CS 342 Deep Learning

Final Project: SuperTuxKart ice-hockey player

## Introduction

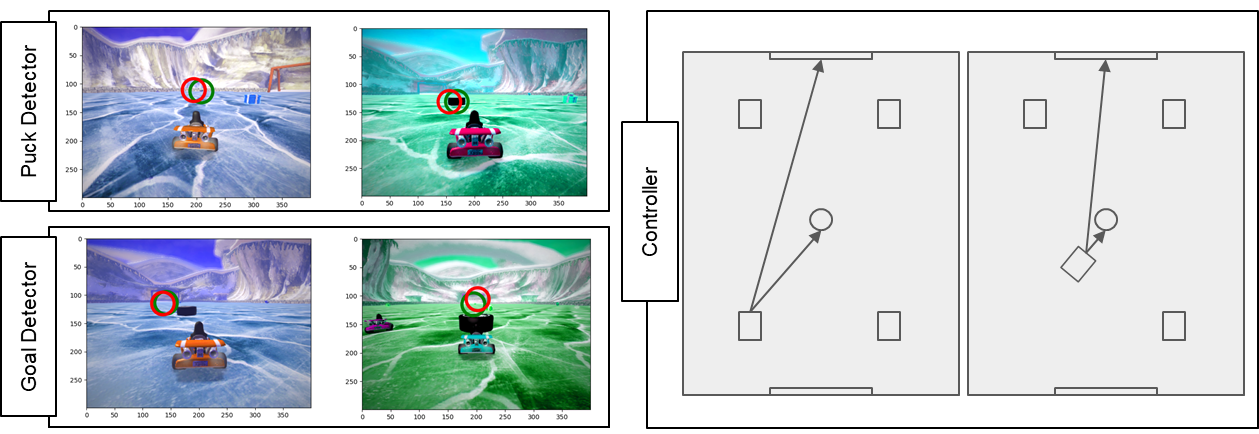
The main object of this project is to create a program that could score as many goals as possible and win the match against agents provided in 2x2 tournaments in which points are earned by propelling the puck past a goal line and into a net. After the time limit is reached, the winning team is one that scores more goals.

## Approach

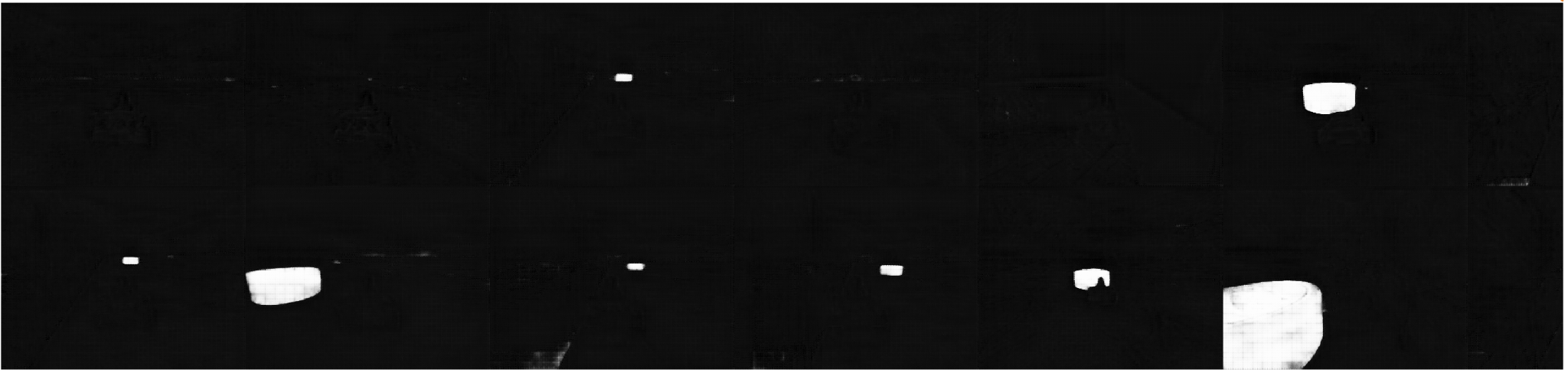
We experimented with both state based and image based models to determine the one that generates a better outcome. We split into two teams to work on each approach and found that each has several challenges as described below.

