Lab 8: Self-Driving Car

Stuart Harley

Introduction: In this lab I will be training a Deep Neural Network (DNN) using Keras to enable a car to drive autonomously. Additionally, I will be generating my own training data to see the effect of using different data on the resulting model quality. I will also be playing with some hyperparameters to get a sense of how they affect the overall model quality.

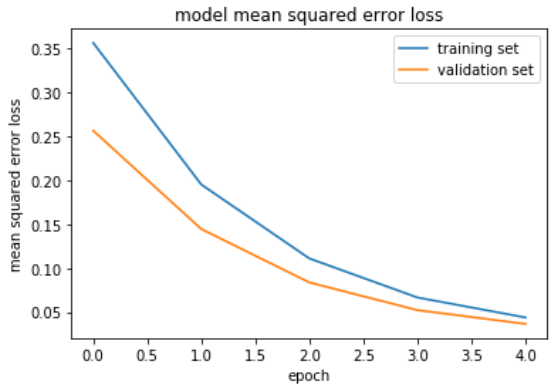
Learning Outcomes:

- Use the simulator to collect data of driving behavior
- Use a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around the track without leaving the road
- Summarize the results with a written report

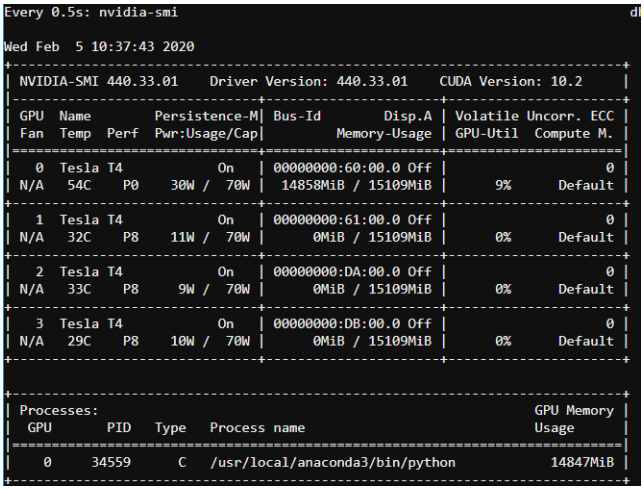
Training model with a batch size of 32 and 5 epochs:

Graph of Mean Squared Error Loss: As shown, as the model progressed, the loss and validation loss both decreased, as expected since the model is improving.

```
['/data/cs2300/L8']
Total Images: 24108
Train samples: 19286
Validation samples: 4822
Epoch 1/5
603/602 [==============================] - 94s 157ms/step - loss: 0.3566 - val_loss: 0.2567
Epoch 2/5
603/602 [==============================] - 92s 153ms/step - loss: 0.1953 - val_loss: 0.1446
Epoch 3/5
603/602 [==============================] - 91s 151ms/step - loss: 0.1110 - val_loss: 0.0837
Epoch 4/5
603/602 [==============================] - 84s 139ms/step - loss: 0.0665 - val_loss: 0.0522
Epoch 5/5
603/602 [==============================] - 89s 147ms/step - loss: 0.0438 - val_loss: 0.0365
dict_keys(['val_loss', 'loss'])
Loss
[0.35667272448156506, 0.19531690385793363, 0.1110519664965699, 0.06655794382018072, 0.043807407253685794]
Validation Loss
[0.2566589530154117, 0.14458353627721882, 0.08371403369602945, 0.05215281080755887, 0.03649572368320317]
```



GPU utilization: As shown, the GPU utilization runs at about 9% on one GPU.



Training models with various parameters to see the effect on loss and GPU utilization:

32 batch size, 1 epoch: GPU utilization did not change, loss was comparable to the loss after the first epoch of the original model.

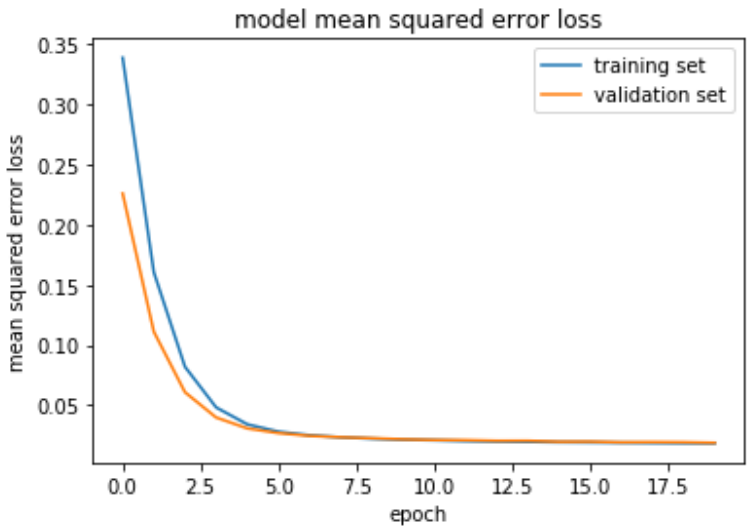
32 batch size, 10 epochs: GPU utilization did not change, loss improved to only about .02 in the final epoch with no evidence of overfitting.

32 batch size, 20 epochs: GPU utilization did not change, loss improved to about .0175 with no evidence of overfitting.

