A survey of image and state-based game agent implementations in SuperTuxKart Ice Hockey

Jing Yi, Mitchell Atkinson, David Kidd, Nathan Nix

Abstract

This report covers our development of an agent designed to play games of "SuperTuxKart Ice Hockey". We investigate a number of image and state based approaches to training an agent to score as many goals as possible. Our training dataset consists of roughly 4000 simulated games with 8,685,832 x 11 observations and 8,685,832 x 3 action labels. These 8.7 million data points are fed through a shallow linear network to produce three kart actions (acceleration, steer, and brake). After multiple trials, we create an imitation agent that is a close approximation of Jurgen averaging 28 goals across 32 games.

1. Introduction and Motivation

In this project, the team was assigned the challenge of training agent(s) to play 2-v-2 Ice Hockey in SuperTuxKart. Similar to Rocket League, SuperTuxKart requires the player to drive a kart around the field with the objective of steering a puck into the goal. The game is played for up to 1000 frames, or until 3 goals are scored. Our primary objective was to ensure that the agent could consistently score at least one goal per game. We were allowed to choose to develop either an image-based agent or a state-based agent. An image-based agent relies on visual images to detect the puck, and has access to team players data but nothing else. A state-based agent has all the information including player states, opponent states, and the puck state. Each approach has its own advantages and challenges.

Throughout this project, we tried different approaches to tackle this problem. Initially, we attempted to build an image agent that achieved high accuracy on puck detection within images; however, due to the intricate logic required to accurately track and steer to the puck's location and score goals, we formally transitioned to a state-based approach. From there, we explored two approaches simultaneously for developing our state-based agent: reinforcement learning and imitation learning. Although we ultimately opted for the imitation learning approach, the progress made in our previous attempts proved invaluable as all the initial work provided us with essential building blocks for solving the problem including the generation of training data, useful feature extraction,

lessons learned from early missteps and unexpected roadblocks, and generally, experience developing within the SuperTuxKart environment.

Our use of imitation learning to train our agent to play the dynamic game SuperTuxKart is akin to projects like DeepMind's AlphaStar. Initially, recordings of human players were used to represent the "expert" player for the model to imitate. After establishing a baseline, the model moved to being updated via reinforcement learning where it was rewarded according to results of games played [1]. These methods allowed the model to create its own decision making process which enabled it to adapt and make decisions under uncertainty. Success of these approaches in games show their potential for these types of models to be applied in various industries with real world application.

In this paper, the Methods section includes all the methods we tried: vision learning, reinforcement learning, and imitation learning. It outlines each method's brief background, experimental design and setup, and model architecture. The Results section presents the performance metrics and accomplishments (or aspirations) of each learning approach. Since imitation learning is proven to be the most effective way, we mainly discuss the optimizations and their influence on the results. Finally, in our Conclusion, we review our final graded results of our imitation learning model, a recap of our strategy shifts and the lessons learned, and ideas we would consider for each approach in the future. Due to the time limitation and strategy changes, we believe this model can be further optimized to achieve even higher scores and cover these in our Conclusion as well. For example, continuing to optimize the imitation learning parameters, DAgger training implementation, designing a manual expert agent that is better than Jurgen, and applying reinforcement learning.

2. Methods

2.1 Vision Learning

Deep Learning offers many different vision learning concepts and techniques including image classification, semantic segmentation [2], object detection [3], and vision-based driving. We attempted to use all of these techniques to build an image-based agent. Here, we will cover this experience in detail including: how we generated our training data and labels, training and performance, and the roadblocks and pitfalls we faced throughout.

2.1.1 Image Data Generation

To generate training data, we decided to create a dummy agent so that we could control the agent's behavior as much as possible to produce the image results we desired for training. This agent was initially coded to do just one thing: chase the puck. While this kept implementation simple, it had a couple drawbacks such as, getting stuck against walls and inside goal gates, and overshooting the puck with poor steering. To address these

problems, we optimized the dummy agent adding logic that would identify a stuck kart and get it back to chasing, as well as, adding logic to decrease acceleration significantly as we approach the puck. These improvements helped enough for us to generate a considerable amount of image data with the puck in frame. In total we created 10,000 images by simulating 10 games (5 games as the red team and 5 as blue at 1000 frames per game) between our dummy agent and the provided agent Geoffrey which was selected after careful review of the performances of each of the provided agents (Geoffrey, Yann, Yoshua, and Jurgen). Specifically, we noted that games between Geoffrey and our dummy agent consistently resulted in scoreless games which we deemed ideal for generating the "puck heavy" image data we required.

