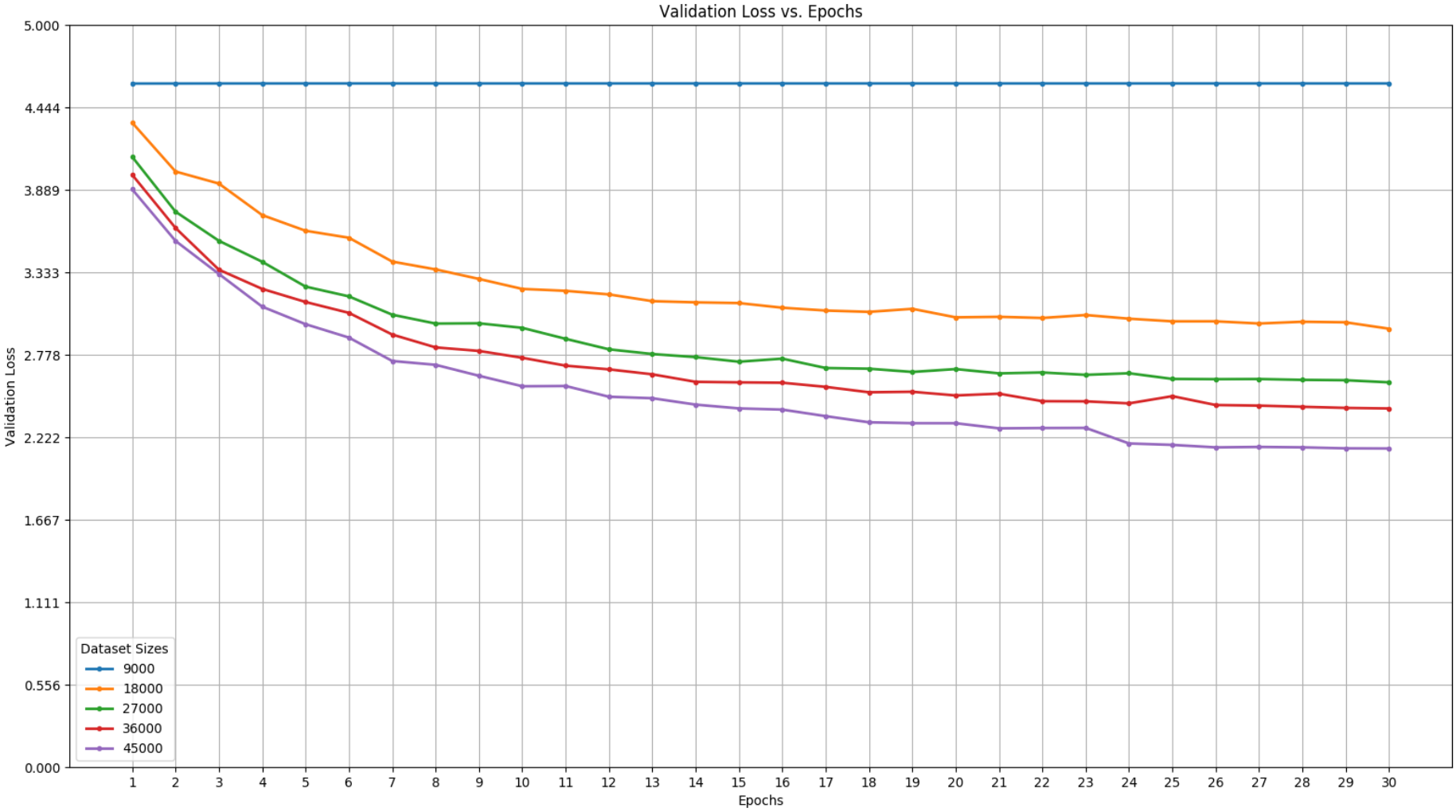

Validation Loss vs. Epochs
Parameters in millions
7.579
7.739
7.899
8.059
Validation Loss
2.2500
2.2414
2.2328
2.2241
2.2155
2.2069
2.1983
2.1897
2.1810
2.1724
2.1638
2.1552
2.1466
2.1379
2.1293
2.1207
2.1121
2.1034
2.0948
2.0862
2.0776
2.0690
2.0603
2.0517
2.0431
2.0345
2.0259
2.0172
2.0086
2.0000
29
30
Epochs