### Image based

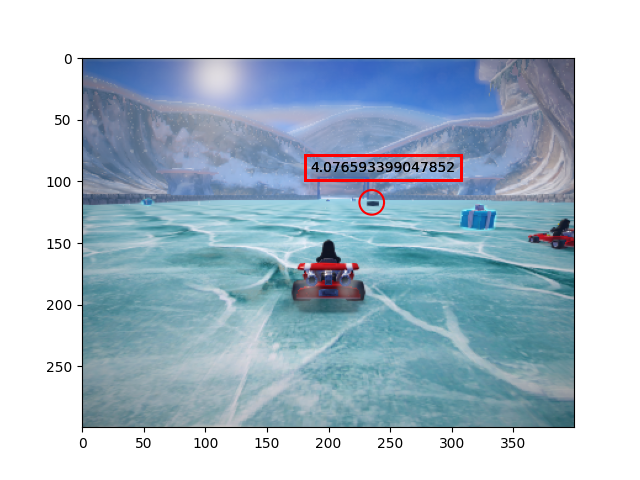
For our vision based approach, we generated a dataset to implement a planner to predict puck location. During this process, we realized that in some of the images, the puck doesn’t show up but the model still attempts to predict an aim point. Realizing this is not a viable solution, we thought about training another classification model that can detect puck’s existence in frame. If it does then we can utilize the planner to detect its location. After several trials and errors we found out that the model works good if the puck is present but in an event that the puck is not presented in the image we tend to get false positives that could lead to terrible results by the controller.



In our second attempt, we utilized a detector similar to the master solution in homework 4 with a few additional layers as we observe it makes the model more stable with the dataset we use. The image based agent consisted of a detector to detect the puck. The model was able to learn well and reach . Our initial dataset only included 2,000 images. We found that the model trained on this dataset perceives the pole as the puck and keeps getting stuck in the net. Therefore we generated a second set of dataset with around 14,000 pairs of images and labels with some focus on the positions where the agent easily gets stuck and mistakes other objects for the puck. Afterwards, the trained puck detectors were used in a custom controller that tries to move towards the puck and hit the puck in a way that it moves towards the target goal. The figure below shows the puck detector model’s performance during training. The model was able to detect the puck’s location pretty well during training as indicated in the figure below. We also tested changing the data from rgb to grayscale to improve the accuracy, however this did not lead to significant improvements as our rgb model was performing great.



We then base the confidence of the puck detection on the score generated by the model. Using this figure, we embed the function in the controller to determine the agent’s course of actions.