2048 batch size, 5 epochs: GPU utilization increased to about 60% on the second GPU. Loss was in the .4's after each epoch, decreasing slightly in each. The is because the batch size is significantly higher, so there is more averaging, so there has to be more epochs to achieve the same loss.

Training a model with 32 batch size, 5 epochs, and flipped images: The loss ended up being slightly less than the original model so flipping the data reduces loss. This is because flipping the data augments the dataset so it has more data to train from, so it can train better.

Training a model with 32 batch size, 20 epochs, and flipped images: These are the parameters I used for my final model (for this part). The loss ended up being about a .018. The mean squared error loss graph for this model is shown below.



Testing the previous model in the autonomous mode of the driving simulator: The car successfully drove around the entire lake track without issues. It never left the track. The car stayed at about 15-16 mph the entire time.

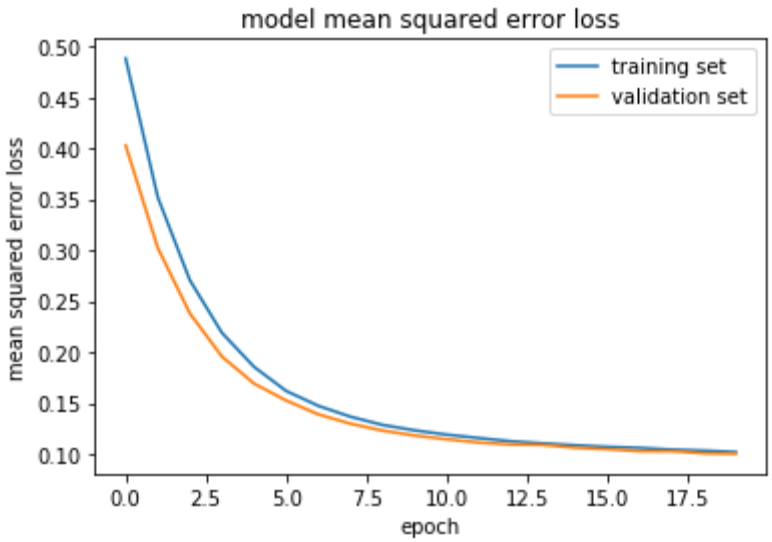
In the jungle map, the car immediately crashes into a divider and gets stuck. This makes sense since the roads look differently in the jungle map and the car was not trained on images from this map.

Generating a few seconds of my own data: I drove the car manually for just a few seconds on the lake track and recorded it. I used the data generated from this to train a new model. This model's dataset is significantly smaller than the one I previously used, so it took only a couple of seconds to train. Because the dataset is smaller, the loss is also worse, at about .43.

When I ran this model on the lake track, the car actually did better than I expected. It stayed on the right edge of the track and made it most of the way around the track before eventually going off-road and getting stuck. I was mostly surprised that the car made it across the bridge without any problems because on the few seconds of data I created, I was not on the bridge.

When I ran this model on the jungle track, the car immediately veered off the road to the right and got stuck. This is not surprising because the dataset had no data from the jungle track.

Generating Final Data: For my final data I manually drove the car around each track three times in a row, being careful not to leave the road. This ended up creating a total of 23721 images in my dataset which I trained my model with. I trained the model with 20 epochs, a batch size of 32, and I flipped the data. When I created the model without flipping it some overfitting occurred. Flipping the images got rid of the overfitting. The model took about 30 minutes to train. Below is the mean squared error loss graph. The loss ended up being about .10.



When testing this model in the driving simulator, the car was able to drive around the lake track without any issues. The car is also able to drive around the jungle track without any issues.

Conclusion:

1) The provided data was "good" for the lake track, as it was able to drive around the lake track without any issues. However, this same data was bad for the jungle track, as it would crash and get stuck immediately. Therefore, I would hypothesize that the provided data was generated on the lake track but not the jungle track. Therefore, the car would know how to act on the features of the lake track, but not on the jungle track whose features are different.
2) I ran a line profiler on the cell that trains the model. 99.9% of the time that is taken in training the model is from the following line:
   history_object = model.fit_generator(train_generator, steps_per_epoch= \
   len(train_samples)/my_batch_size, validation_data=validation_generator, \
   validation_steps=len(validation_samples)/my_batch_size, epochs=my_epochs, verbose=1)
   This is the fit method. This is the method that actually trains the model. It is in this line where the model weights are adjusted according to the data values so that better accuracy can be achieved.
3) The model training time appears to be mostly proportional to the amount of data collected as shown in the table. Obviously, it also depends on using the same hyperparameters in training. In this table I also included my first attempt at training the car where the car was not able to successfully drive either track as evidence.

Number of Images in Dataset	Time to Train Model
46	~ 5 sec
6048	~ 9 min
23721	~ 30 min

4) My first attempt at making a model to navigate both tracks was unsuccessful. I ran a lap of both tracks once but the model trained couldn't drive around either track successfully. The car almost made it around the lake track but really did poorly on the jungle track. I believe this is because I was going full speed on the jungle track and with all the turns and inclines, the car tried to imitate what I did but couldn't. Aka, this dataset had flaws, so the model was also flawed.
   In order to make the final model that successfully went around both tracks I made my dataset larger by driving around both tracks 3 times. I also went much slower on the jungle track even using the brake in order to keep the car under greater control. Because the dataset was larger, the model was able to learn more and become good enough to drive the car around both tracks. The dataset was also better because I drove the car better on the jungle track.