**Lab 9: Hyperparameter Grid Search**

Stuart Harley

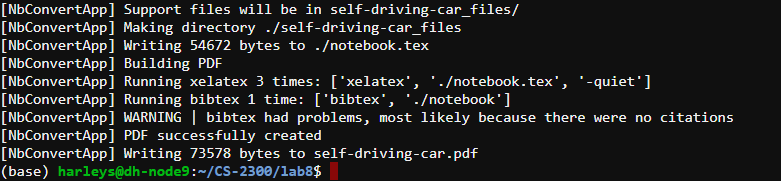
**Introduction:** In this lab I will be using the command line to execute my model training on the self-driving car model. First, I will use the anaconda environment, and then I will get experience running my training within a singularity container. I will use this opportunity to be able to easily run simultaneous experiments with different models and hyperparameters to find an “optimal” hyperparameter configuration and model structure.

**Learning Outcomes:**

* Running a model from the command line
* Running a model within a container
* Writing scripts to launch multiple containers and training instances
* Creation and execution of a grid search of models and hyperparameters

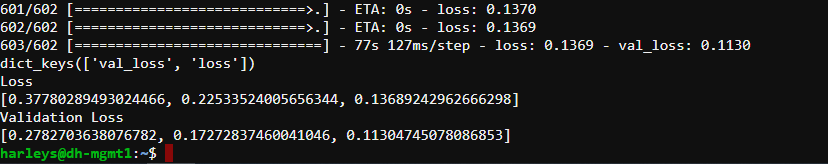
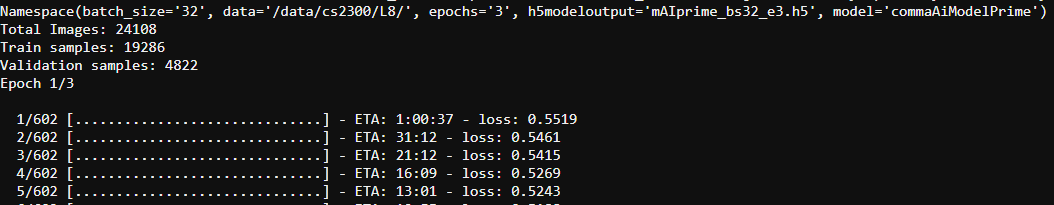
**Part 1:**

Screenshot of command line running the notebook from lab 8 and creating a pdf. (Ran the provided dataset for 5 epochs).

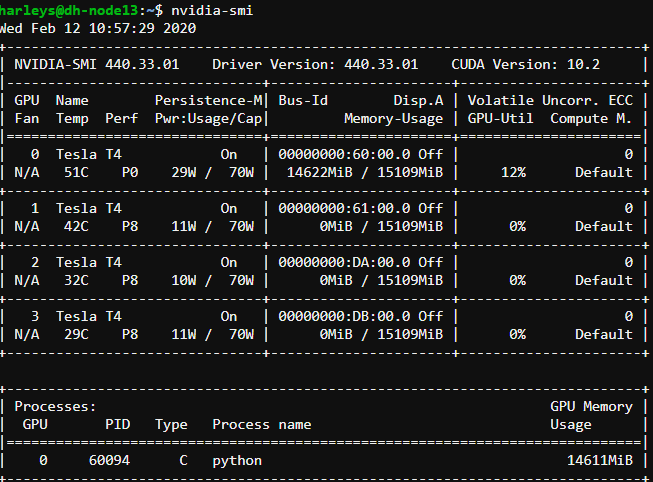


**Part 2:**

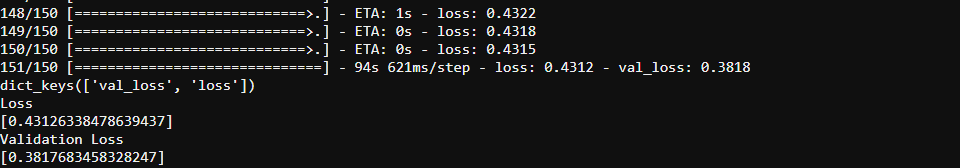
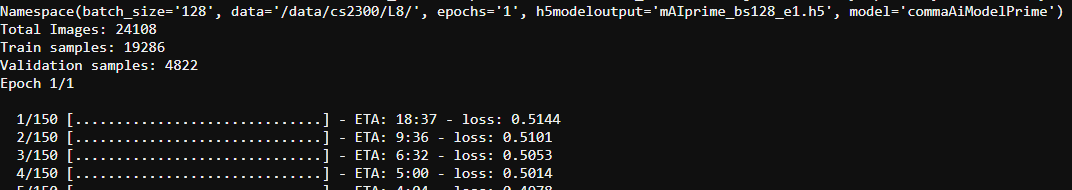
Screenshot of command line starting and ending running model.py (run 1: 32bs 3e)