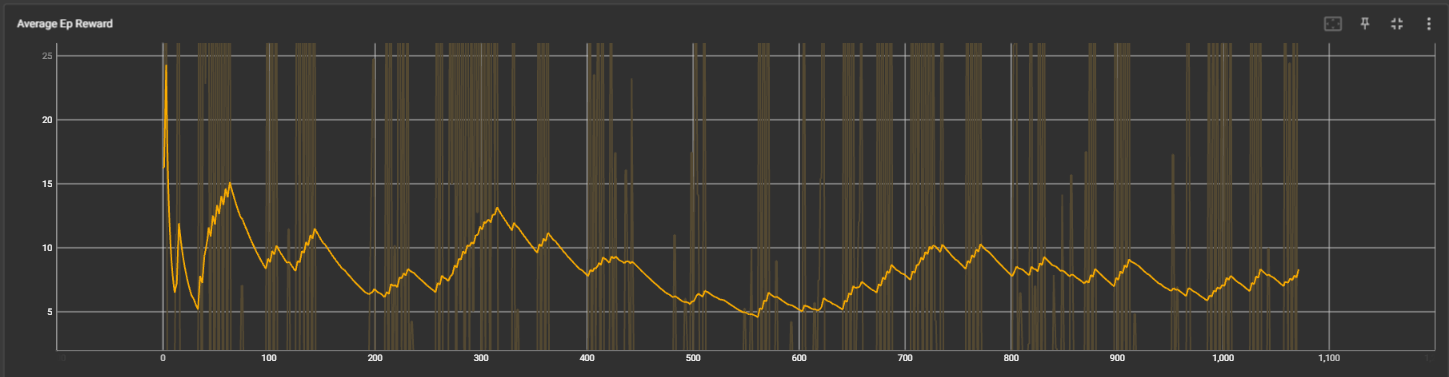


#### Controller

Instead of trying to detect a number of possible maximum detections, we set it to 1 to reduce the computational efforts required. The basic strategy of our agent is to let it speed up when it detects the puck in frame and try to score. When it loses track of the puck, it will move backwards a few steps for a “cool down” period of number of steps. Beyond this threshold, the agent tries to move back to the goal and wait for its opponent to bring the puck to its own goal then try to steal it. After some research, we found that there are 3 categories of karts: light, medium and heavy. The one most suitable for soccer games are those in the light category. Testing results show that our agent yields the highest score when it uses kart “kiki”. Furthermore, at the beginning of the match we set the kart a way to move straight forward to ensure that we detect the puck.

### State based

We tried implementing the Soft Actor Critic model (SAC), an off-policy model that seeks to maximize entropy as it learns in order to fully explore the action space while converging on a solution. SAC models utilize two different networks. The first is the actor network that takes in environmental variables which in our case are features/states of the game. These features of the environment are fed into the agent and through a few basic linear layers, it returns the 4 different actions: acceleration, steer, brake and drift. The complication of the problem with the data type is that two of the actions, acceleration and steer, are continuous, and the others are discrete. Networks are usually built with either one data type or the other as an output. We tried to implement a hybrid SAC that can return two continuous actions and 2 discrete actions while still maintaining the ability to run the gradient through and calculate a loss function, utilizing practices outlined in the cited Delalleau paper. This involved tweaking the critic network so that it predicts the Q values of the environment given any of the discrete actions. We integrated braking and drifting as discrete actions by mapping them from four values produced from the same hidden layer that provides the parameters for the two continuous actions. These four values are fed as probabilities into a categorical distribution, which is then sampled to choose from four different actions: Braking and drifting, no braking and drifting, braking and no drifting, and neither braking or drifting. The hardest part of dealing with this model was tweaking the input features and reward function to make it learn efficiently and not get stuck in local minima. A rather comical bug with our early reward functions resulted in the karts circling close to the ball but never kicking it. It turns out it was getting more highly rewarded for approaching the ball than actually moving it down the field! The method of training the model posed even more difficulties.Originally we rigged up a system for games played between the class provided AIs to be saved into a replay buffer which was then sampled from and learned off of. The idea would be that we’d be able to bootstrap off of the performance of the TA agents. In practice learning off-policy like this was remarkably slow, so we switched to learning after every step of play of our own agent. These data points marry together the fact that reinforcement learning problem sets can prove challenging to effectively converge on a global minima. In the state-based actor realm, there were a number of options that were not fully explored, such as imitation learning that could have provided a different solution, however the sheer number of reinforcement learning algorithms published give credence that each problem set is unique. Ultimately, after various attempts at training and optimizing our state based models (some trained in excess of 24 hours), our group elected to prioritize the image based agent.



*Even with finely tuned parameters, training of the state based agent was slow*

## Conclusion

Splitting the team into two working groups proved extremely beneficial, as it allowed us to explore both options of state and image based agents. Although later state based models showed signs of learning, the rate was too slow with time constraints for submission and our group pivoted to focus on the image agent. We found out that

Citations

1.Delalleau, Olivier, et al. “Discrete and Continuous Action Representation for Practical RL in Video Games.” *arXiv:1912.11077v1 [cs.LG] 23 Dec 201*, Dec. 2019, https://doi.org/10.48550/arXiv.1912.11077.

https://drive.google.com/file/d/1kAJwZjyEThiA6zOUPlHpSnyi6VajQwnr/view