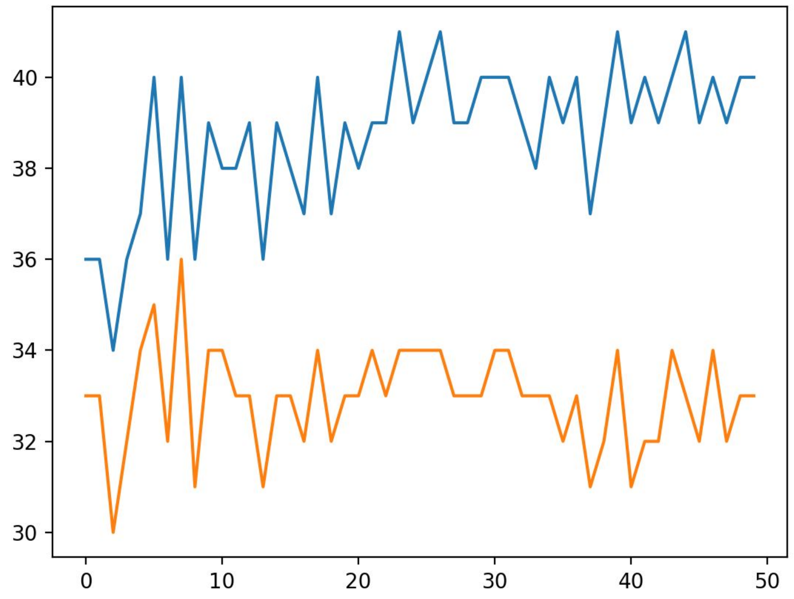
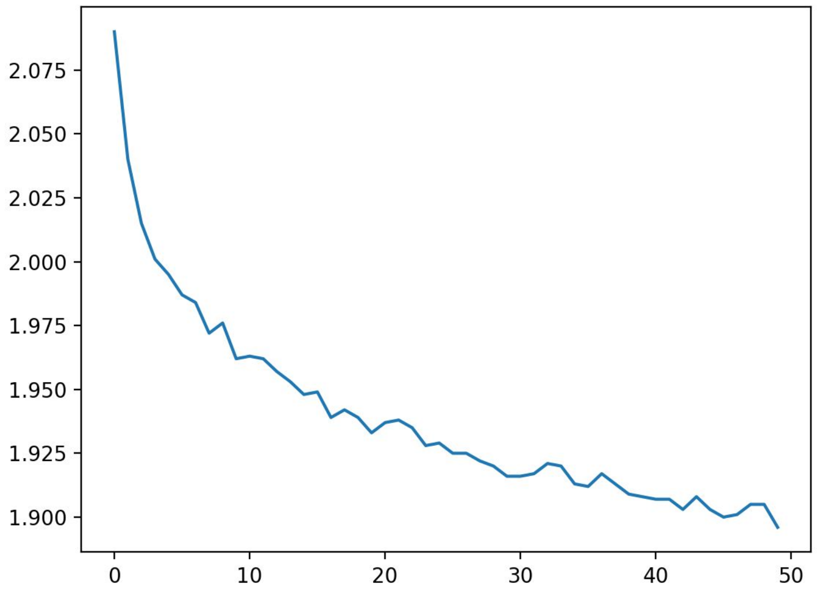
2.1 LazyNet: Training a classifier using only one fully connected layer

This took about 30 minutes to run and it doesn't do that well, worse than my model from the last homework assignment. There's a lot of variance in both training and testing accuracy. Training accuracy seems to have an upward trend before plateauing while testing accuracy seems to fluctuate around a constant (say 32 or 33). The loss function is also very bumpy. The poor accuracies and bumpy loss graph is probably due to all the noise of the inputs.

Accuracy and Loss graphs for LazyNet:

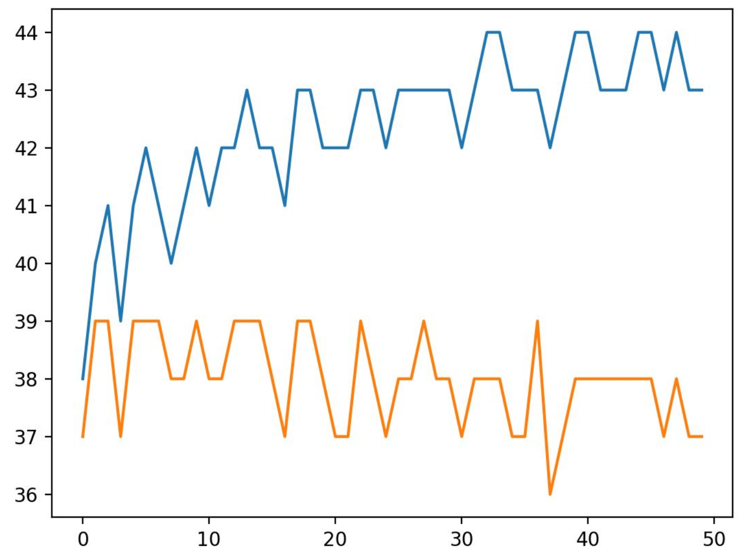


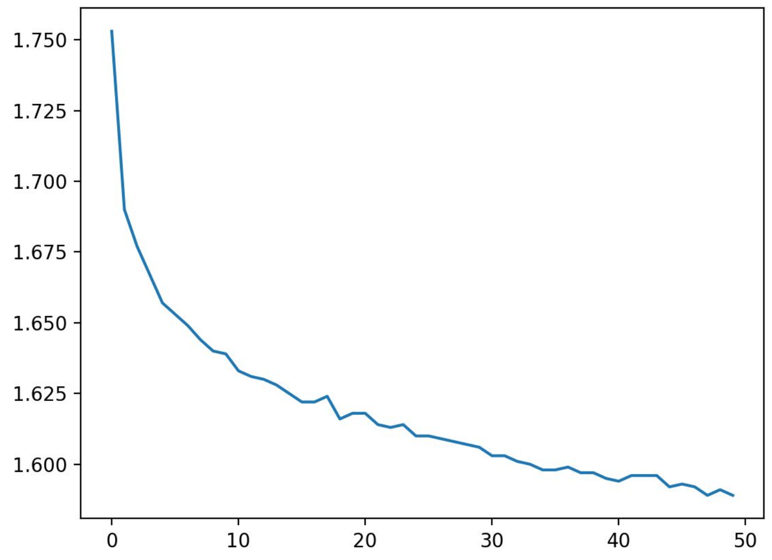


2.2 BoringNet: Training a classifier using multiple fully connected layers

This took about an hour to run. Similar trends to LazyNet but with more plateauing spikes instead of spikey spikes. Loss function is also a bit smoother. It also improved accuracies slightly. Training accuracy ended up around 43 while testing data seemed to fluctuate around 38 and decreased slightly to end up between 38-37. It looks like the model is starting to overfit since training accuracies increased towards the end and testing accuracies started to slightly decrease. The fluctuations and bumpiness is still probably due to the noise of the inputs.