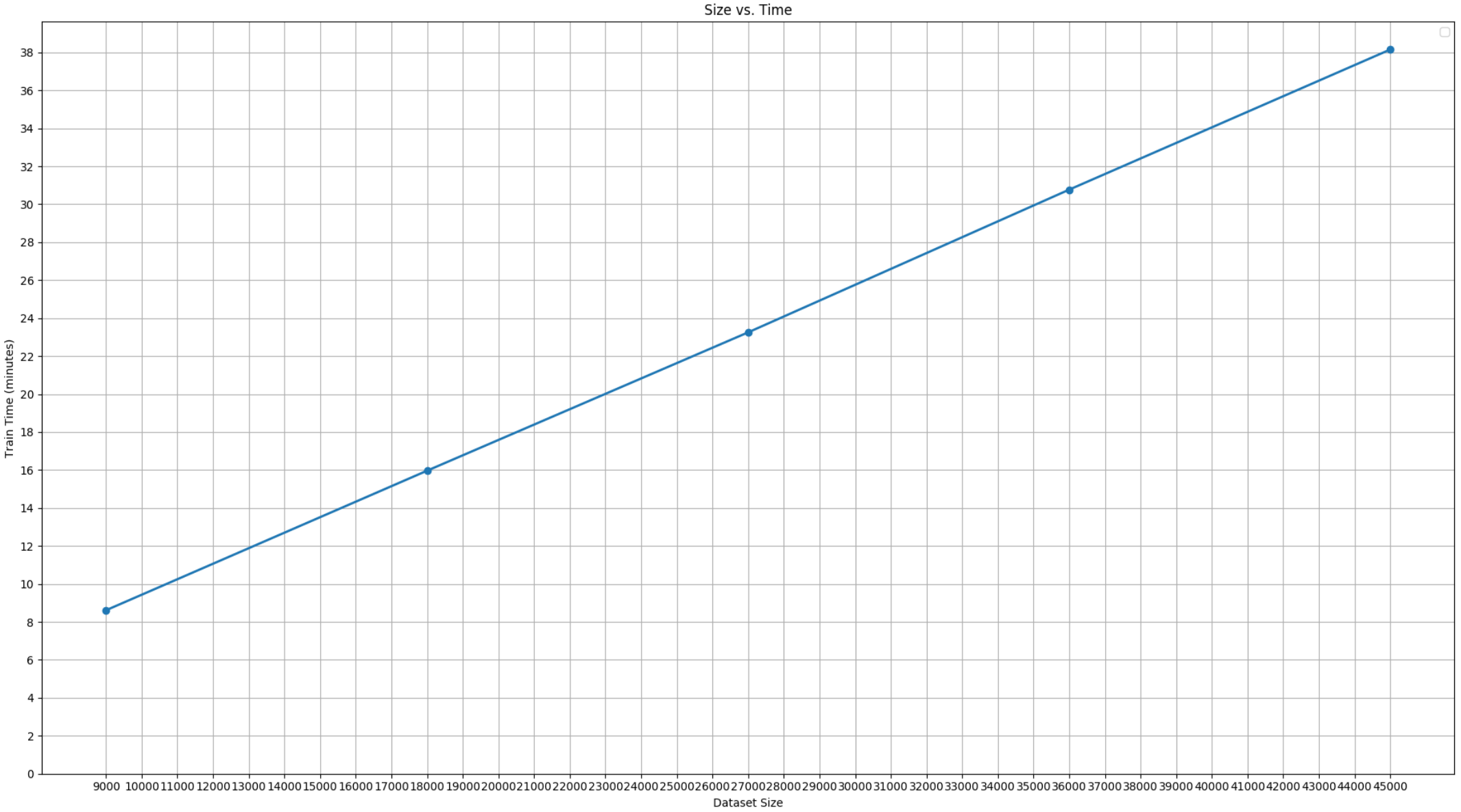

Size vs. Time
Train Time (minutes)
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
7.500
7.625
7.750
7.875
8.000
Model Size (no. of parameters in millions)

Largest Network Size Graphs




Top 5 Validation Accuracy vs. Epochs
Top 5 Validation Accuracy
Epochs
Dataset Sizes
9000
18000
27000
36000
45000
1.000
0.889
0.778
0.667
0.556
0.444
0.333
0.222
0.111
0.000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30


Validation Loss vs. Epochs
Validation Loss
5.000
4.444
3.889
3.333
2.778
2.222
1.667
1.111
0.556
0.000
Epochs
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Dataset Sizes
9000
18000
27000
36000
45000


Size vs. Time
Train Time (minutes)
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
Dataset Size
9000 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 20000 21000 22000 23000 24000 25000 26000 27000 28000 29000 30000 31000 32000 33000 34000 35000 36000 37000 38000 39000 40000 41000 42000 43000 44000 45000

Insights Gained

A) Increasing network size slightly increased validation loss at each epoch.

- As network sized increased, training loss was higher at each epoch than the previously smaller network, meaning the network would need more iterations to converge. It seems that the smallest two networks converged around epoch eighteen to nineteen, while the others were still decreasing up until the last epoch. Therefore, from the trends of the graphs, the larger parameter networks will converge at a larger iteration than tested here, most likely resulting in better loss and accuracy comparative to the smaller networks at that point. However, in thirty epochs the smaller networks will decrease and converge faster. This shows that smaller networks could be good models to train to test whether a larger network would actually work given the design of the model and the data provided, with using smaller resources, time and data.

B) Increasing network size had little to no effect on training/validation top five accuracy

- From the graphs, all four differently sized networks seemed to follow the same trend and converge around the same point, with the first two smaller networks out performing the larger ones slightly. However, as seen on the validation loss, it seems that these numbers will continue to climb with more iterations, and converge at a higher accuracy. I think that even though the networks are bigger, the general structure of the network gives high top five accuracy that is not effected by network size by a large margin, meaning that smaller networks can show how larger models may perform. This also shows that this network structure may be a good network to build a bigger model off of.

