Final Project Writeup

Team 13: Payton Suiter, Lewai Attia, Jake Grogan

Introduction

For this project, we trained an ANN to play puck soccer in the SuperTuxKart environment. This write up will discuss our model, how we trained it, how we collected and cleaned the data, how we created our controller, future development, and lessons learned.

Model

Our model (PuckDetector()) utilized a fully Convolutional Neural Network with a Convolutional Block and an Up-Convolutional Block along with skip connections. We found that layers of 16, 32, 64, 128 delivered the best results. Along with the CNN, we also utilized a Spatial Argmax function to determine the hotspot (puck) in the image.

Training + Data Cleaning

As for our loss, we tried out both MSE loss and L1 loss. Our train file consisted of the basic training epoch loop using image transformations. To generate our data, we took the original tournament code we were provided and made some alterations using the same (-1,-1) to (1,1) grid as in homework 5 which allowed us to generate our training data and labels as follows:

1) IMAGE (total ~20k images)
    - Every frame of each player's view in a 2 vs 2 game
2) LABEL
    - If the puck was in view - (x,y) coordinates of the puck's location
    - If the puck wasn't in view - (-1,-1) (or the top-left corner of the players screen).

It took a number of iterations for us to land to this schema, especially when it came to deciding what our labels would consist of. Initially, we did not do any "filtering" of the labels in any way, so images that did not have the puck were not distinguished from those with the puck, producing inconsistent labels. When we first started to train using this data, we didn't really see much "learning" going on, our losses generally remained constant or high even after altering the various hyperparameters to our model.

Eventually we realized our error and were able to figured out how to recognize images without the puck at data generation. Our first attempt from that point was to simply leave out all images where the puck did not exist (image set of ~6k, let's call this "TSET A"). We finally started to see improvements in learning and we actually saw great success in our model locating the puck if it was on the screen (let's call this model, "Model A").

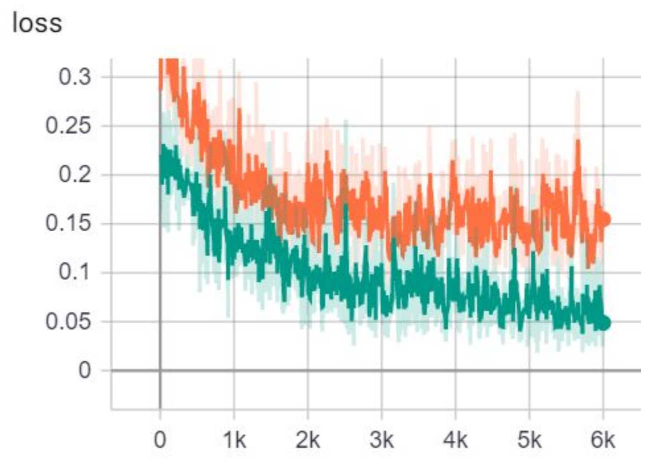

Figure A:
Loss of L1 (upper, orange) vs MSE (lower, green).

L1 loss stays pretty much stagnant 2k global-steps, whereas our MSE continues to trend downward over time. This is just one example of many, as we pretty much saw this difference consistently throughout different training hyperparameters and datasets.

However, when the puck was not on the screen we noticed erratic behavior from the model. Namely, the predicted puck location jumping around the screen. This made writing a controller really difficult because we had no way to determine when the kart was tracking/hitting the puck vs. searching the map. To address this, we generated a new training set (image set size: ~14k) of images both with and without the puck and set the label to (-1,-1) if the puck was not in the image. We chose this area of the screen because the puck would never be found in that location, which provided us with an excellent indicator for our controller of when the puck was not on screen. We achieved immediate success with this strategy and were able to effectively program our controller to identify the correct state.

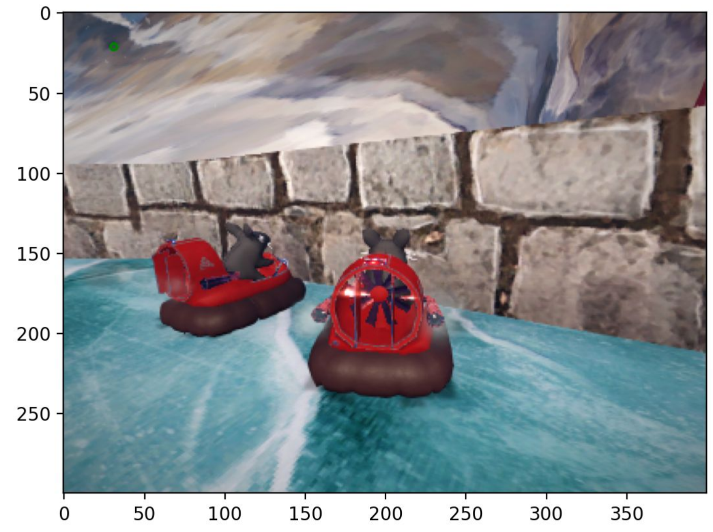

Figure B: Green dot representing where the model thinks the puck is, when there is no puck on the screen.