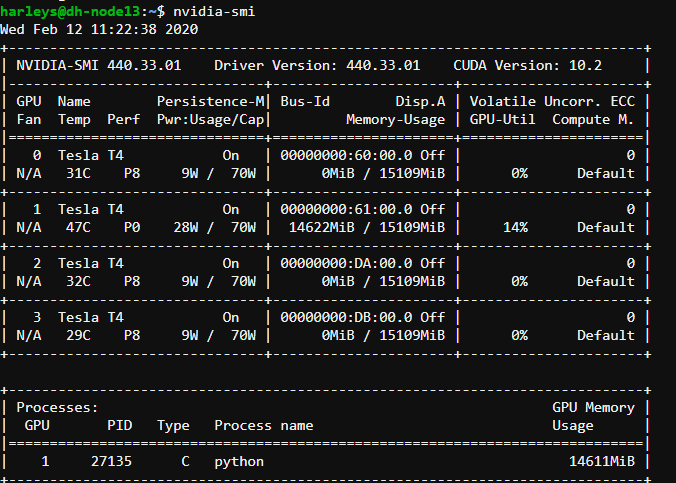
Screenshot of GPU utilization view from the command line (run 1: 32bs, 3e)



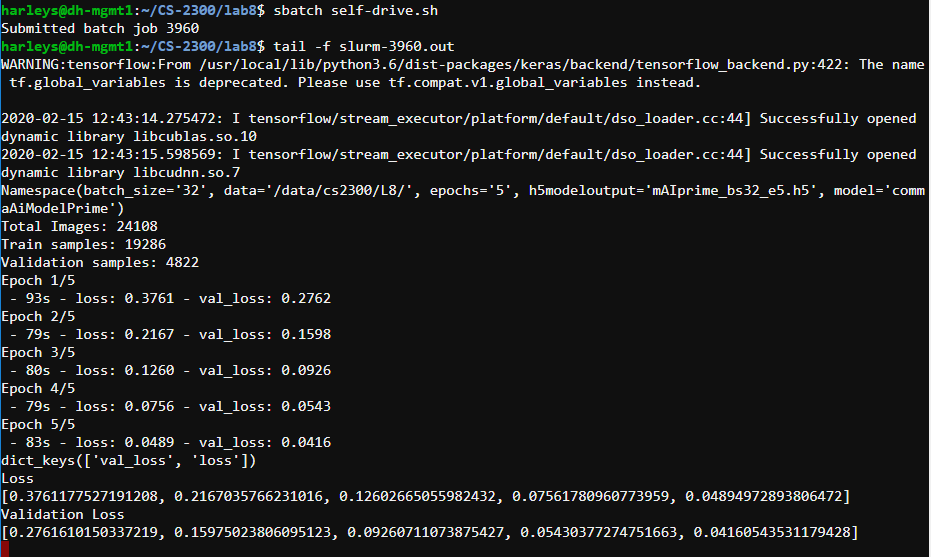
Screenshot of command line starting and ending running model.py (run 2: 128bs, 1e)

