GTA V SELF DRIVING CAR WITH MOTION

Evan Miller
Student | Redei Labs | Central Michigan University

Introduction

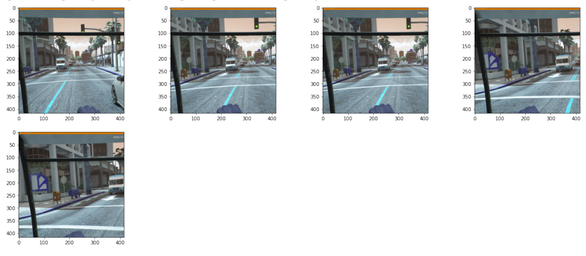
In this project, we set out to create an autonomous driver for the game Grand Theft Auto V. This will be accomplished by training a Neural Network on a set of images taken from actual players driving in the game. As a secondary goal, we aim to integrate the game's telemetry data tracking into a program that controls a motion simulator in our lab.

This project is an attempt to improve upon the methods used in Aiden Yerga Gutierrez and Iker Garcia's project *GTAV-Self-driving-car*. In their documentation, they noticed that their network was driving by focusing on the path highlighted on the GPS while ignoring the actual environment. With this in mind, we have 3 goals for the AI portion of this project

1. Remove UI components that allow the model to "cheat" as it learns
2. Encourage the model to react to objects like cars and pedestrians by highlighting them in the dataset
3. Rewrite training functionality using Generators to allow for easier modification and ease of updates to modern versions of Tensorflow

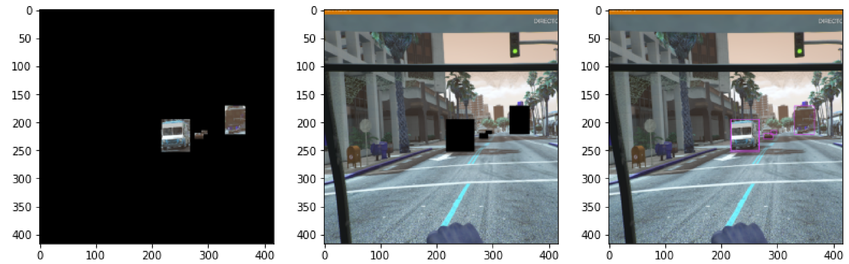
Dataset

Our datset is comprised of screenshots taken while actual players drove around our training route. These screenshots are grouped into sequences of 5 and saved as 3 channel Numpy arrays, which are then paired with the combination of buttons pressed on the last image. We choose to analyse sequences of images because they encode information about momentum and direction of movement for our model.



Object Detection

We utilize two techniques to highlight the objects in our images. Both of these methods are based on utilizing the YOLO object detection algorithm pretrained to recognize objects that are commonly found on the road such as cars, bicycles, and people. This network was trained by Lavanya Shukla as part of the Lyft 3D Object Detection for Autonomous Driving challenge on Kaggle.

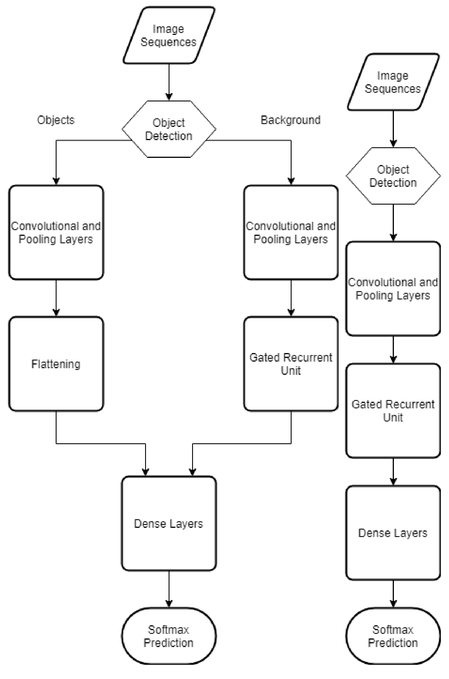


Our first method uses our object detection model to create bounding boxes for any objects found in our image. The coordinates for these boxes are then used to create a mask that we apply to our images to separate them into two: one that contains objects and one that contains the background. Each image will be sent as an input to separate Neural Networks, which are later concatenated, so as to allow the model to react to the environments and objects differently.

For our second method, we will draw our bounding boxes directly onto our images. While this requires that both the objects and backgrounds are processed by the same network, it greatly reduces the complexity of the inputs to our model, which theoretically makes it easier for our network to learn patterns.

Neural Networks

The hyperparameters for both networks are adapted from Aiden's project, since we had little time to tune our model ourselves. The training process was completed using a custom Tensorflow data generator that reads data from our saved arrays and preprocesses them by performing object detection before grouping them into arrays. This was done as the entire dataset of images is too large to fit into working memory in its entirety. We also chose to calculate training weights for our classes using the formula $\frac{num\ largest\ class}{num\ class}$ where $num\ class$ is a count of all instances of a specific key combination in our dataset and $num\ largest\ class$ is the maximum of $num\ class$ over each class. We used this method instead of balancing our dataset because it is important that the long straightaways are all recognized, and balancing the dataset significantly reduces the model's exposure to these areas.



Motion Simulation

We used our labs custom Motion Simulator, which has been in development since 2017, for this project. The cockpit came from Tennessee, the motors came from Florida, and the inverters came from Utah. We have two primary electric motors which are geared to deliver approximately 2000 ft-lbs of torque in an instant, which is roughly the equivalent of two semi trucks chained together. The simulator can be seen in the video in the Results section.

We wrote a custom plugin using OpenIV to connect our simulator to GTA V. Fortunately, the game keeps track of the telemetry data for a vehicle the player is in, so these values can be monitored and passes to an interpreter for the motions of our machine.

Results

Both the separate image and image highlighting methods achieved reasonable accuracies on our dataset, and performed similarly when tested in the game. The motion simulation plugin works as intended, the training process has been simplified using a generator, and the AI demonstrated behavior that showed it was recognizing both background environments and vehicles. However, it had many shortcomings.

Our model learns entirely based on imitation, so it oversimplifies certain necessary tasks. One example of this was observed when the AI approached a turn during training. It began to slow down to take the turn, but it was slowing down too quickly. Since it was still recognizing an environment where it should be holding the S key, it slowed all the way to a stop and began to go backwards. After backing up for a few seconds, the model again recognized an area where it should go forwards and began to repeat this cycle.

To address the issue with recognizing distance in our images, one could use a heatmap corresponding to the distance to objects in each image. Many algorithms to do this exist, and they could easily be implemented during the preprocessing phase of our training generator.