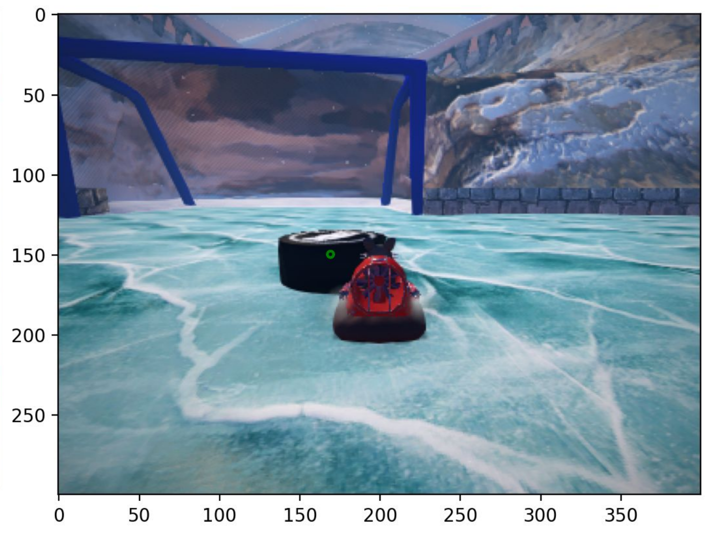

Figure C: Green dot where the model thinks the puck is, when the puck is on the screen.

While training our model, we utilized the following approaches:

- Experimented between MSE and L1 losses - ended up finding more success MSE.
- Used both cropped (~6k) and uncropped (~14k) images for our training data (original pictures of size 300 x 400). Cropped images to save on training time when trying out different concepts and produce a number of “base models”. Once we selected our favorite base model, trained against full size images.
- Final model trained over a course of 3, 20 epoch waves (60 epochs cumulative). Again, mainly done to “check model progress” throughout the course of training due to long training times.

Controller

Overview

The general approach for the controller consists of evaluating our surroundings and then taking action based on a set of predetermined states as described below:

1. Evaluate input image using trained model
2. Determine state value in order: {“kickoff”,“stuck”, “in_goal”, “searching”, “attack”}
3. Act in accordance with determined state

__Init__

Upon player initialization, we established the operation conditions for the game as follows:

- A kart was selected for the player (‘Xue’ in our case),
- Queues were created to save past puck locations, kart locations, states, and actions

- Own-team and opponent goal locations were generated
- Initialized “state lock” boolean

Kickoff

The kickoff logic only occurred at the beginning of a match and was designed to rapidly accelerate towards the center of the map while aligning the kart for a head on shot at the puck.

Stuck

If not in the “kickoff state”, stuck logic kicked in if it was determined that the kart had not moved for a period of time. The stuck logic was separated based on the side of the field and the direction the kart was facing. The kart would then attempt to steer out of that stuck location and towards the last known puck location.

In_Goal

If not in the “stuck” state, inGoal logic was specialized stuck logic to get the kart out of the goal. The logic directed the kart out of the goal based on which goal the kart was in, the direction that the kart was facing, and which “quadrant” of the goal the kart was in, to determine the most efficient direction to drive out.

Searching

If not in the “in_goal” state, searching logic kicked in when the puck was determined to be “out of sight” for three frames. When it was decided that the puck was “out of sight”, the kart would start a full search mode based on its current quadrant and the direction the kart was facing. The kart would then drive a pattern that attempted to cover the map and relocate the puck.

Attack

Finally, if not in the “searching” state, the attack state was the logic was the primary logical controller for the kart. It utilized puck location, kart location, and goal locations to accomplish the following:

- Define the speed and steering angle based on the puck location
- Define the “attack cone” - the angular region representing the opponent’s goal
- Determine the angle between the kart heading and and the “attack cone”
- Steer towards the “attack cone” and drive the puck on net if the puck was right in front of the kart
- Check if it just exited the “searching state” and slow accordingly

Future Ideas

Follows is a list of techniques that we wanted to try but did not have time to implement:

- Defensive logic
- Second model to identify other players
- Third model to identify items
- Implement logic to manage kart to kart communication
- Implement logic to manage item use

Lessons Learned

We had several lessons learned while attempting this project:

1. *Data getting “smushed”* - this affect the model’s ability to parse the image and thus affected training. The TA’s guidance in resolving this was invaluable.
2. *Defining controller actions* - we initially tried to define everything up to edge cases at once in our controller which caused a lot of issues. We finally realized that we should start with core functionality and expand out.
3. *Data cleaning* - our initial models struggled to learn effectively due to the data we provided. We realized that we had not distinguished between photos with and without the puck, and that this was causing a lot of confusion in our model. We defined each of these (described above) and it greatly improved our model performance.