Emily Strong
INFO 7374 Spring 2019
Assignment 1 Part 1

Introduction

For part 1 of this assignment, I trained a multi-layer perceptron to the CIFAR-10 data set and tuned some of the parameters. The starter code for the model has 20 epochs, a batch size of 128, one hidden layer, and 512 neurons and a relu activation function for both the input and hidden layers with softmax as the activation function in the output layer, a learning rate of 0.001 and dropout rate of 0.2. The optimizer is RMSprop. This model does better than chance but the accuracy is low. Figure 1 shows the performance metrics of the model.

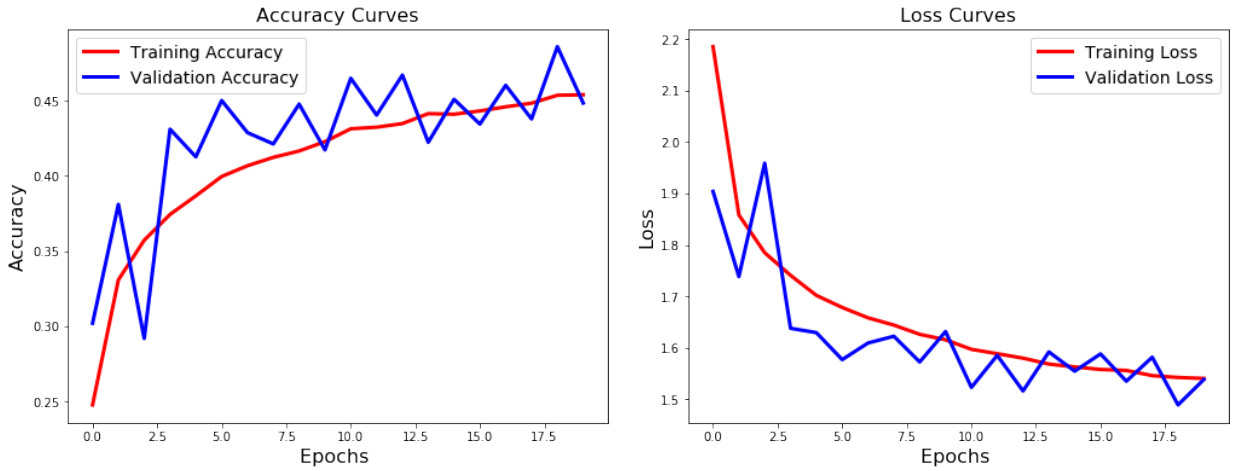

Figure 1: Accuracy and Loss Curves for Starter Code

Experimental Results

1) Epochs

For the first experiment I tested increasing the number of epochs in increments of 20. From looking at the accuracy and loss curves, I found that between 40 and 50 epochs the model begins to overfit, so I selected 40 epochs to use for the remaining experiments. Table 1 gives the final test accuracy and loss for each number of epochs.

Table 1: Epochs Experiment

Epochs	*Test Accuracy*	*Test Loss*
20	0.4808	1.471243192
40	0.4854	1.465798267
60	0.4755	1.484059889
80	0.4692	1.558606393
100	0.4553	1.549388475