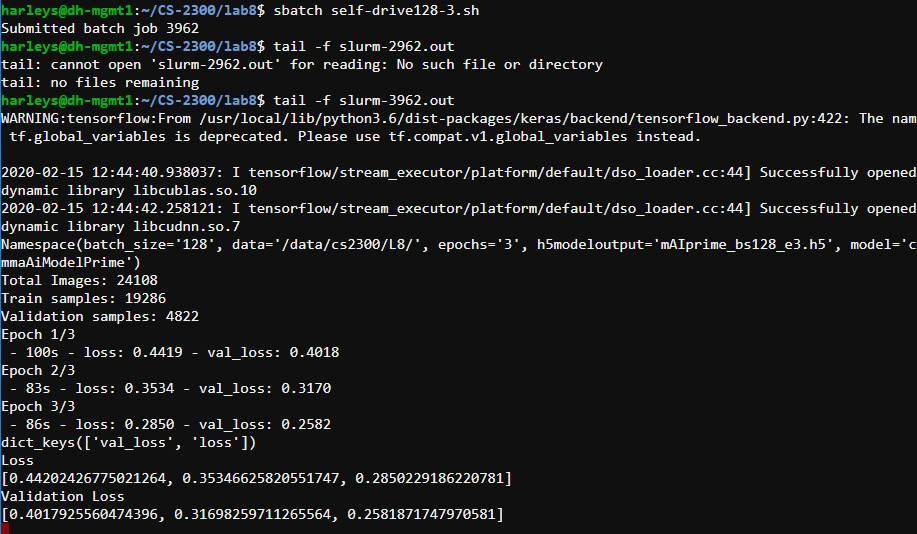


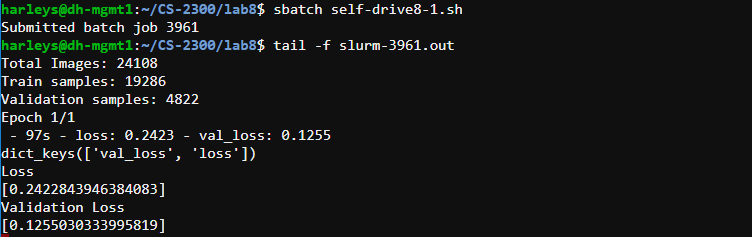
Screenshot of GPU utilization view from the command line (run 2: 128 bs, 1e)

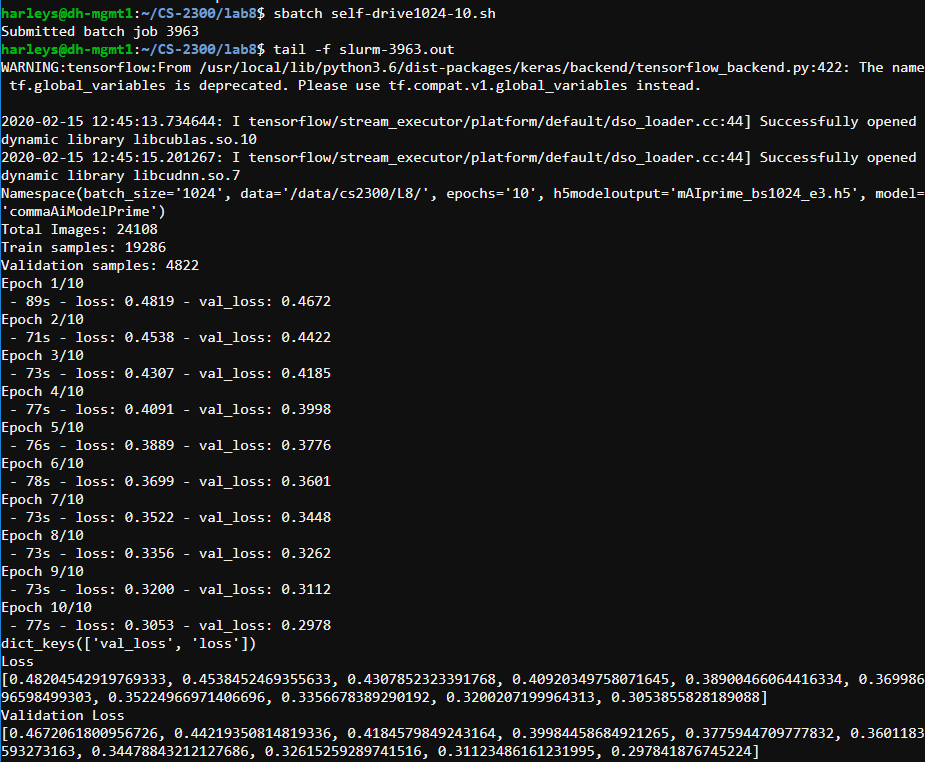


**Part 3:** Shown: Command line running of 4 jobs running simultaneously. (running in 4 different shells). The jobs are creating models of varying epochs and batch sizes.