We also generated labels for our image data during the simulated games. These labels include the visibility of the puck, the kart's aim point coordinates, and the kart's distance to the puck. One challenge we faced with the aim point coordinates was the transformation from coordinates that make sense "in game" (i.e. those coming from the game state itself) to those that would make sense in the context of the image. Another way to think of this is that we needed to take the coordinates of a puck at a bird's eye view of the arena and transform them to coordinates from the point-of-view of our agent's kart as seen from our sampled images. We designed functions to achieve all these transformations[4]. These functions also allowed us to check if our aim point had correctly targeted the puck or not. If the fully transformed aim point coordinates were outside of the arena, then the label would return a false detection on the image.

2.1.2 Model Architecture

A fully convolutional network (FCN) that is similar to the U-Net [2] was implemented in our vision model (fig. U-Net Design). There were three convolutional blocks with increasing channel sizes [16, 32, 64], and three up-convolutional blocks with channel sizes in reverse order [64, 32, 16]. Our FCN also included residual connections between the same-channel convolutional and up-convolutional blocks. The output of our network was a heatmap that either contained the image coordinates of the detected puck or no coordinates if the puck could not be identified.

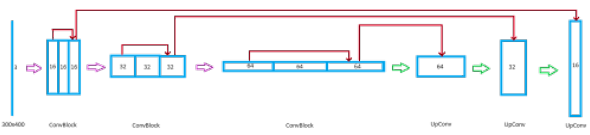

Figure 1: Our implementation of the U-Net Architecture [U-Net Design]

2.2 Reinforcement Learning

For our reinforcement learning approach, we surveyed a number of potential on-policy and off-policy algorithms and delved deeper into Proximal Policy Optimization (PPO).

To support this, we implemented our own gym environment to shape the observation, action, and reward space as effectively as possible.

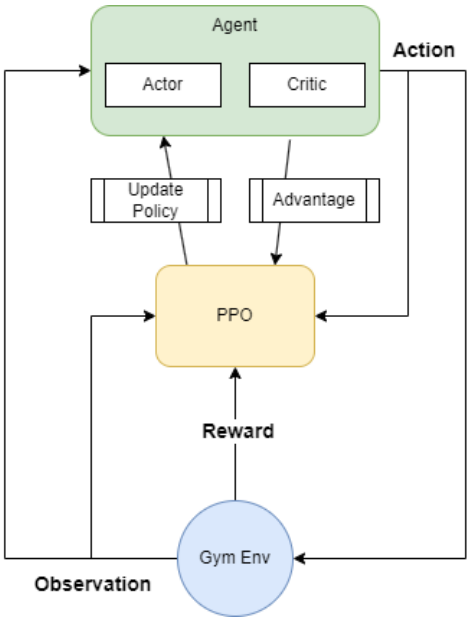

Figure 2: Our implementation of Seer training algorithm and process flow [5]

2.2.1 Gymnasium Environment

In implementing our PYSTK Ice Hockey environment we heavily reduced the observation space to a smaller subset of calculated features, a simplified action space to mimic a video game controller, and a specific reward function with the primary objective being to reward trajectories and actions that scored goals.

Action Space

SuperTuxKart accepts both continuous and discrete values to control the acceleration, steering, drift, brake, and turbo properties. Each action is independent of each other. Their combinations create a very large potential action space. To simplify this problem, we followed research that suggested simplifying the actions into a multi-discrete action space [6] in Table 1. This made the actions simpler as if they were button presses on a controller and helped reduce the amount of possible actions from a near infinity space down to 48 potential actions.

Table 1: Action Space

Original Space	IceHockey Env Multi-Discrete 2
Acceleration: Continuous [0, 1]	Acceleration: Discrete (2)
Steer: Continuous [-1, 1]	Steer: Discrete (3)
Drift: Discrete (2)	Drift: Discrete (2)
Brake: Discrete (2)	Brake: Discrete (2)
Nitro: Discrete (2)	Nitro: Discrete (2)

Observation Space

In most sampled Gymnasium environments, the observation space typically consists of the entire frame. However, one of the challenges of this project is that the observation space is massive, and there are many supplements and derivations of additional features that are essential for our model to learn. We broke down our observation space into 38 features in Table 2 as follows:

Table 2: Observation Space

Angular Features	Distance Features	Velocity Features
Kart to Puck Angle [x,z]	Kart to Puck Distance [x, z]	Kart Velocity [x, z]
Kart to Puck Angle Diff	Kart to Goal Distance [x, z]	Kart Velocity [x, z]
	Puck to Goal Distance [x, z]	Kart to Puck Velocity [x, z]
		Puck to Goal Velocity [x, z]

Note: The euclidean norm equivalent of each feature containing coordinate values was also included as part of the observation space

Reward Function

The critical component of any reinforcement learning algorithm is the shaping of the reward function. In this project, one of the key challenges given the sparsity of the event occurring was determining how to efficiently reward the agent when it accomplished the objective. Another key challenge we had to overcome was how to encourage it to solve the problem as fast as possible without providing too much guidance (thus preventing the agent from actually making its own deductions and state-action mappings effectively). Since our game and objective is most similar to that of Rocket League, we decided to leverage a similar reward function to what was used in training Seer [5], with some key deviations. As opposed to Seer, our objective was to score as many goals as possible, regardless of whether we actually won the game. During an entire episode (which was a single game played until one team scores 3 points or 2000 frames had elapsed), we rewarded our agent for meeting specific objectives and step awards. They are as follows:

- Objective Rewards
  - Goal Scored: 1 points was awarded if a goal was scored
  - Winning the Game: 0.75 points were awarded if the game was won, with a 2x multiplier if the game was 3-1 or 3-0 implying our agent overwhelmed the opponent
- Step Rewards (All features below had weights that would sum to 1)
  - Kart to Puck Distance: The agent was given an increasing award as it moved closer to the puck

- Kart Behind Puck: To encourage the kart to move towards the ball first and not pass it, the agent was rewarded if it was closer to the puck than the goal
- Facing Puck: A reward was given If the absolute value of the angle difference of the kart to the puck was less than 0.05
- Velocity to Puck: A reward between -1 and 1 was given if the velocity of the kart was away or towards the puck respectively
- Velocity: To encourage the kart to always be moving, a weighted reward was given based on the squared euclidean norm of the kart's velocity
- Kickoff: This reward is the same as the Velocity to Puck reward, but only awarded if the ball stays in the initial position.

The final calculation of the reward was then clipped to a range of [-5, 5] in order to ensure that an overwhelmingly bad or good game didn't push the agent's behavior too far from the norm.

2.2.2 Proximal Policy Optimization

While REINFORCE is a common baseline reinforcement learning algorithm, it is sensitive to perturbations and noise which can result in unexpected policy shaping. Given the expansive observation and state space in the environment, ideally, we aimed to ensure that updates to our learned policy did not deviate too far from its current state. This thinking, and subsequent research into similar Rocket League algorithms[5,7] led us to implementing Proximal Policy Optimization (PPO). In the below subunits we discuss some of the implementation details in our training algorithm.

Advantage Actor/Critic Network

To reduce the high variance and noisy gradients usually associated with REINFORCE updates, we employ an Advantage Actor-Critic network[5]. This network consists of two shallow fully connected networks, where the Actor generates probabilities for each of the actions in the action space. The critic network uses the same initial FCN block as the actor, but instead produces a single value based on the resultant features. These outputs are stored in a Memory Buffer, along with specific actions, observations, and rewards in order for the PPO algorithm to calculate and apply policy gradient updates.

Trust Region Optimization and Clipped Surrogate Objective Function

To limit the amount of change to the policy, PPO leverages Trust Region Optimization[8]. In order to achieve this, we implemented the Clipped Surrogate Objective Function given below:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

Where $r_t(\theta)\hat{A}_t$ is the unclipped objective function and $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ is the