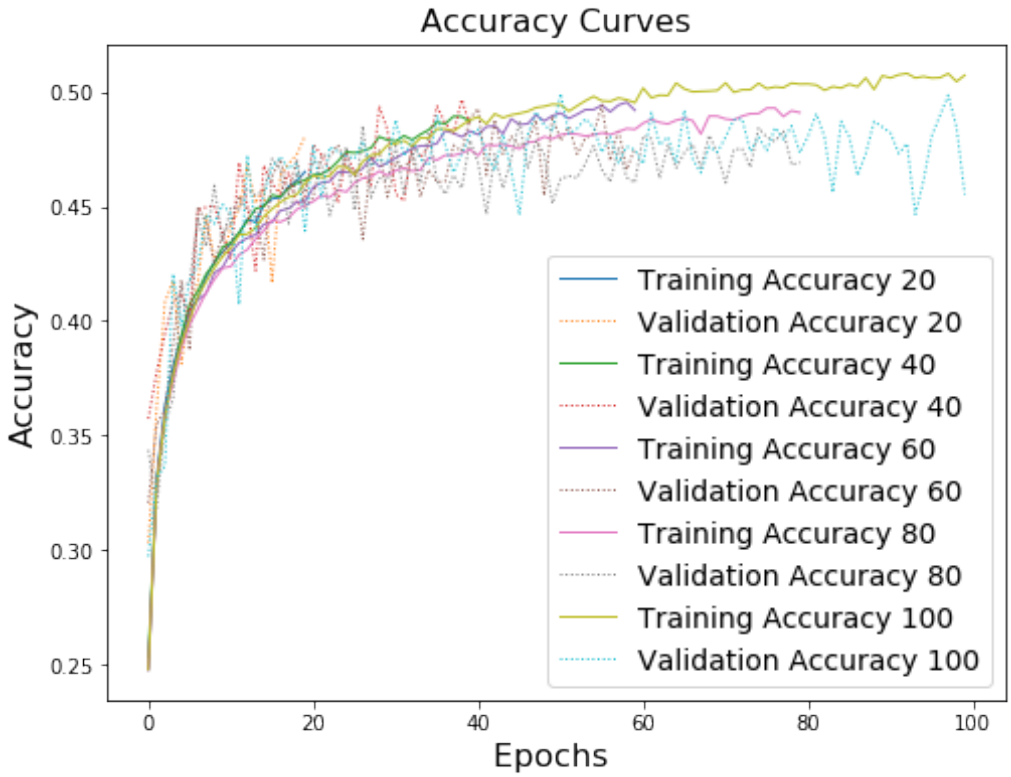

Figure 2: Epoch Experiment Accuracy Curves

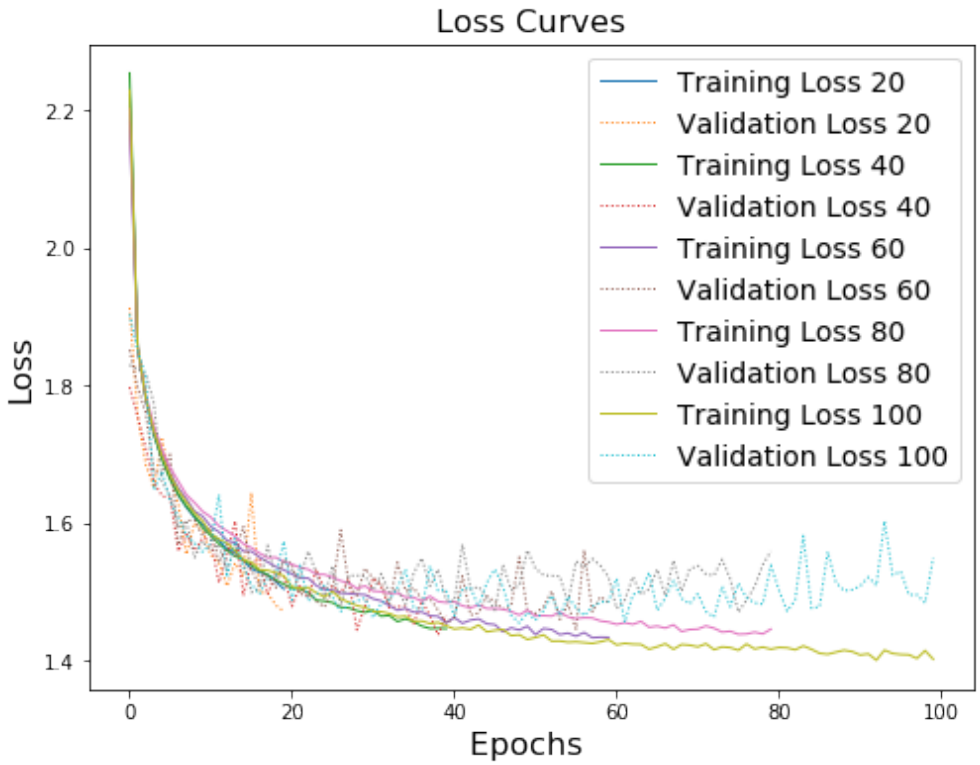

Figure 3: Epoch Experiment Loss Curves

2) Batch Size

For the batch size experiment, I tested doubling the size four times. 512 images per batch gave the highest test accuracy and lowest loss, and the plots do not indicate overfitting.

Table 2: Batch Experiment

Batch Size	*Test Accuracy*	*Test Loss*
128	0.4746	1.483835472
256	0.49	1.426200302
512	0.5002	1.410775115
1024	0.4719	1.467798519
2048	0.4642	1.532059594

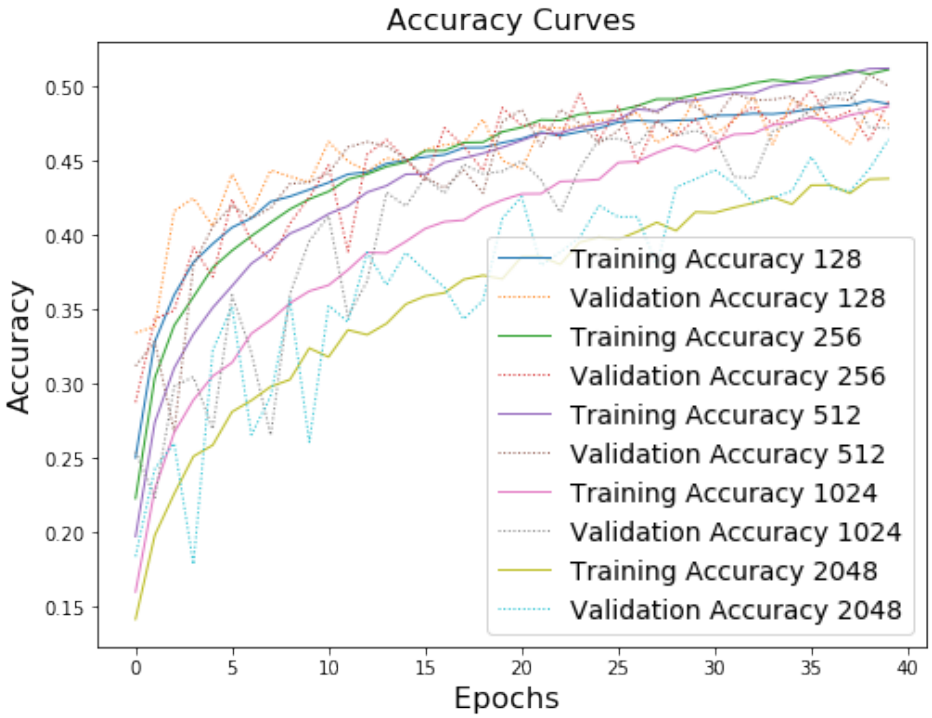

Figure 4: Batch Size Experiment Accuracy Curves

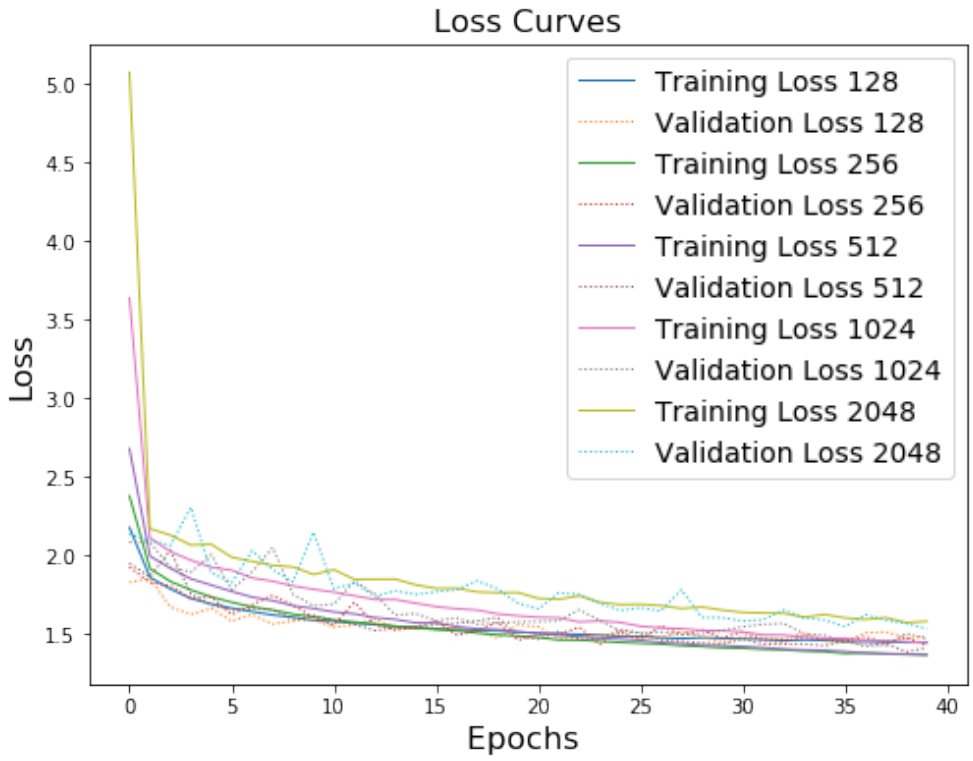

Figure 5: Batch Size Experiment Loss Curves

3a) Architecture - Number of neurons

I tested halving and doubling the number of neurons, and also having a compression of the data from the input layer to the hidden layer. 1024 neurons in the input layer and compressing to 512 in the hidden layer gave the second highest accuracy and the lowest loss. Although having 1024 neurons in the hidden layer had a marginally higher accuracy, it also was deviating slightly from the training accuracy, suggesting it may be overfitting, though at the scale of the difference it may have been noise. I thus selected 1024 neurons for the input and 512 for the hidden layer, but experimented more with this in part 3b.