Accuracy and Loss graphs for BoringNet:

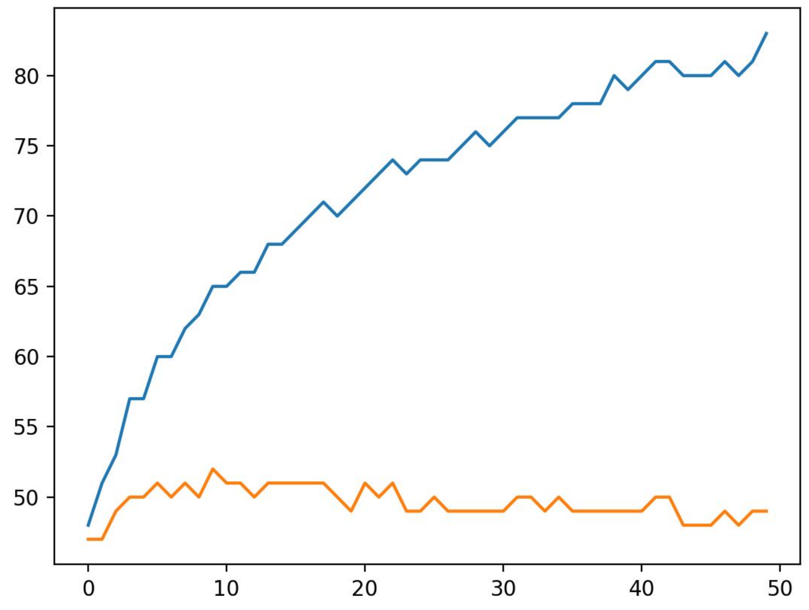


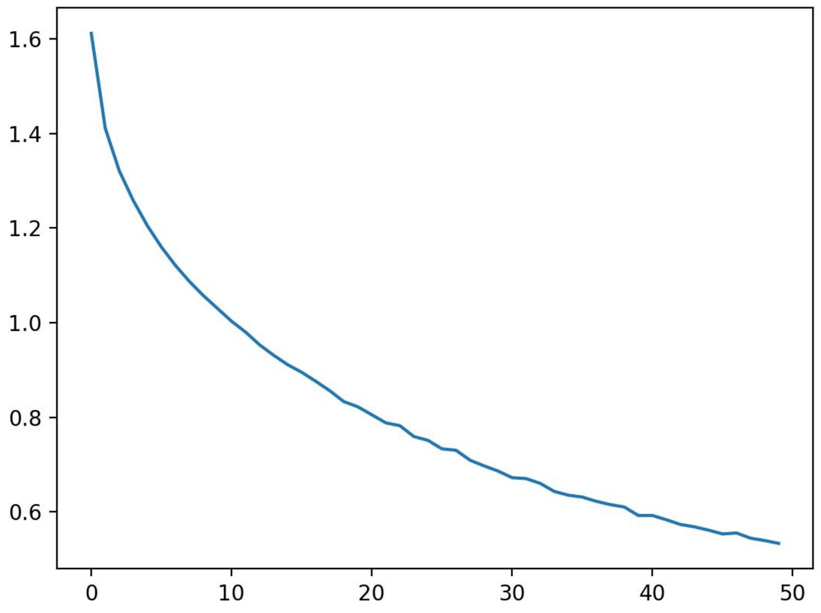


2.2.1 Try training this model with and without activations. How does the activations (such as ReLU) affect the training process and why?

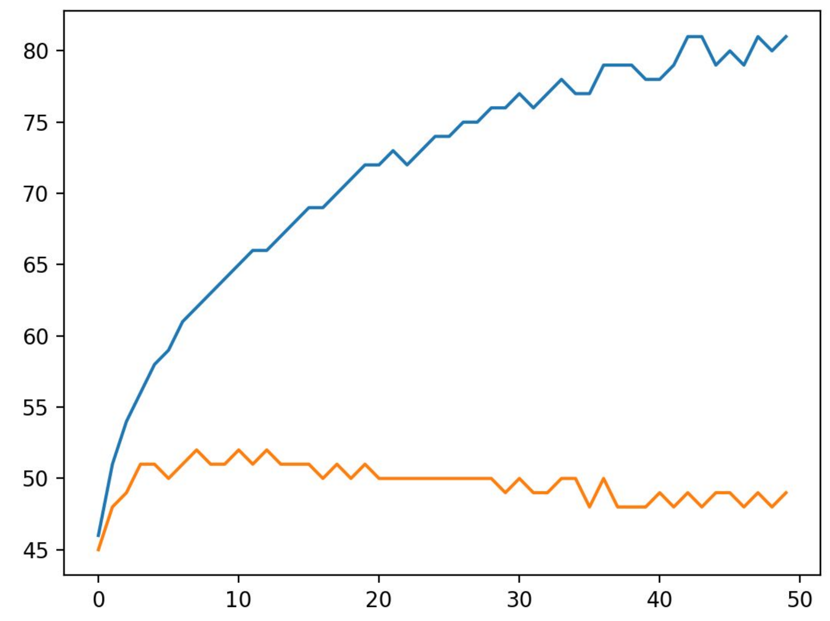
I tried using both ReLU and LReLU activations just to make sure results were consistent. Each model took around 1 hour to run. I only activated the first two layers and left the last one without because that's how the example on pytorch used activations. With activations, both training and testing accuracies improved but training accuracies improved much more so exaggerating the overfitting trend we saw in BoringNet without activation. This is probably due to the activation layers making the inputs more similar to each other and reducing noise.

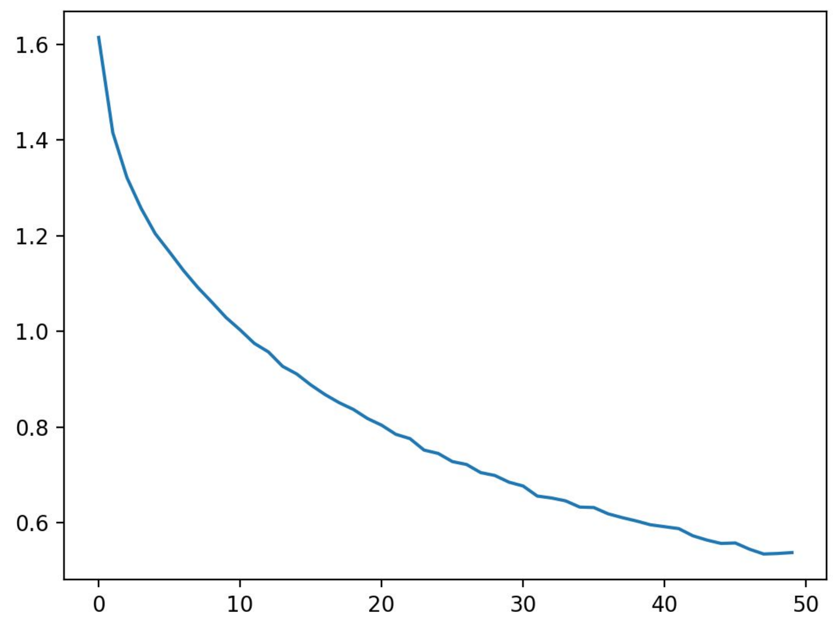
Accuracy and Loss graphs for BoringNet with ReLU:





Accuracy and Loss graphs for BoringNet with LReLU:





2.3 CoolNet: Training a classifier using convolutions

Originally, I was just going to try one CoolNet, which is basically the example on pytorch but with LReLU activations instead of ReLU because I found from last homework that my model with all LReLU activations resulted in the best accuracies. However, I was naive and ambitious and unaware how much of a headache running the rest of the assignment was going to be. So, I made a script that was supposed to run while I slept to try a bunch of variations of the example from pytorch (eg using all ReLU activations, no activations for the convolution layers, tried doing one convolution layer, tried to adjust the input output size in between layers, tried to adjust kernel size for convolution layers, etc). I stuck to just trying ReLU or LReLU activations and the same pattern as pytorch because damn there are so many activations and I didn't want to do too much debugging/understanding pytorch documentations (sorry I am not an overachiever I just want to stay sane and graduate).

