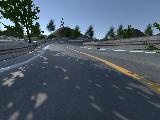
**Winter Break**

I started training the auto-encoder in simulation. The code can be found here: <https://github.com/mgagvani/aae-train-donkeycar>

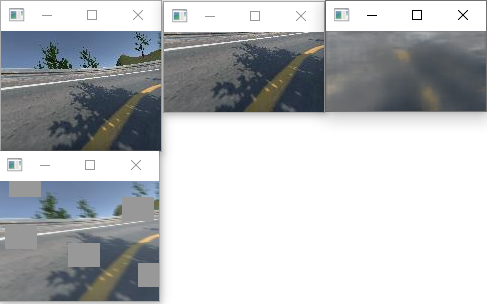
The idea of the autoencoder is that it compresses an image of the road into a small latent vector which is then fed into the reinforcement learning algorithm through a wrapper class. However, before it can be used in RL it needs to be trained with a corresponding decoder. I started by collecting data manually by controlling the car with a joystick. The original implementation of the data recording used the arrow keys, but I wanted more control over my dataset, so I added the ability to control the car with a joystick while recording data. The resulting dataset consists of 976 pictures, but not the corresponding throttle/steering values.

Here are the relevant details of the autoencoder: The encoder and decoder consist of 4 convolutional/transpose convolution layers with ReLU activations, except for the last layer of the decoder, which uses sigmoid. The input images are of size (160, 120, 3) which are cropped to (160, 80, 3) because the top 1/3 of any given picture doesn’t contain anything very useful:



In order to improve the quality of the reconstruction and create a more robust autoencoder, the data is augmented through random left-right flipping, random shadows, Gaussian blurring, motion blurring, random pixel jittering, and cutouts (small rectangular portions of the image are replaced by the average of those pixels). Note how different the shadows are in each image. We want the autoencoder to produce a latent representation of the road invariant to shadows or lighting, hence the transformations.

Here are a few examples of the autoencoder being used to reconstruct the original source image. This functionality won’t be needed for reinforcement learning (only the dimensionality reduction the encoder will be) but it is interesting to see what the autoencoder does:

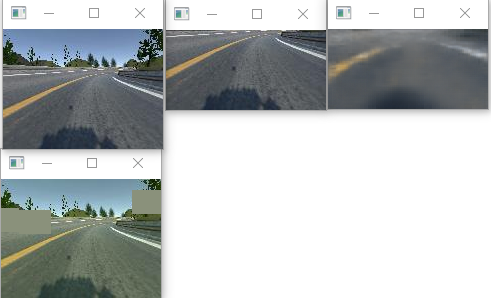


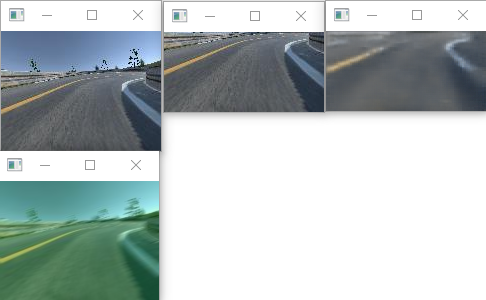
*Top left: Input image*

*Bottom left: Augmented input image*

*Middle: Cropped input image*

*Right: Reconstructed input image*



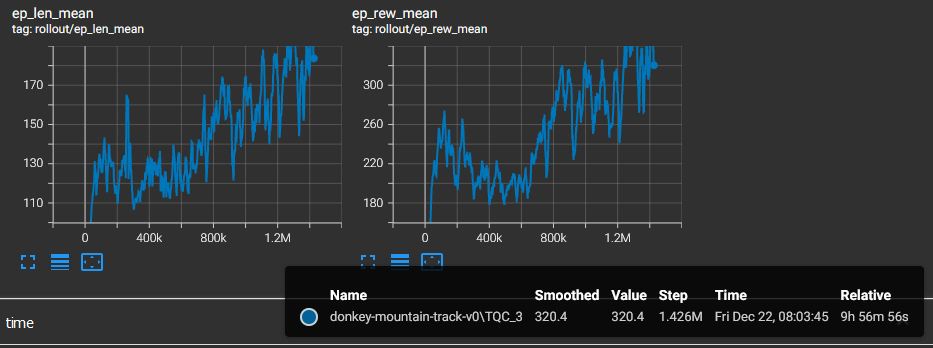


Note how the reconstructed images contain the yellow and white lines, but they don’t generally have shadows or texture on the road due to the augmentations.

Running on n=500 samples, the autoencoder runs at 140 Hz on my Nvidia GPU. Hopefully, I will be able to run it at a comparable speed on the Nvidia Jetson board since both use CUDA.

The next step was to create a wrapper around the environment which alters the observation space from an image to the latent vector generated by the encoder. The speed of the vehicle is also concatenated to the end of the latent vector, creating an observation space of shape (33,) in the wrapper. The policy was changed from a CNN to a simple fully-connected network with two layers of 256 neurons each – a significant advantage of preprocessing the images with the autoencoder.

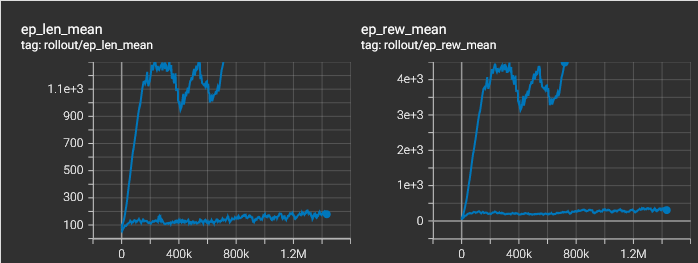
I was then able to run the reinforcement learning training using the autoencoder to pre-process the images taken by the car. Below are the logs of the car’s training over time:



The first graph shows episode length over time while the second graph shows reward over time. The black info box on the bottom right shows the result of the most recent episode. Although these results look promising, in reality, the car was struggling to make it around the track consistently even after training overnight.

**Wednesday, January 3rd, 2024**

To fix the issues I was having over winter break, I recorded more data for the auto-encoder and re-trained a newer one. I was able to decrease the reconstruction loss from 0.66 to 0.24. Using the autoencoder trained on more data allowed me to successfully train a reinforcement learning agent in the simulator.



The bottom line is the original graph from above, while the top line is from the new training session. The car was successfully able to navigate the track after about an hour of training, and the best laptime it achieved was 15.5 seconds, an improvement over the 18-20 seconds I was getting when the policy learned directly from RGB images.

I also talked with Dr. Gabor about the AprilTag which I had mentioned in a previous journal report. AprilTags are visual markers with standardized sizes and patterns. I have printed one out and stuck it on the cart next to the desk in room 200C.

OpenCV has functionality to obtain the relative pose of a calibrated camera relative to an AprilTag. I will need some sort of way to reset the car’s position during a reinforcement learning session each time it crashes. I could try to “play backwards” previous steering commands, but this is prone to drift over time from tire slippage and inaccuracies in the steering servo motor. Thus, it may be necessary to strategically position AprilTags along the track to give the car an accurate position measurement as it drives.