Table 3: Neuron Experiment

Number of Neurons	*Test Accuracy*	*Test Loss*
Input: 512 Hidden: 512	0.5068	1.404678665
Input: 512 Hidden: 256	0.5009	1.415665766
Input: 1024 Hidden: 1024	0.5196	1.386047803
Input: 1024 Hidden: 512	0.5158	1.374123655
Input: 256 Hidden: 256	0.4712	1.489402296

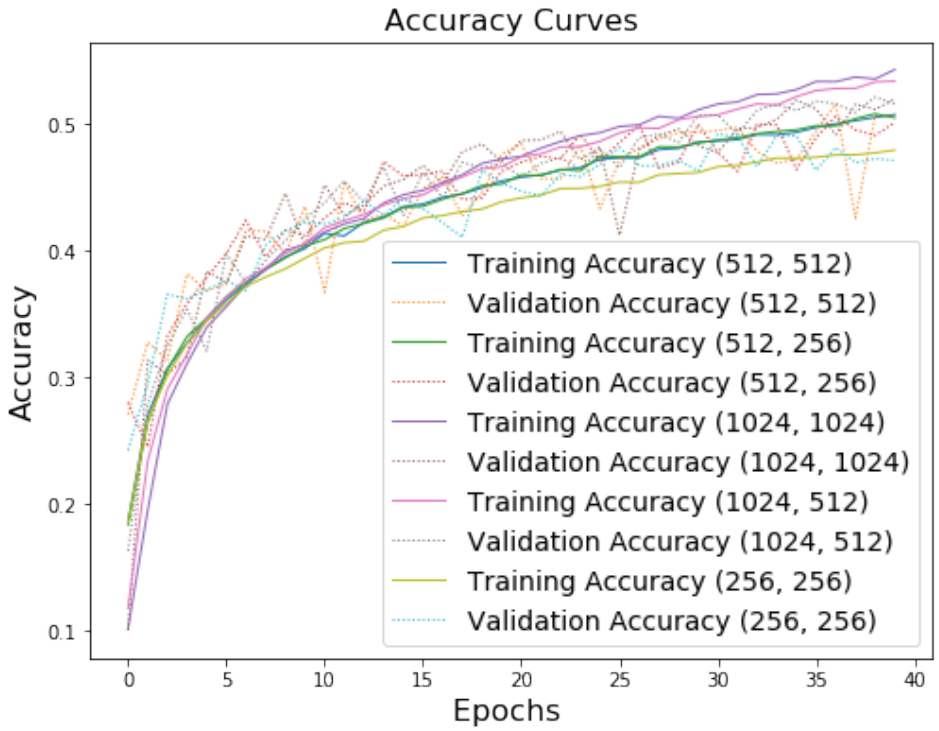

Figure 6: Neurons Experiment Accuracy Curves

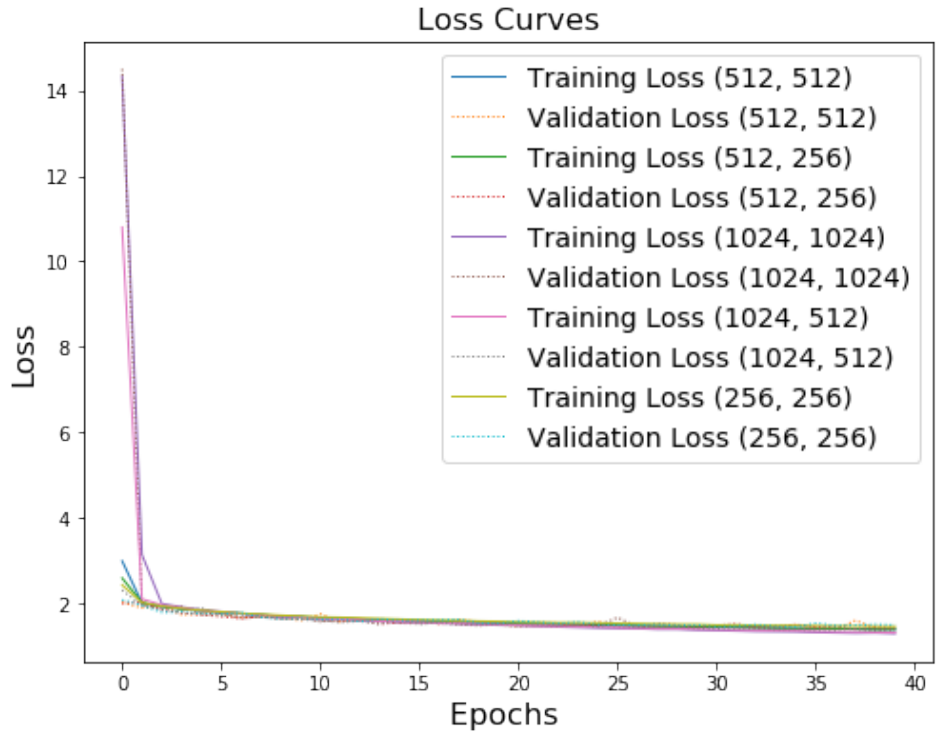

Figure 7: Neurons Experiment Loss Curves

3b) Architecture - Number of layers

I tested adding more hidden layers as well as continuing with the data compression between hidden layers. Table 4 summarizes the accuracy and loss scores, and figures 8 and 9 show the

corresponding plots. Having two hidden layers, the first with 512 neurons and the second with 256 gave the highest accuracy and the lowest loss, and is the architecture I have selected for the remaining experiments.

Table 4: Hidden Layer Experiment

Number of Neurons	*Hidden Layers*	*Test Accuracy*	*Test Loss*
Hidden: 512	1	0.5005	1.431460875
Hidden: 512, 512	2	0.4891	1.451796052
Hidden: 512, 512, 512	3	0.5036	1.40284867
Hidden: 512, 256	2	0.5078	1.392914082
Hidden: 512, 256, 128	3	0.5018	1.3962129