C) Increasing data set size dramatically decreased validation loss, and increased top five accuracy.

- As the size of the training data set increased, there were large improvements in both training loss and top five validation accuracy. Top five training accuracy seemed to improve only slightly, but that may be because of more over fitting on the smaller amount of data. The largest data set size tested scored the lowest validation loss, and highest top five accuracy. With the smallest training set size of nine thousand, the network did not train at all, even with fine tuning and adjustments to dropout. This shows that more clean data will result in much better network performance, as it reduces over fitting and gives the network more data to extract key features and make predictions. The performance with the largest data set could further be improved by augmenting the data.

D) Increasing model size and data set size considerably increased training time.

- With each increase in either model size or data size, the time to train the model increased by about seven minutes. This is expected behavior, as increased data size means each

model needs more time to train on all the data on each epoch, and increased model size means more computational power is needed for each batch, which will increase the time to train an epoch on the whole model. It is an interesting note that when adding the same amount of data each time, the time to train the model increases linearly, which would make sense given that nothing else is changing. But when model sized increased, it is different at each increase, creating more of a curve. Most likely, as we increase the number of parameters, training time would exponentially increase.

Implementation Details

A) Network Description

- The network consisted of two blocks of two convolution layers followed by a max pooling layer. I increased the network size by adding another convolution layer to the second block first, then the first block, then finally the second block again. The final network size had the first block with three convolution layers followed by a max pooling layer, and the second block having 4 convolution layers followed by a max pooling layer.
- The classifier layer had one hidden dense layer and an output layer for probability of predicted image.
- Dropout was added after the first block, after the fourth convolution layer in the second block, after the second block, and after the first hidden layer in the classifier dense layer to help with over fitting.
- Activation functions of all layers was the ReLu function, except the last output layer, which had a softmax activation for probability predictions.

B) Language and Framework Used

- I used Python as my programming language, employing the Keras deep learning library with the TensorFlow backend. I used matplotlib to build the graphs.

C) Hardware Details

- I trained this model on my desktop, with the use of the NVIDIA 1060 GPU, with 6 GB of VRAM. I also had 16 GB of ram available to use.

Bonus

1. Dropout is used for regularization as to help prevent over fitting, which is when the network memorizes the training data instead of generalizing it, resulting in very high training accuracy but poor validation and testing accuracy. By using dropout, a randomly selected percentage of the nodes in the layer are “turned off”, meaning they are not used to calculate the predictions, therefore they are not trained on. This helps vary the weights used to help the model generalize better instead of memorizing the data, since more parameters makes it easier to over fit.
2. MSE is better for regression problems, since in these cases the outputs are like a real-valued function of the inputs, meaning they are closely related. Such as the case in linear regression problems, where this formula is derived from. Cross-entropy is better suited for binary or multi-class classification problems. Cross-entropy is shown to train models quicker when trained for these types of problems as it provides better information about how wrong a prediction is compared to MSE. This is because of the way weight updates work compared to the cross-entropy function. Early in learning, derivatives will be large from the cross-entropy function, causing greater weight updates. As learning continues, derivatives will get small, and so will the weight updates. This makes sense as the updates should be smaller since the error will be smaller later in training, where large updates could cause a jump out of a local or global minimum, which could happen more with MSE.
3. Dimensions with zero-padding:

Conv_1	(batch_size, 32, 32, 200)
Conv_2	(batch_size, 32, 32, 200)
Conv_3	(batch_size, 32, 32, 200)
Max_Pool_1	(batch_size, 16, 16, 200)
Dropout_1	(batch_size, 16, 16, 200)
Conv_4	(batch_size, 16, 16, 200)
Conv_5	(batch_size, 16, 16, 200)
Conv_6	(batch_size, 16, 16, 200)
Conv_7	(batch_size, 16, 16, 200)
Dropout_2	(batch_size, 16, 16, 200)
Max_Pool_2	(batch_size, 8, 8, 200)
Dropout_3	(batch_size, 16, 16, 200)
Flatten_1	(batch_size, 12800)
Dense_1	(batch_size, 550)
Dense_2	(batch_size, 100)

Without zero-padding:

Conv_1	(batch_size, 31, 31, 200)
Conv_2	(batch_size, 30, 30, 200)
Conv_3	(batch_size, 29, 29, 200)
Max_Pool_1	(batch_size, 14, 14, 200)
Dropout_1	(batch_size, 14, 14, 200)
Conv_4	(batch_size, 13, 13, 200)
Conv_5	(batch_size, 12, 12, 200)
Conv_6	(batch_size, 11, 11, 200)
Conv_7	(batch_size, 10, 10, 200)
Dropout_2	(batch_size, 10, 10, 200)
Max_Pool_2	(batch_size, 5, 5, 200)
Dropout_3	(batch_size, 5, 5, 200)
Flatten_1	(batch_size, 5000)
Dense_1	(batch_size, 550)
Dense_2	(batch_size, 100)