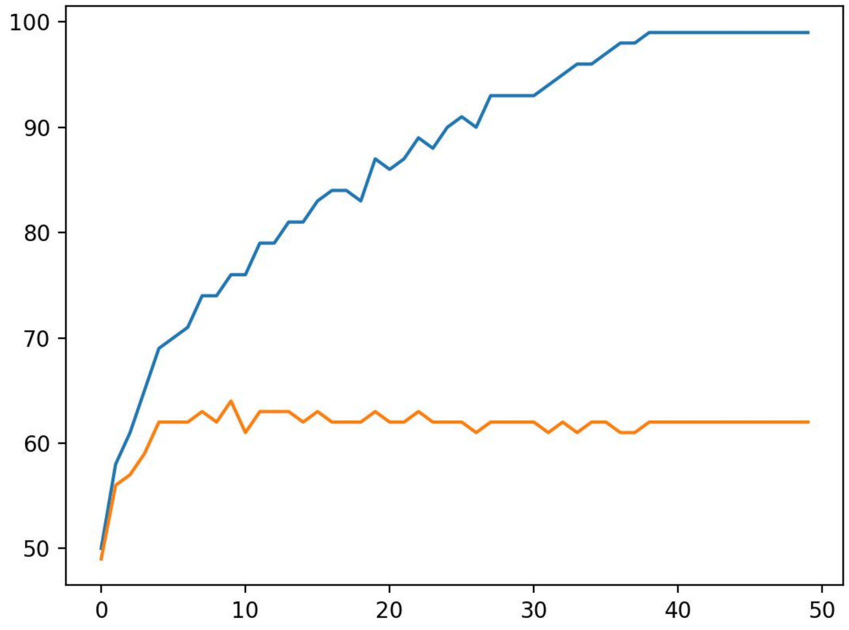
However, I fucked up and only one ran that night (RIPperoni), so the next day I fixed my error and ran each with 30 epochs, looking hella dumb carrying an open laptop with me everywhere. After that ran, I found that some didn't work, and I wasn't about to try and fix them because one model with 30 epochs still took about 45 minutes to run, so I just looked at the graphs that did and picked a few that looked promising against my all LReLU activations model (the only one that actually ran the night before). Then, I took the top three promising ones and ran those for 50 epochs.

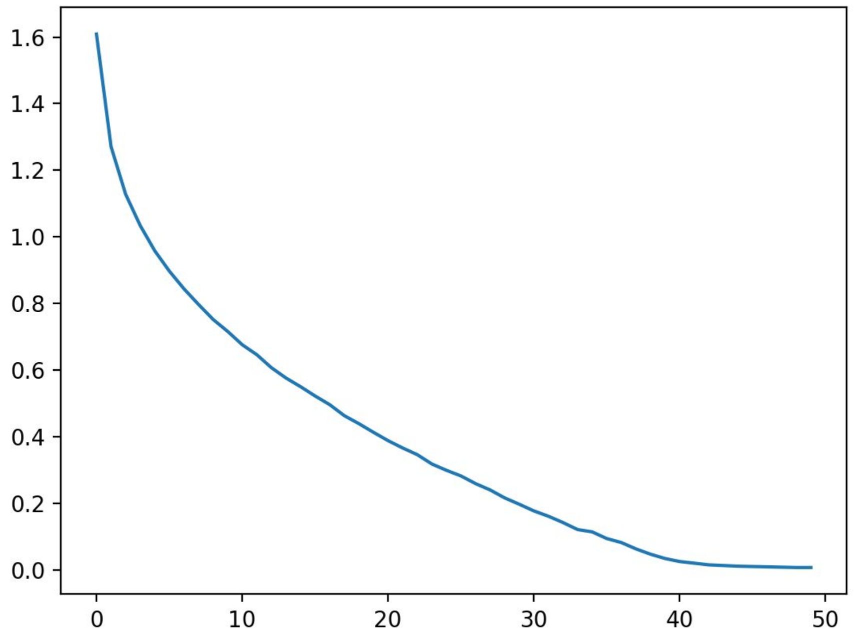
I ended up choosing CoolNet_v7 to become CoolNet_final because they all ended up with around the same testing accuracy, but CoolNet_v7 ended with the smallest training accuracy, so I figured it was the least overfitted which is promising. I then realized that CoolNet_v7 used ReLU activations and I remembered my LReLU model from the last homework, and still kind of naive about the pain to come and against my better judgement, I was like fuck it let's try CoolNet_v7 but with LReLU and call it CoolNet_v10. CoolNet_v10 ended up getting 99% on training accuracy and ended with 1% more for testing accuracy, but in earlier accuracies, CoolNet_v10 had higher accuracies out of all the models I tried. So, again, against my better judgement, I said fuck it I like seeing 99% and 100% and it reached some great peaks for early testing accuracies and maybe it'll end up with better accuracies further down the assignment...so CoolNet_v10 became CoolNet. And that's how I picked my layers. CoolNet_v7 and CoolNet_v10 are basically the example on pytorch without activations on the convolution layers. I ended up getting worried about my choice after seeing the learning rate graphs for CoolNet_v10, so I ended up going back and doing learning rate for v7 and I did the rest of the assignment with both CoolNet_v7 and v10 (which was not cool especially the god awful amount of time it took and since pytorch would crash if I did anything else on my laptop but I'm a doubtful/neurotic person so I had to). So, below are graphs for CoolNet_v7 and CoolNet_v10.

All the models took about 1 hour and 15 minutes to run. All models smoothed out the loss graph (some more than others) and all of them overfitted with a drop in testing accuracies towards the later epochs. They all pretty much had similar overall shapes Most of the top that I chose to try with 50 epochs had training accuracies that converged at 81% and testing accuracies that ended at 61%. All of them also had pretty similar shapes, so that gave me many ragrets and made me feel that I should have just gone with v1 and justified it without trying other models, but oh well.

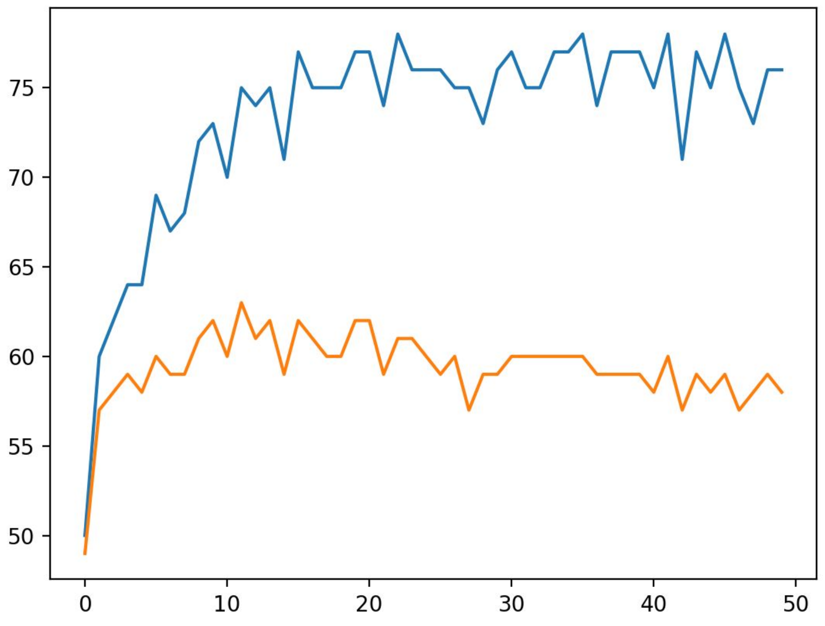
I won't attach graphs for my other CoolNet versions, but they are in my graphs folder. CoolNet_v1 is the one that successfully ran during the night, the ones that were chosen to move on to try with 50 epochs from the initial versions are CoolNet_v3, v4, and v7.

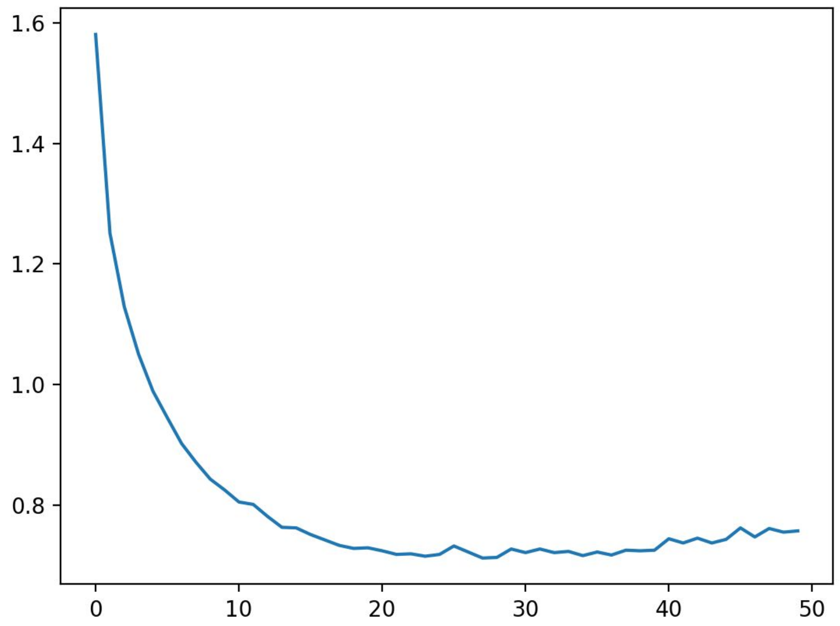
Accuracy and Loss graphs for CoolNet_v10:





Accuracy and Loss graphs for CoolNet_v7:



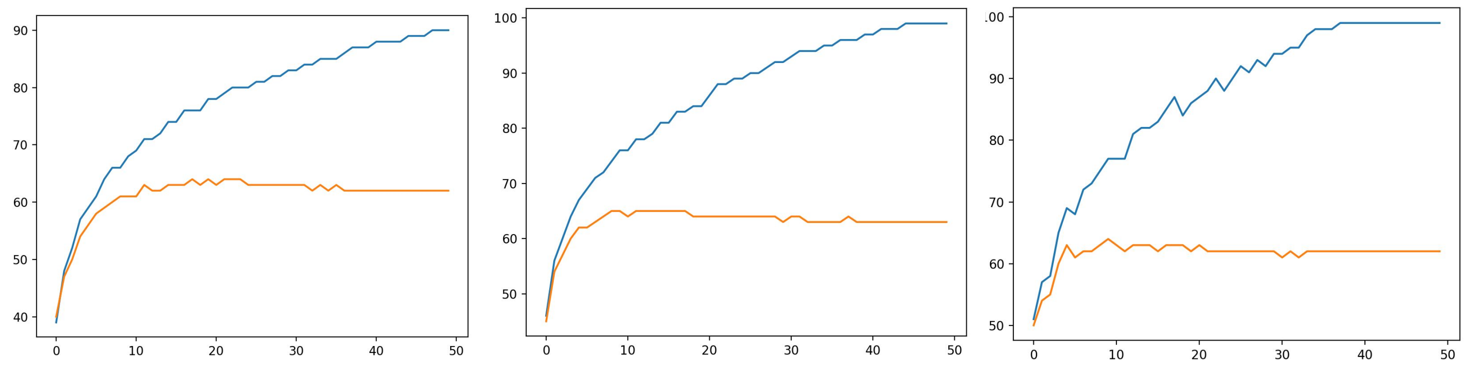


Not sure why v7 was so much more bumpy.

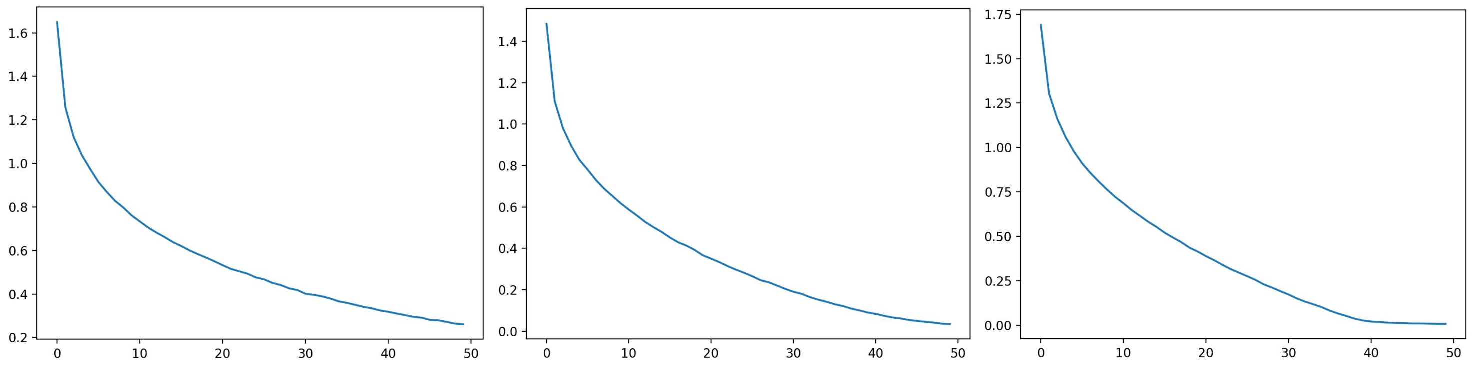
2.3.1 How does varying batch size values affect CoolNet training and why?

I chose to test batch sizes 20, 10, and 5.
I noticed that larger batch sizes resulted in a quicker run (under 1 hour instead of over). I noticed for v10, a small batch size made my accuracies converge faster. The loss changes a little bit but subtly enough that I can’t really put a finger on how it changed (doesn’t help that the y-axis scale changes for each) , but I think a larger batch size resulted in steeper/curvier/more exponential like curve. They all end up with about the same loss but start at different amounts of loss. For accuracies, they all end up with about the same testing accuracy but batch size of 20 ends up with a lower training accuracy for some reason. Batch size is the number of points used for an update. It makes sense that for a smaller batch size it would converge faster because less points are used for the update. I’m not sure how to interpret the loss though because they look super similar to me.

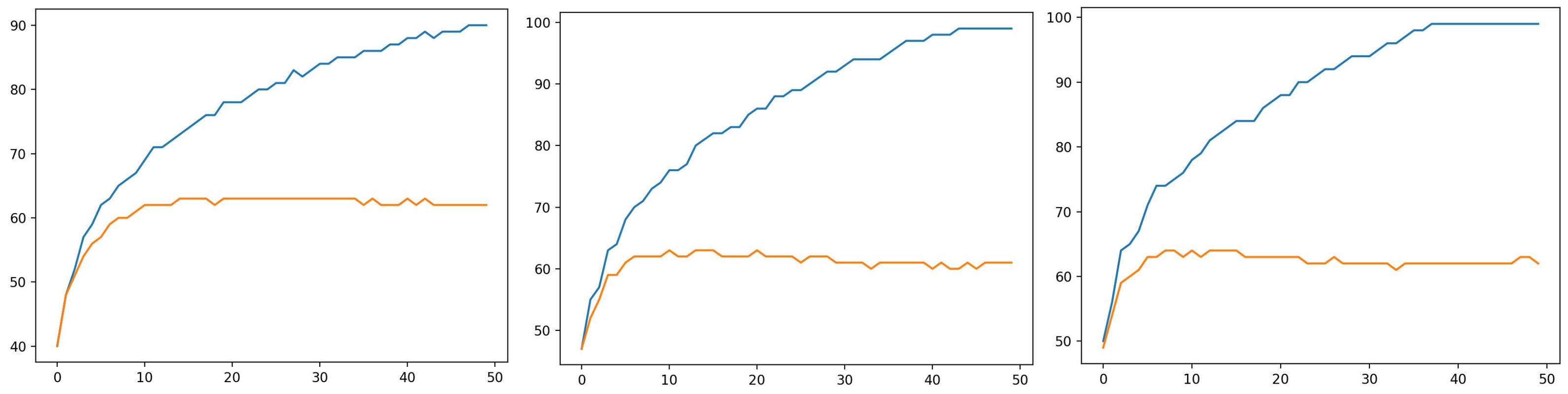
Accuracies for CoolNet_v10 for batch size 20, 10, and 5:



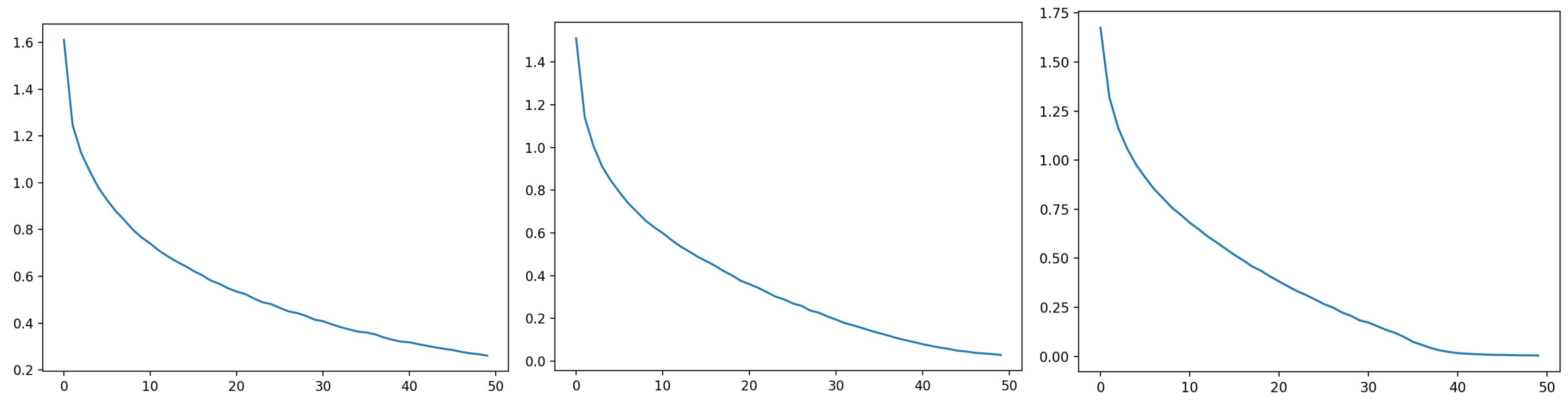
Loss for CoolNet_v10 for batch size 20, 10, and 5:



Accuracies for CoolNet_v7 for batch size 20, 10, and 5:



Loss for CoolNet_v7 for batch size 20, 10, and 5:

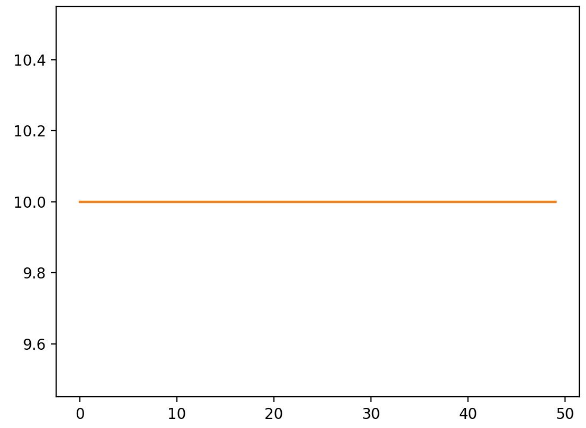
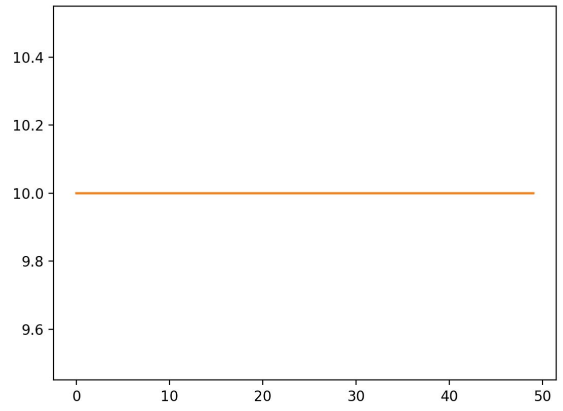
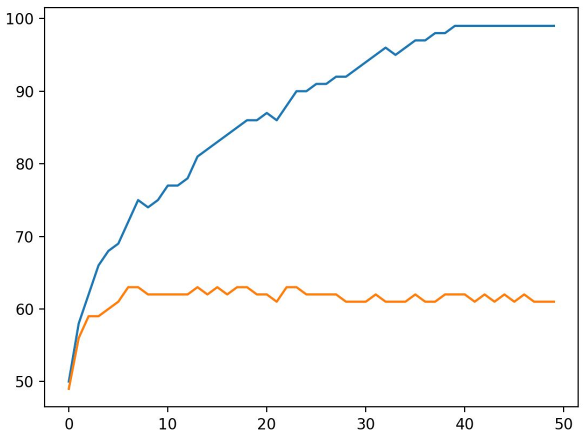
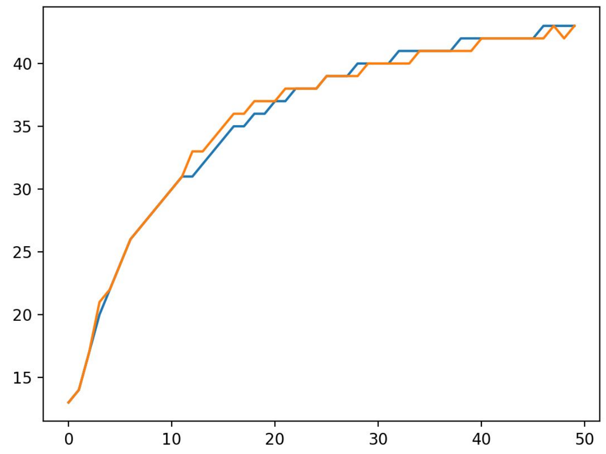


I’m not sure why training accuracies got so high for CoolNet_v7 especially since default batch size was 4...Probably misnamed a log or something, but oh well because it’s too late to re run this one.

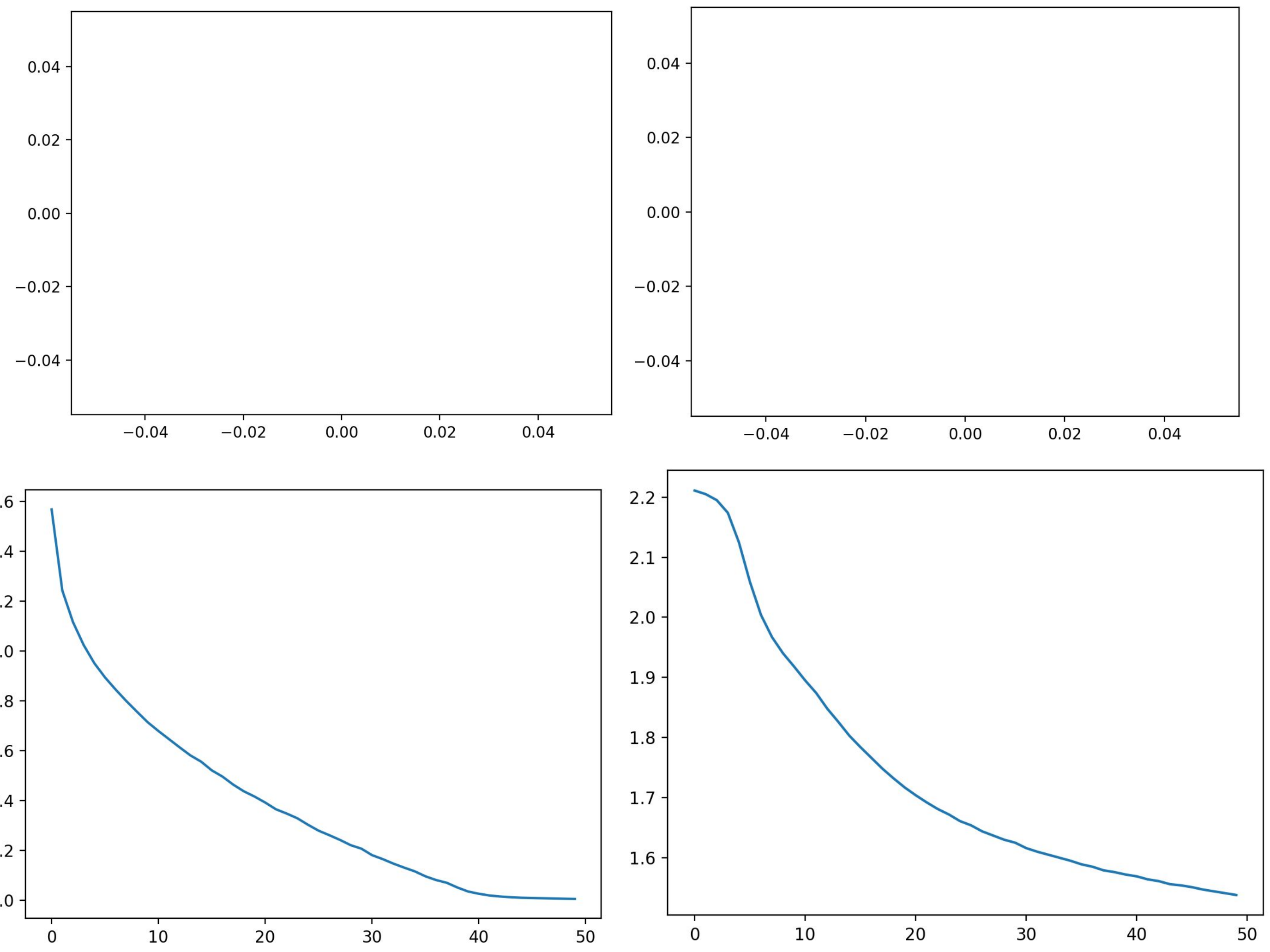
3 How does learning rate work?