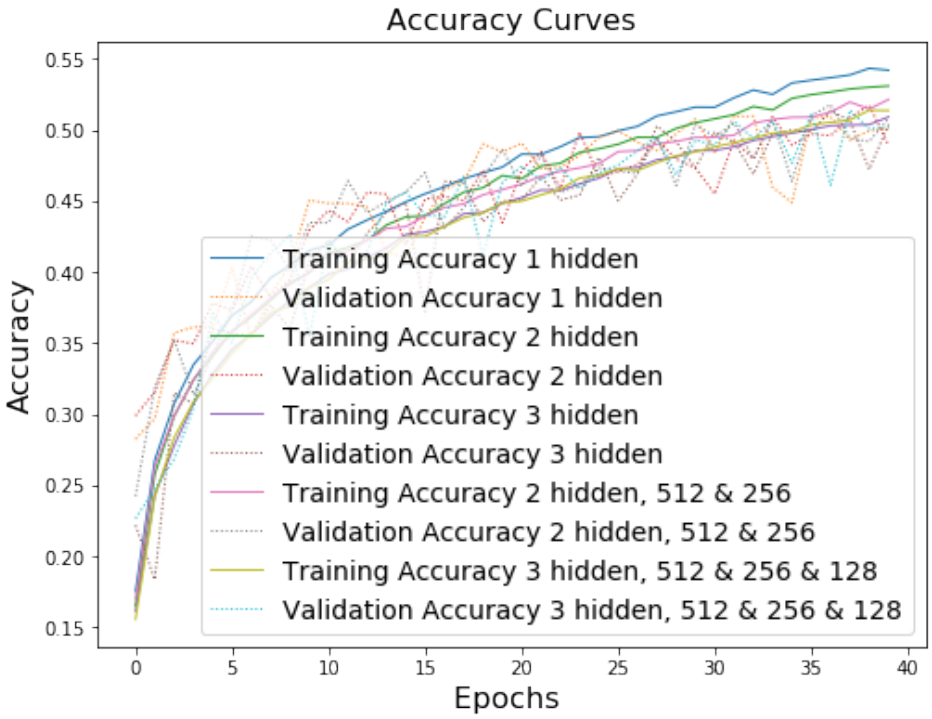

Figure 8: Layer Experiment Accuracy Curves

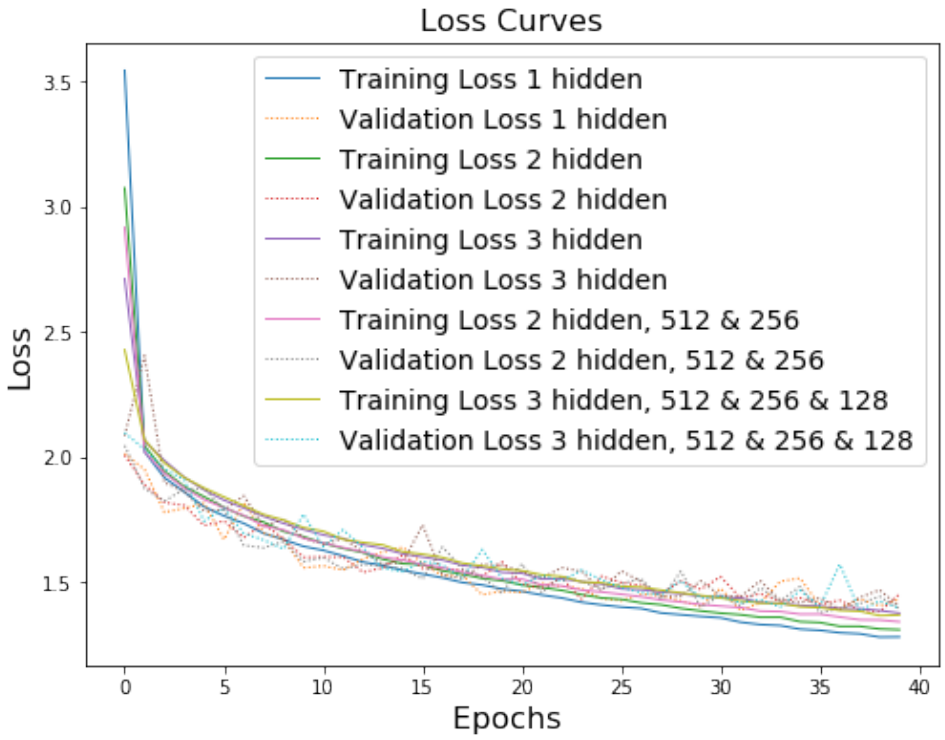

Figure 9: Layer Experiment Loss Curves

4) Learning Rate

For the learning rate I experimented with the range 0.0001 to 0.05. I found that somewhere between 0.001 and 0.005 the learning rate becomes too high and the model fails to train. A learning rate of 0.0005 gave the highest accuracy and the lowest loss, however the accuracy plot indicates that it is overfitting. I therefore selected 0.001 as the learning rate which did not show signs of overfitting.

Table 5: Learning Rate Experiment

Learning Rate	*Test Accuracy*	*Test Loss*
0.0001	0.5177	1.391332731
0.0005	0.52	1.368082947
0.001	0.5135	1.387983605
0.005	0.1	14.50628565
0.01	0.1	14.50628569
0.05	0.1	14.50628569