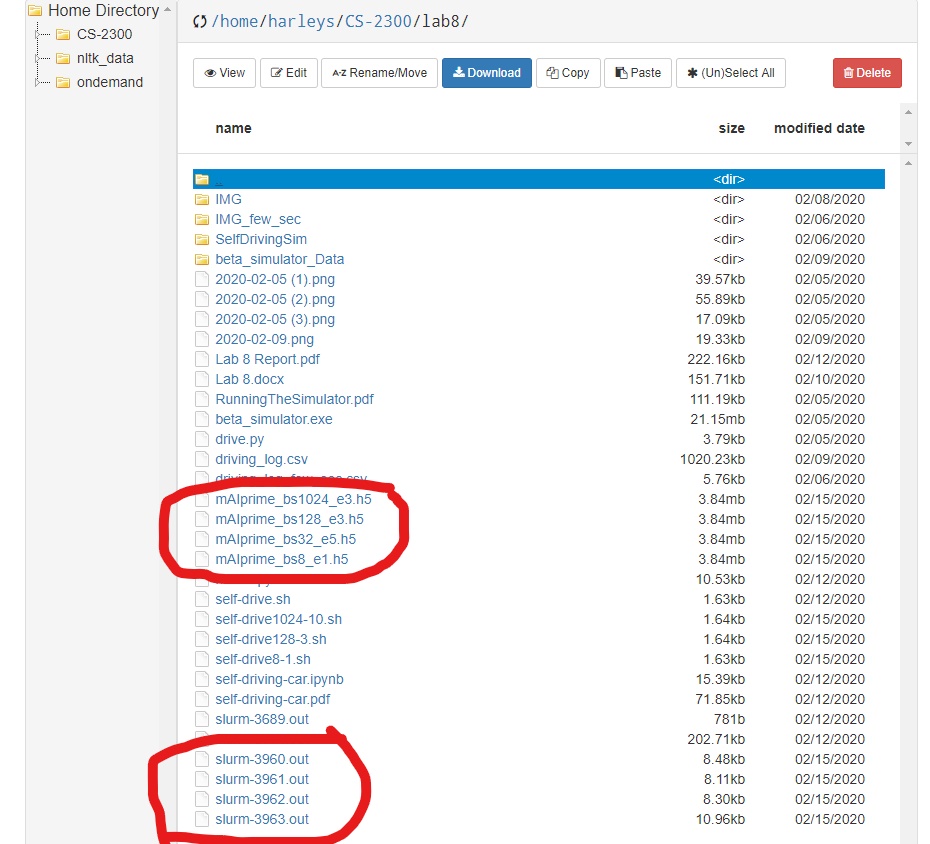








Shown: Output .h5 files and slurm-<jobid>.out files created in file explorer



**Part 4:** For this part I am comparing 3 different types of models in regard to their performance in the driving simulator. The three models are the CommaAIModelPrime model, the basic model, and the nVidia model. In order to standardize the comparison of the performances of these different models, I used the same hyperparameters to train each one. The hyperparameters I used were a batch size of 32 running for 5 epochs. The loss of the CommaAIModelPrime model was about .045. The loss of the basic model was about 3. The loss of the nVidia model was about .01.

CommaAIModelPrime: Lake Track – The car drove around most of the track mostly in the center of the track. However, on the right turn of the track, the car went off the road and into the lake.

Jungle Track – The car immediately veered left and ran into one of the separators between the two roads.

Basic Model: Lake Track – The car immediately turned to the left and did a full circle before getting stuck on the outside of the track trying to drive back on the track. It is trying to drive in a counterclockwise circle.

Jungle Track – The car immediately turned to the right and got stuck on the mountain to the right of the start of the track. It is trying to drive in a clockwise circle.

nVidia Model: Lake Track – The car drove around the entire track without any issues. For a split second it looked like it might go off onto the dirt part of the track, but then it turned back towards the main track.

Jungle Track – The car veered off to the right and went off the road and got stuck,

Conclusion: It was very obvious that the basic model (that had the worst loss) did the worst as all it tried to do were circles. The next best was the CommaAIModelPrime that made it most of the way around the lake track. And the best model was the nVidia (which had the best loss) which was able to navigate the entire lake track. None of the models were able to navigate the jungle track but with more epochs it would probably be possible, at least for the CommonAI and nVidia models. The basic model may not be a large enough network to be able to perform this task. However, in summary, the higher the loss of the model, the worse the driving is.