For both v7 and v10, learning rates of 10 and .1 were too high, so I got bad accuracies and nan loss values. Here is where I thought I fucked up by picking a super overfitted model, so redid everything for v7, too RIPperoni x 1000. 0.01 worked best, .0001 had a learning rate a little too small to get to a good accuracy within 50 epochs. I chose .01 as the best learning rate because it gave the best accuracies within 50 epochs. So, even though a learning rate of .0001 resulted in less overfitting, it just learns too goddamn slow for good accuracies.

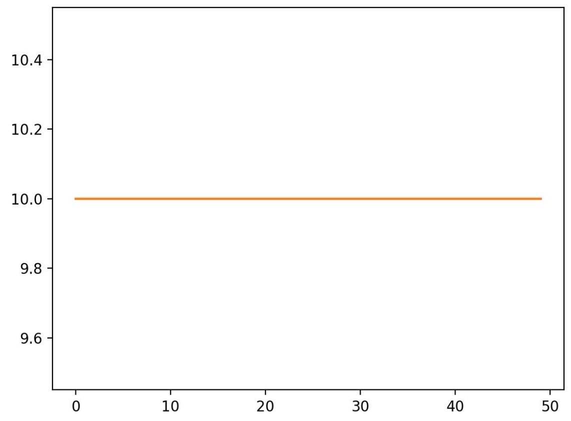
Accuracy for CoolNet v10 with a learning rate of 10, 0.1, 0.01, 0.0001:

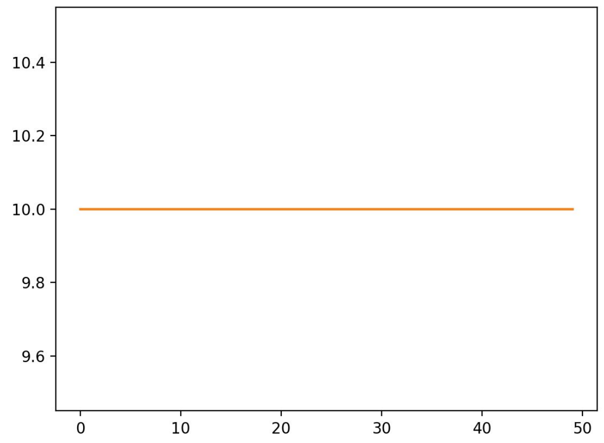





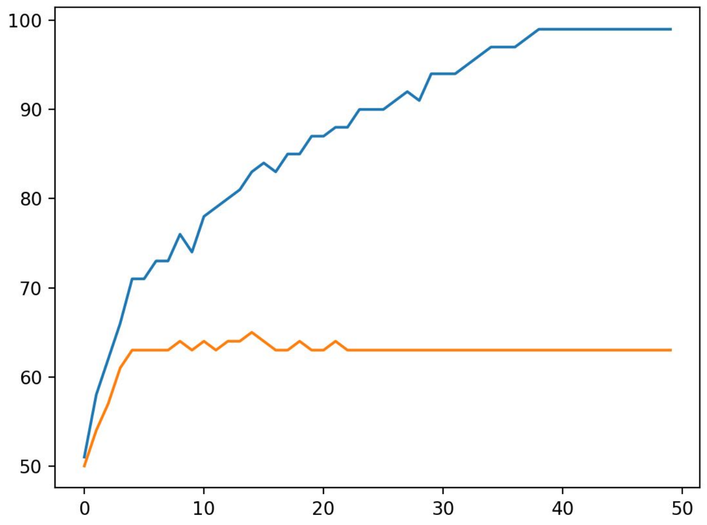
Loss for CoolNet v10 with a learning rate of 10, 0.1, 0.01, 0.0001:

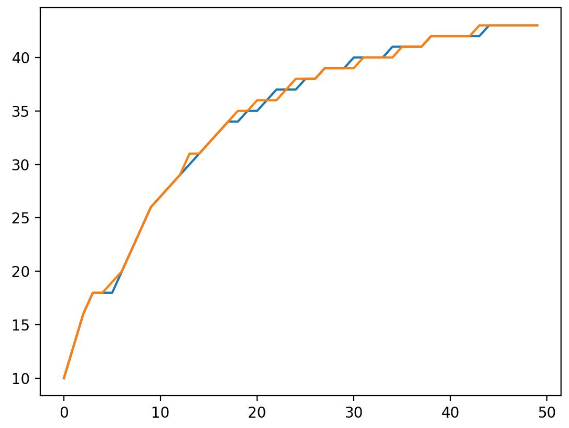


Accuracy for CoolNet v7 with a learning rate of 10, 0.1, 0.01, 0.0001:

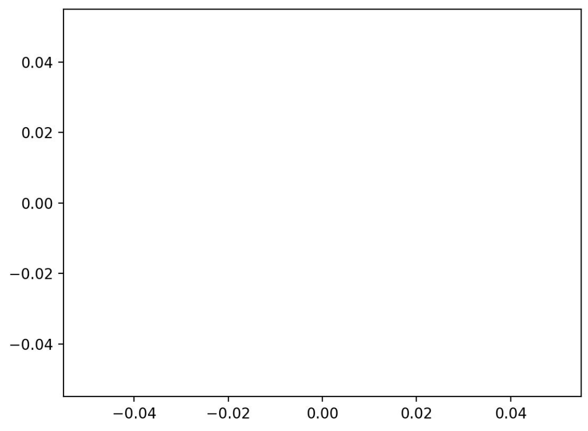


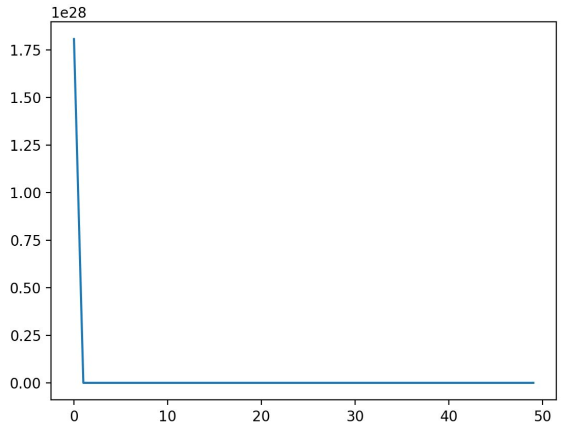


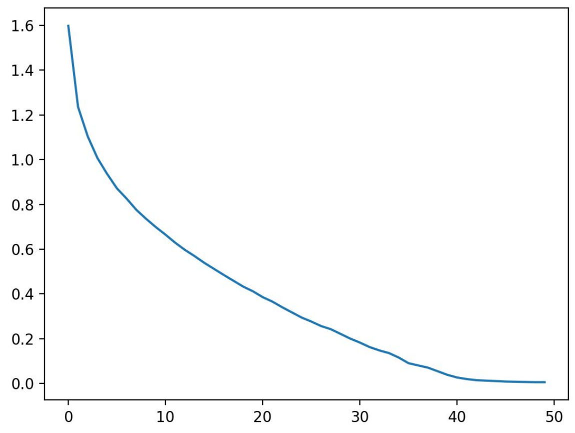


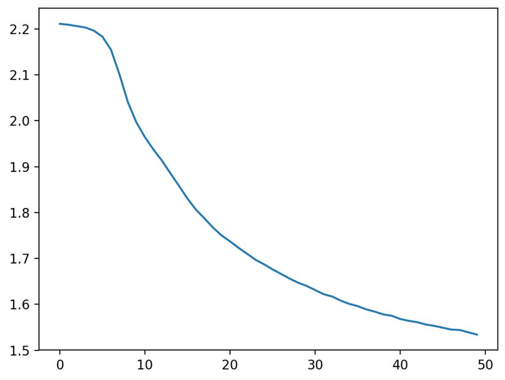


Loss for CoolNet v7 with a learning rate of 10, 0.1, 0.01, 0.0001:





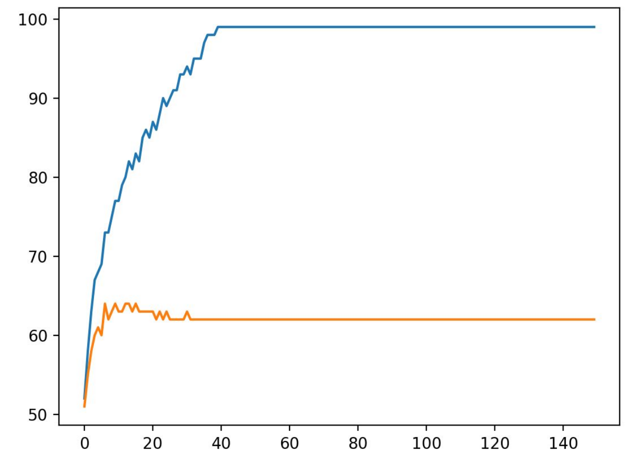


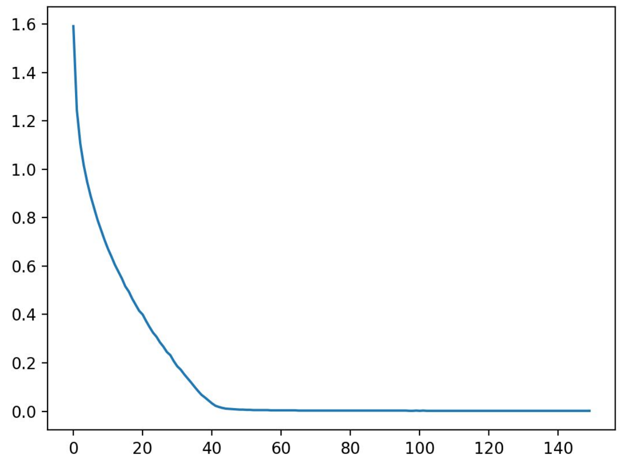


At .0001, the model initially struggles to reduce loss.

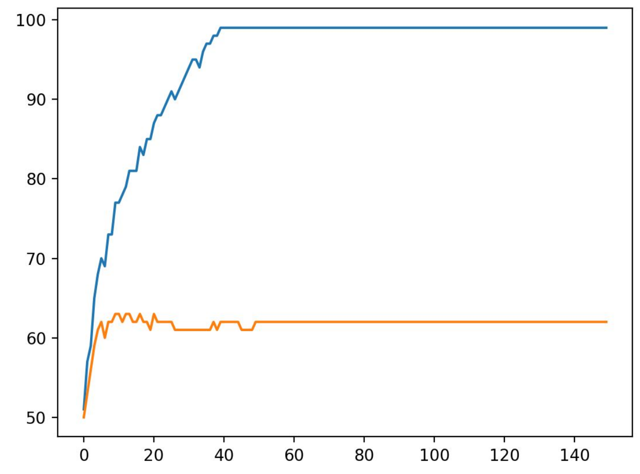
A higher learning rate means it converges faster but converges at a worse value and a lower learning rate converges slower but at a better value. It’s a good idea to reduce learning rate as the training progresses because a higher learning rate at first helps our model narrow towards the right direction faster and us not to oscillate past it back and forth. You can see this in our accuracy plot for testing, previously it would dip down towards later epochs, but here it converges.

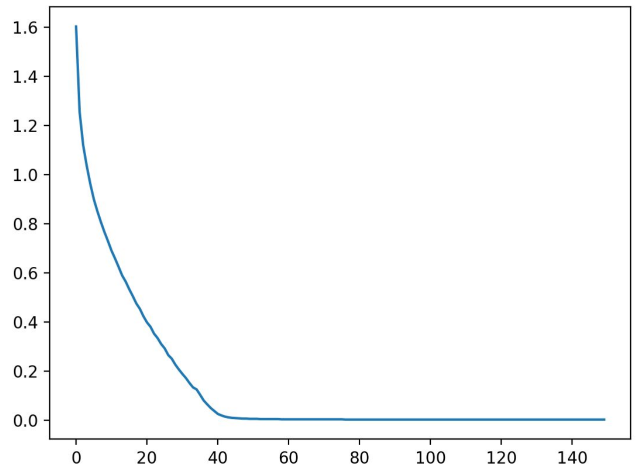
Accuracy and Loss for CoolNet v10 with a decreasing learning rate:





Accuracy and Loss for CoolNet v7 with a decreasing learning rate:

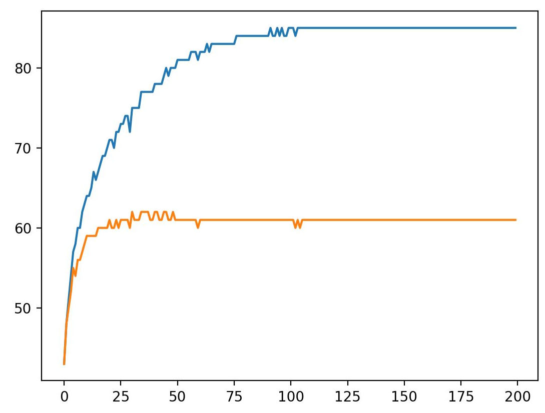
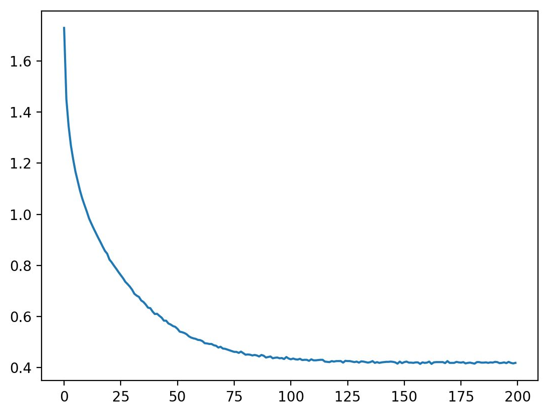




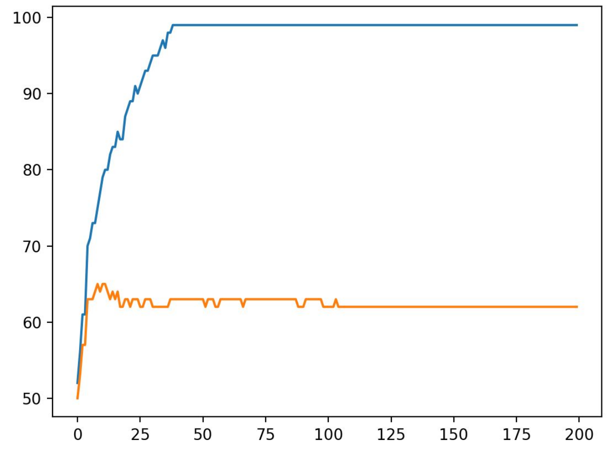
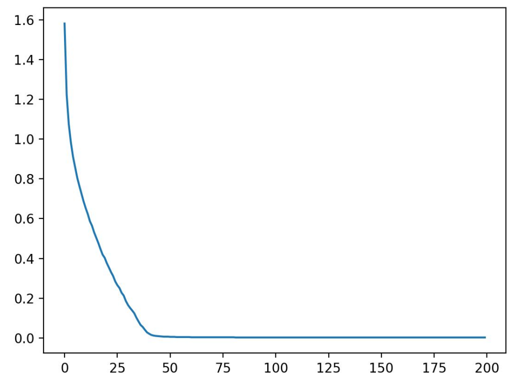
4 Data Augmentation

Training accuracies were lowered and testing accuracies remained relatively the same. Loss graph looks way more closer to an exponential function.