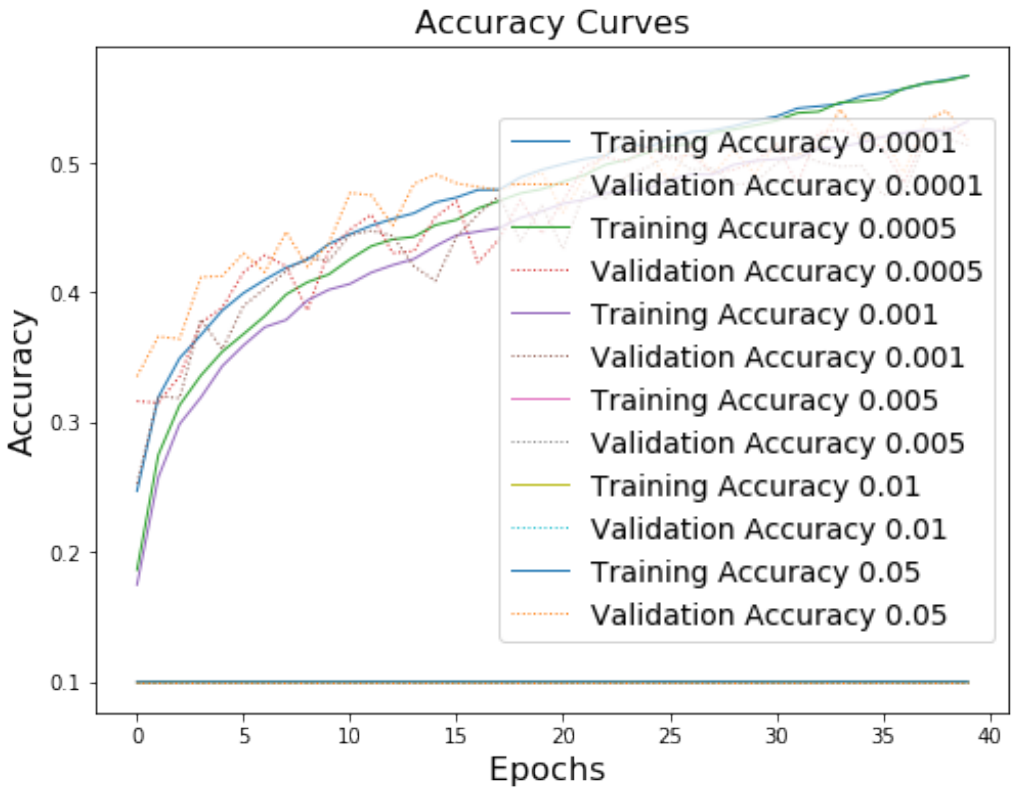

Figure 10: Learning Rate Experiment Accuracy Curves

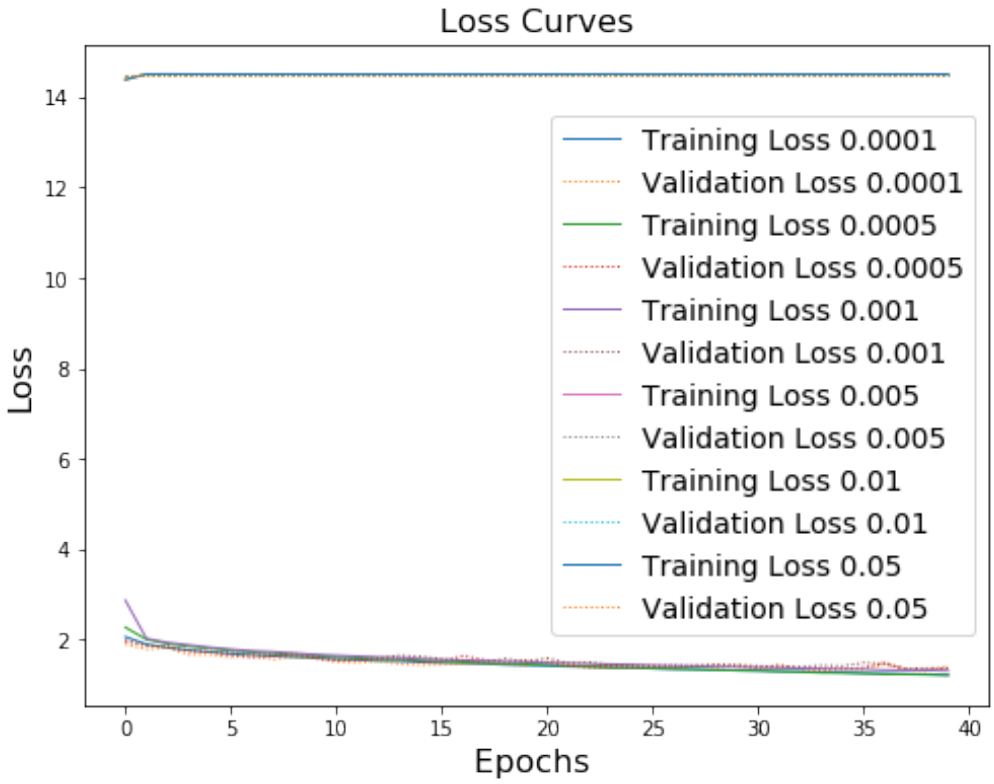

Figure 11: Learning Rate Experiment Loss Curves

5) Activation Function

In testing the available activation functions, I ran two experiments. The results are summarized in Table 6. In the first experiemnt, I tested using each function in the output layer. I determined that softmax is the optimal function for this layer. Softplus had slightly better accuracy and loss but was overfitting. For the second experiment, I tested each function in the two hidden layers while using softmax in the output layer. Similar to the first experiment, relu did not have the best scores but those that scored better were overfitting.

Due to the large number of activation functions, in generating the plots for each experiment I first looked at the training accuracy to identify the ones that failed to train and then only plotted the other metrics for those still under consideration.

Table 6: Activation Function Experiment

Activation Function	*Output Layer Test Accuracy*	*Output Layer Test Loss*	*Hidden Layers Test Accuracy*	*Hidden Layers Test Loss*
softmax	0.521	1.379556113	0.1994	1.998002326
elu	0.3701	4.4490403	0.5154	1.387187471
selu	0.4225	1.74359893	0.1	14.50628569
softplus	0.5261	1.358416271	0.5116	1.379933249
softsign	0.1	1.19E-07	0.509	1.413390068
relu	0.45	1.632549664	0.5108	1.386254755
tanh	0.1	1.19E-07	0.4882	1.424028186
sigmoid	0.1	1.19E-07	0.4815	1.472179993
hard_sigmoid	0.1	13.03310562	0.5286	1.346140686
exponential	0.5098	1.407615144	0.1	1.19E-07
linear	0.1	8.059047767	0.1	14.5062857

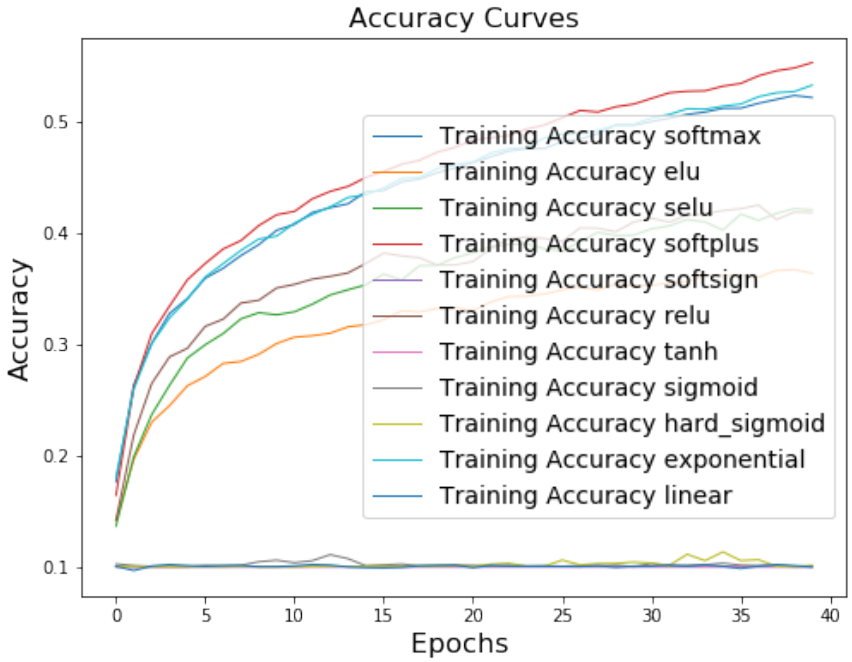

Figure 12: Activation Function Output Layer Experiment Accuracy Curves – Full

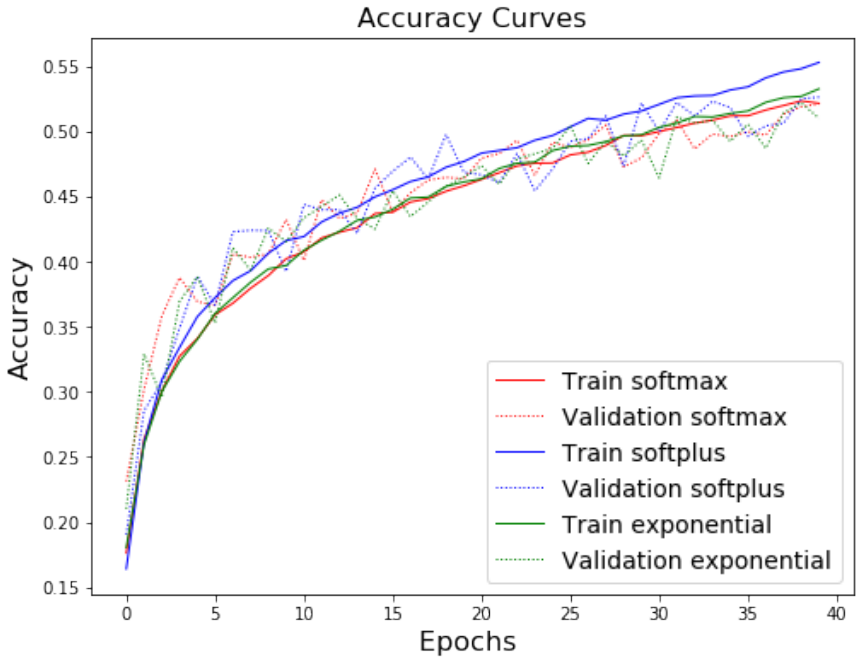

Figure 13: Activation Function Output Layer Experiment Accuracy Curves - Limited

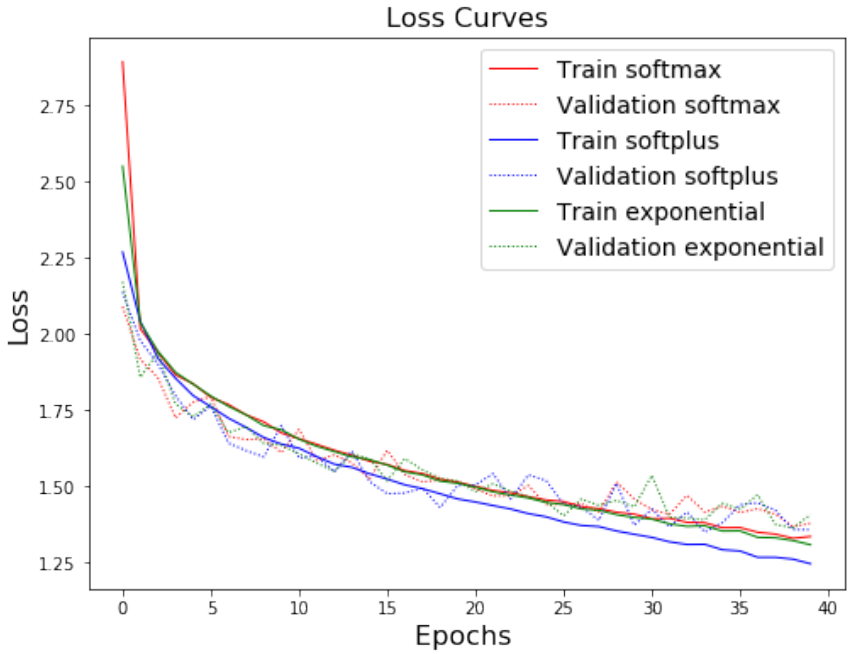

Figure 14: Activation Function Output Layer Experiment Loss Curves - Limited

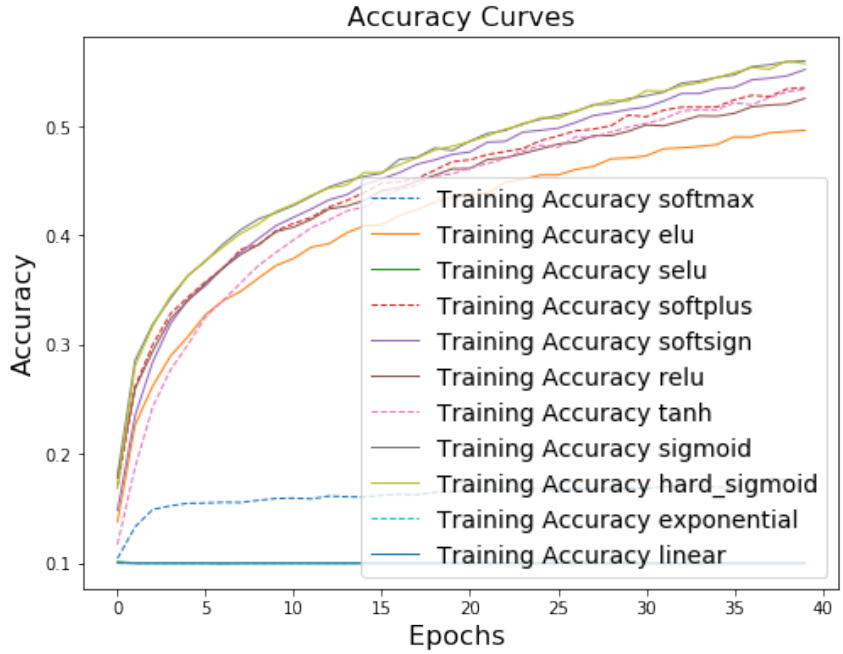

Figure 15: Activation Function Hidden Layer Experiment Accuracy Curves - Full

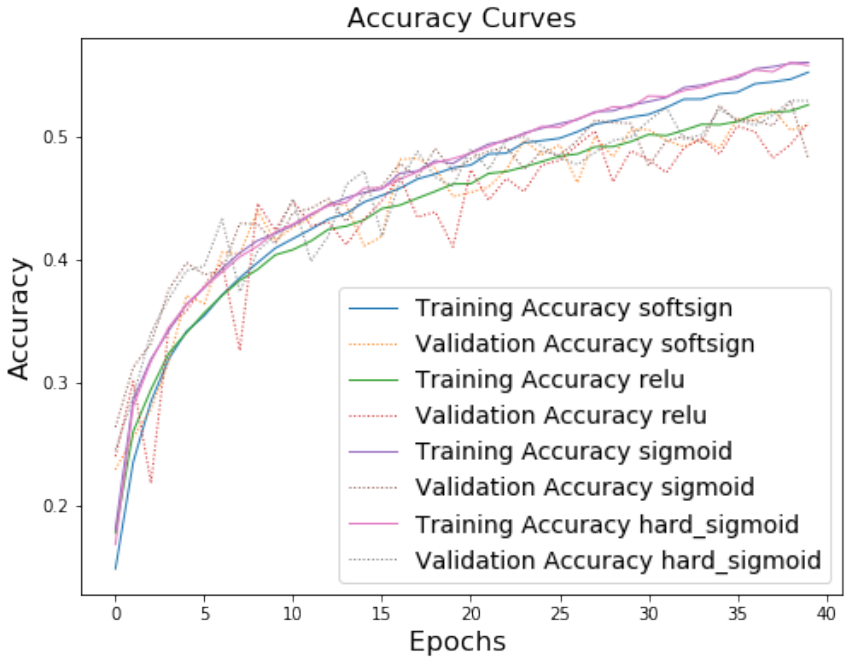

Figure 16: Activation Function Hidden Layer Experiment Accuracy Curves – Limited

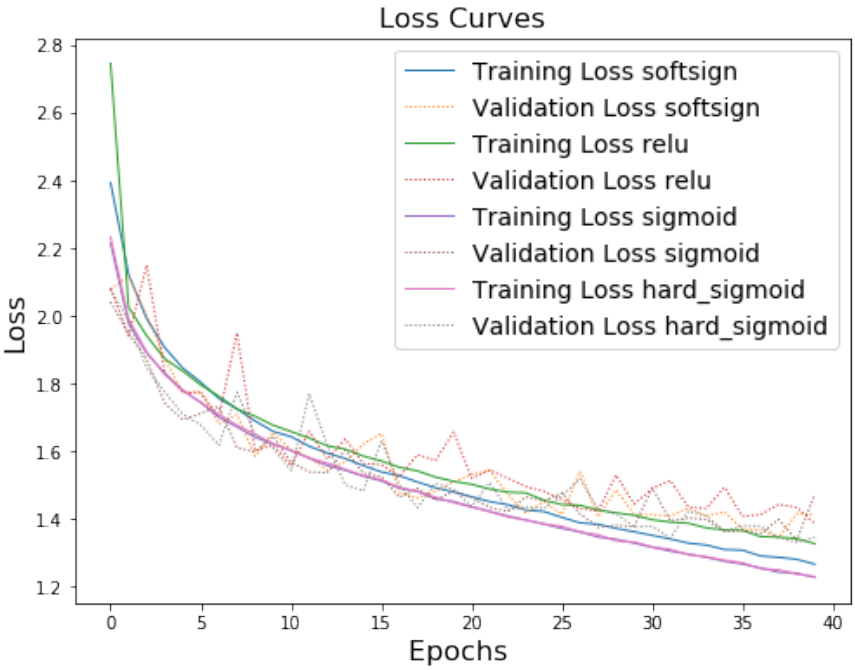

Figure 17: Activation Function Hidden Layer Experiment Loss Curves - Limited

6) Dropout Rate

To test changing the dropout rate, I first ran an experiment of removing the dropout entirely. The results are shown in figure 18. The model rapidly overfits with the test accuracy and loss failing to improve after about 20 epochs. I then experimented with increasing the dropout rate in increments of 0.05 up to 0.3. A dropout of 0.2 gave the highest accuracy and lowest loss, and did not show signs of overfitting.

Table 7: Dropout Rate Experiment

Dropout Rate	*Test Accuracy*	*Test Loss*
0	0.4967	1.810473363
0.05	0.5118	1.491578912
0.1	0.4942	1.493410328
0.15	0.5114	1.398903719
0.2	0.518	1.366844919
0.25	0.4945	1.424665911
0.3	0.4899	1.455182697