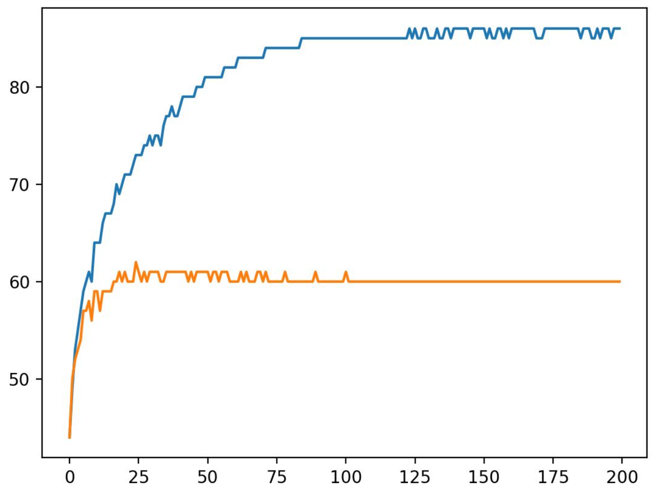
Accuracy and Loss graphs for CoolNet v10 with data augmentation:

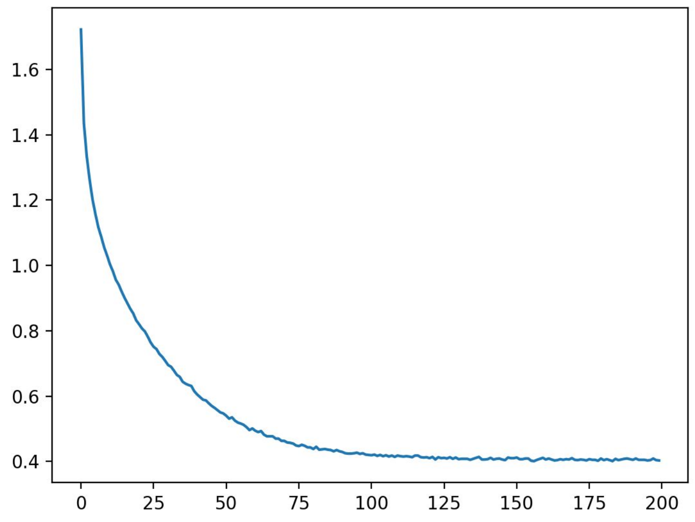



Accuracy and Loss for CoolNet v10 without data augmentation and with:

Loss for CoolNet v7 without data augmentation and with:



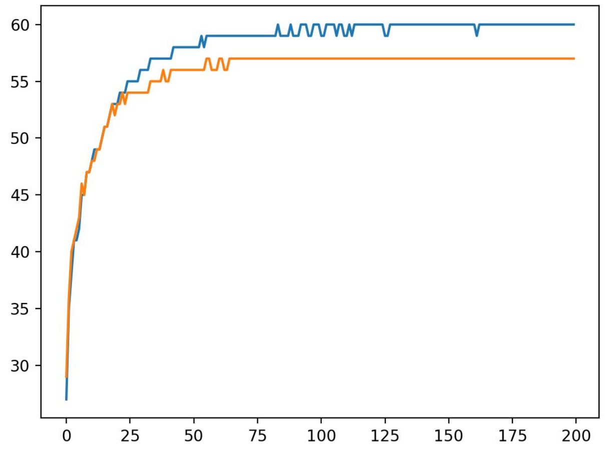


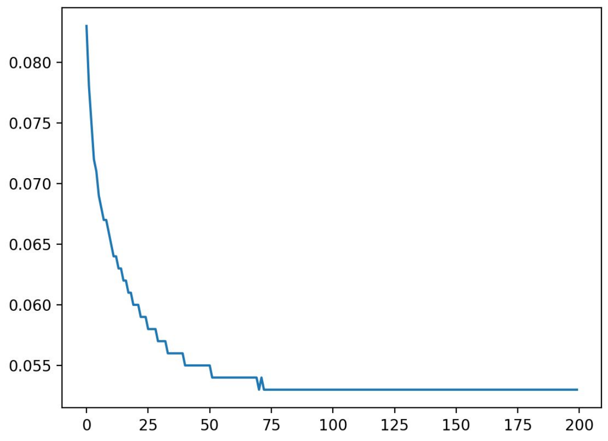
I forgot that I was running without data augmentation so I stopped a run at like 120 epochs because I thought I was supposed to be doing the MSE run for it so oh well.

Testing accuracies remained about the same, but training accuracies was lowered and also more bumpy, I did quite a few data augmentations that had probabilities for actually augmenting, so this makes sense.

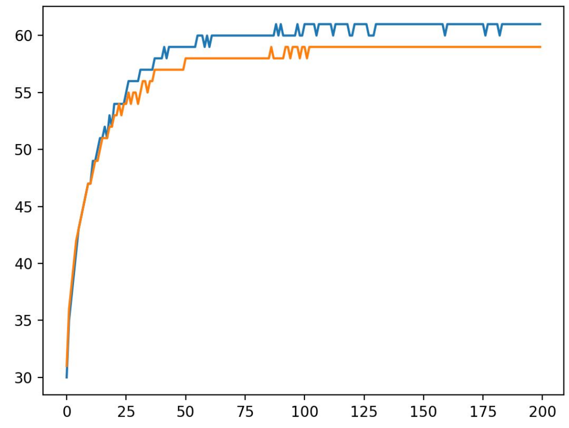
5 Mean Squared Error loss instead of cross entropy
I couldn’t run this test for both CoolNet v10 and CoolNet v7 because one run with epochs = 200 and data augmentation takes 5 hours, so 10 hours for both v10 and v7, can you tell that I hate myself?

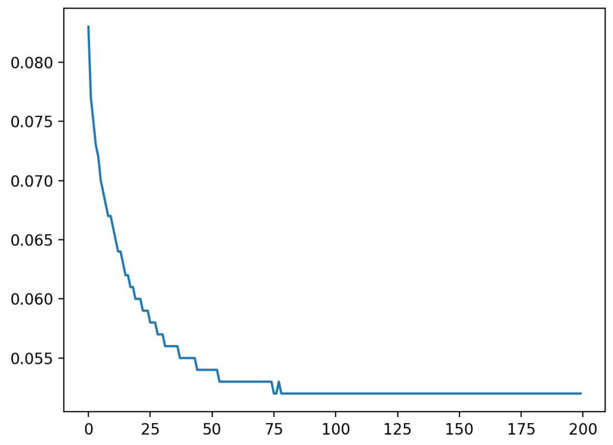
Accuracy and Loss graphs for CoolNet v10 with data augmentation, 0.01 learning rate, 200 epochs, and MSE loss:





Accuracy and Loss graphs for CoolNet v10 with data augmentation, 0.01 learning rate, 200 epochs, and MSE loss:





The loss is less smooth, but the main difference is there is much less overfitting. I think this is because MSE makes it so there is less reduction in loss as it runs and it somehow affects how the model learns.

Conclusion

Don’t run neural nets on a CPU. Also, for future classes, you guys can set up a separate attu dedicated for the class, this way it won’t be so brutal (I literally couldn’t use my laptop or else pytorch would crash) and give instructions on how to set up a virtual python environment so we can run it on attu (it’s hard to just do on attu because permissions). Also, I know you warned that it would take a long time to run but 5 hours for one run is ridiculous and also wished someone told me to just go ahead with a random model instead of trying out so many and getting scared that I picked a bad one (I know I should have gone to office hours and posted online but I’ve made a fool of myself on the board before so idk).