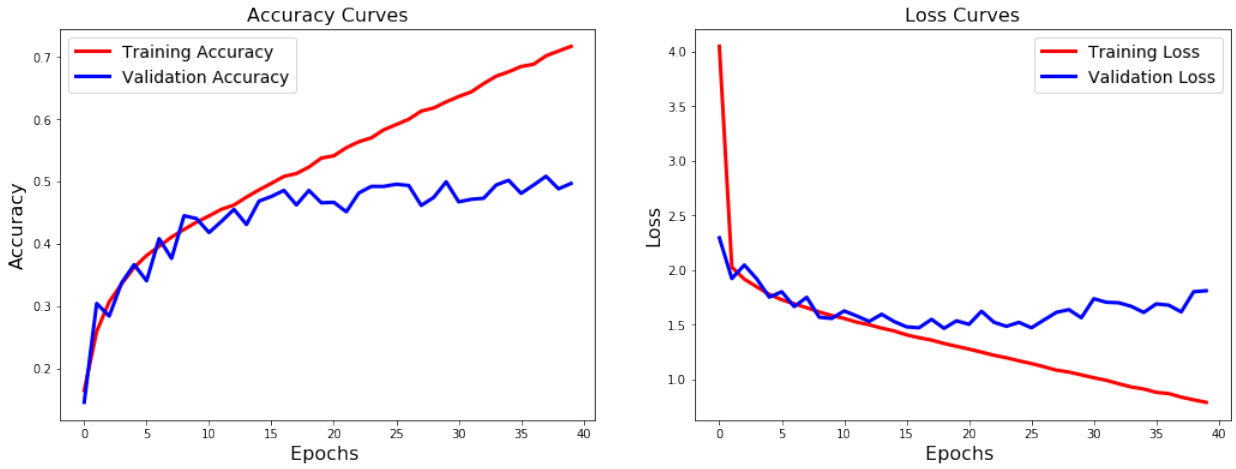

Figure 18: No Dropout

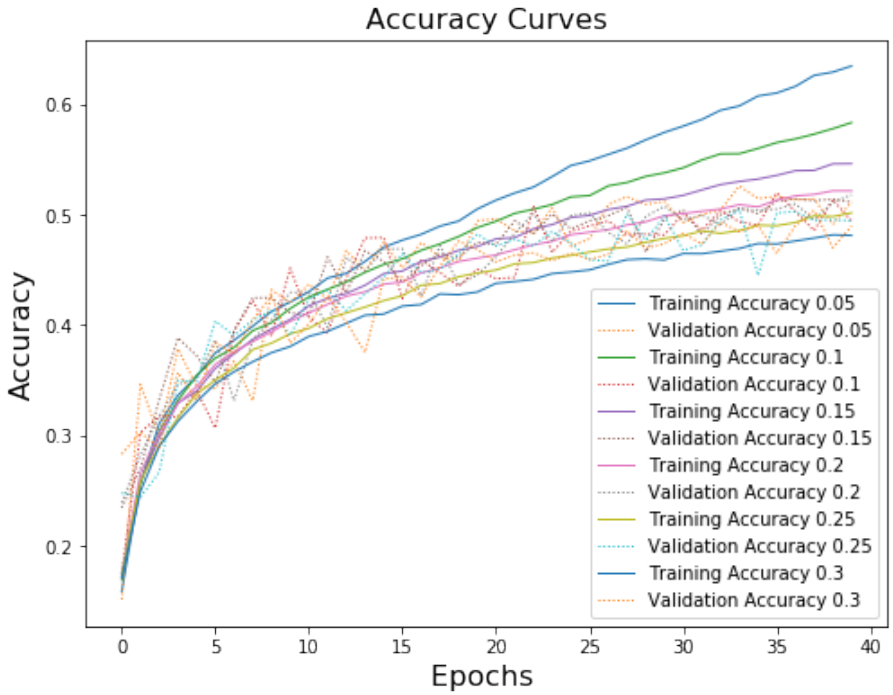

Figure 19: Dropout Rate Experiment Accuracy Curves

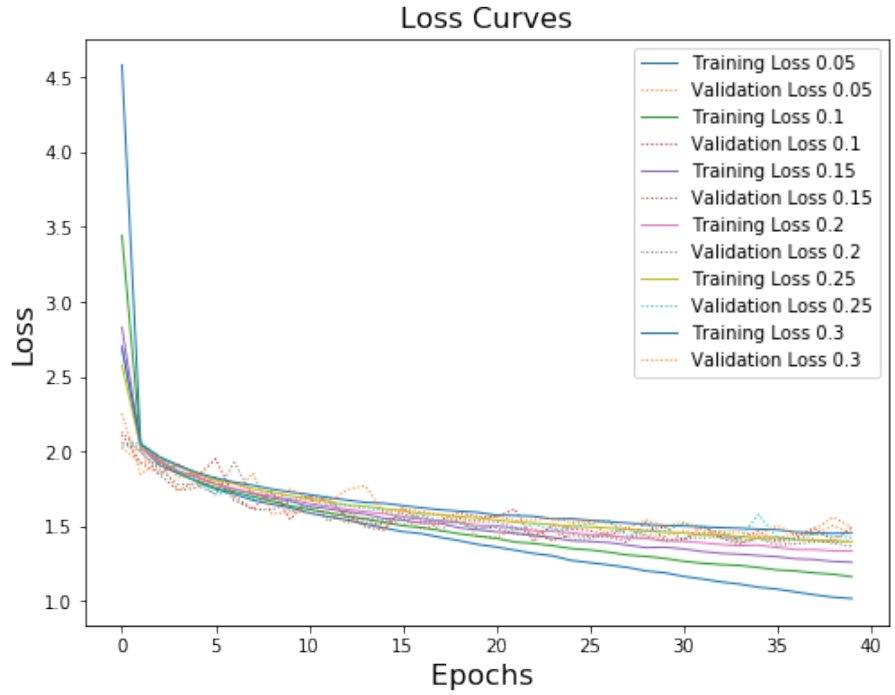

Figure 20: Dropout Rate Experiment Loss Curves

7) CPU vs GPU

As an additional experiment, I ran a comparison of training the model on a CPU versus a GPU. The differences in accuracy and loss are within the margin I would attribute to noise, however the CPU took 5x longer to train. This time difference is influenced by both the hardware and the

model architecture. In Part 2 my team found that the CNN took 6.625x longer to train on the same CPU/GPU pairing.

Table 8: Hardware Experiment

Hardware	*Test Accuracy*	*Test Loss*	*Runtime*	*Rate*
CPU	0.4971	1.398227687	319.6150	158 us/step
GPU	0.5037	1.411879149	60.9640	31 us/step

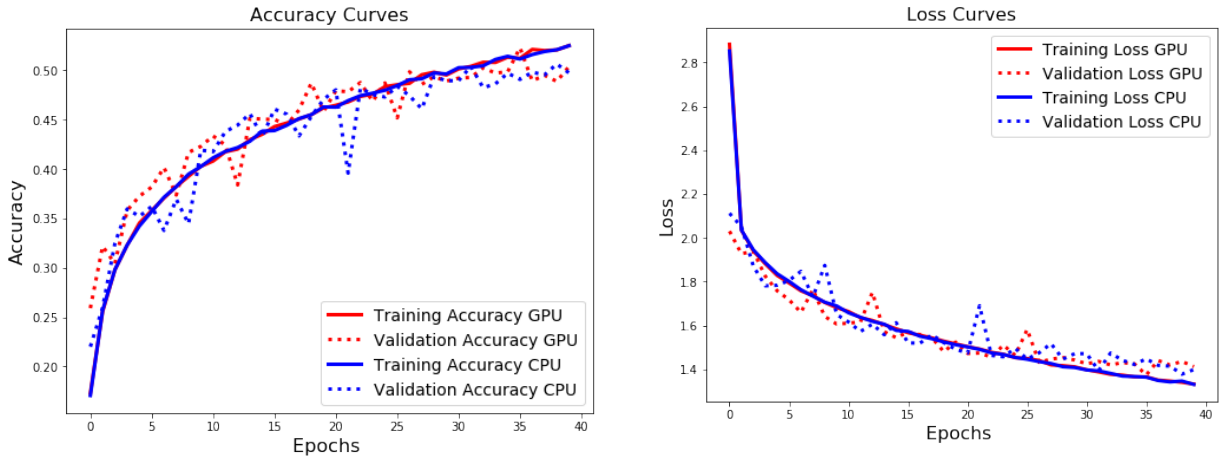

Figure 21: CPU vs. GPU Accuracy and Loss Curves

Conclusion

Based on my experimental results, I recommend an architecture of:

- 40 epochs
- batch size of 512
- Input layer: 1024 neurons, relu activation
- 1st hidden layer: 512 neurons, relu activation
- 2nd hidden layer: 256 neurons, relu activation
- Output layer: 10 neurons, softmax activation
- Learning rate of 0.001 with the RMSprop optimizer
- Dropout rate of 0.2 for the input and hidden layers

As shown in table 9, this architecture brings an average 10% improvement in the accuracy and 8% improvement in the loss over the starting architecture. These numbers were obtained by averaging the accuracy and loss across all runs of the same architecture, as the starting configuration was run twice and the final one was run seven times across the various experiments.

Figure 22 shows the plots of the starting and final models, and it is clear that the final model is not overfitting. This is because throughout the experiments, I selected parameters that gave the best accuracy and loss without overfitting. When I selected a value that did not give the

best scores, the difference between the best scoring ones and the ones selected was always within a range that it may have been noise and the gains of selecting those other parameters would have been very small relative to the effect they had on the fit.

That being said, the final accuracy is still only 51%. By comparison, the convolutional neural network used in part 2 gave us a 76% accuracy without any parameter tuning. It is possible that with a different optimizer and further tuning some of the parameters within more narrow ranges the MLP architecture might be further improved, but it unlikely these changes would increase the accuracy by another 25 percentage points. The CNN is thus a more appropriate type of model for classifying the CIFAR-10 data set.

Table 9: Final Results

Model	*Mean Test Accuracy*	*% Change*	*Mean Test Loss*	*% Change*
Starting Architecture	0.46465	10%	1.504798293	-8%
Final Architecture	0.510271429		1.38909433	

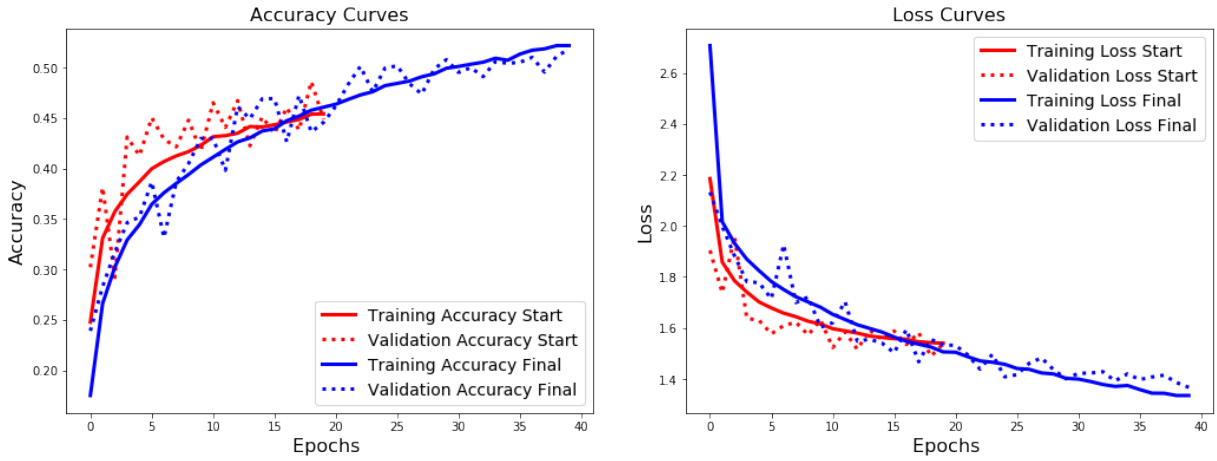

Figure 22: Final Results