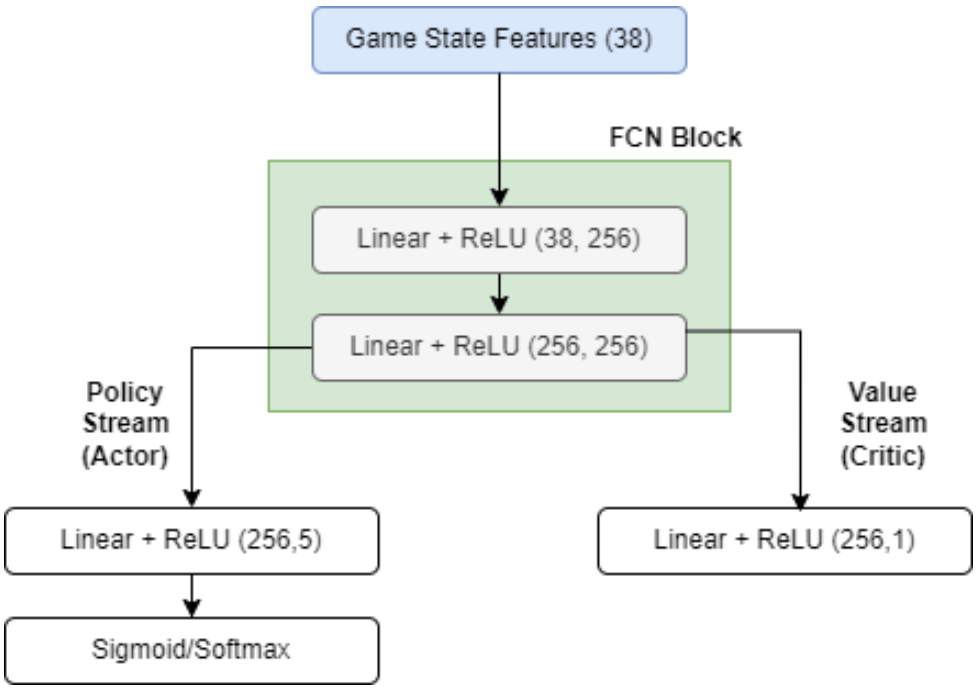

Figure 3: PPO Architecture

clipped objective function. The second function helps to ensure that the first function doesn't swing the policy too far, resulting in instability of the network.

Finally, we leverage Bernoulli and Categorical distributions to generate multiple samples of the action to build the initial policy.

2.3 Imitation Learning

Without a clear reward function, imitation of a high performing agent (i.e. one that consistently scores a lot of goals) emerged as a viable approach. The imitation-based strategy leverages the observed success of specified agents as a template for decision-making, circumventing the need for a reward mechanism. However, it is important to acknowledge the potential limitations of this approach. While imitation may yield short-term gains in emulating successful behaviors, it may not necessarily lead to optimal performance in all scenarios. In other words, the agent to be imitated caps the performance of the imitation agent.

2.3.1 Agent Selection

Among the five given agents, Jurgen scored the most goals on both red and blue teams (see Table 3). While the grader had a 50% chance of flipping the team sides among the 8 games we sampled, for convenience we assumed that the team put through the grader was on the red team. Although we were only viewing a limited snapshot of each agent's performance, Jurgen's consistency, and near doubling of goals scored versus the next best agent, made Jurgen an ideal candidate for imitation.

Table 3: Expert Agent 2-v-2 Results

	Geoffrey	Jurgen	Yann	Yoshua	Total as Team 2 (Row)
Geoffrey	4:4	14:7	5:6	N/A	17
Jurgen	7:14	7:7	4:8	2:9	38
Yann	4:4	8:4	1:1	2:4	13
Yoshua	4:0	12:2	4:2	0:0	4
Total as Team 1 (Col)	19	41	14	4	
Total Score	36	79	27	8	

2.3.2 Data Generation

To generate data, we created two separate datasets. Our initial dataset was comprised of 200 games of Jurgen against 4 opponents. In each simulation, we ensured we captured data where we swapped sides, but left the starting location of the puck in the center, resulting in 400,000 data points. In the expansive dataset, we simulated roughly 4000 games of Jurgen playing a variety of opponents, starting from both sides of the field, and with a random distribution of starting puck locations. This dataset ended up including 8.7 million data points. Our overarching objective was to ensure that the dataset encompassed a wide range of game circumstances across the entire field, capturing the full spectrum of Jurgen's performance in diverse contexts.

The next decision in our data generation was the number of features we would need to extract for training. We investigated Jurgen's 11 features against the other expert agents' 17 features to determine which would be most useful. They each shared a few features including the kart locations, the puck location, and a feature indicating how far away each kart is from the puck. While we carefully reviewed and considered many of the available observations and action features available within SuperTuxKart for our agent, we decided instead to go with a more straight-forward approach of having our agent mirror the 11 features Jurgen utilized exactly. This allowed us to imitate Jurgen as closely as possible.

2.3.3 Network

We used the simple linear Action network containing one input layer, four linear hidden layers (with a ReLU activation layer), and one output layer (fig. Action Network Structure). We opted for a shallow and fully linear network since our inputs and outputs were relatively small and not multi-dimensional such as the image tensor inputs seen in vision based learning. Thus, neither a deep network, nor a convolutional network felt necessary in this case.

2.3.4 Loss Function

The three outputs to be obtained from the model are acceleration, steering and braking. The first two outputs are continuous float values while the third is a binary boolean variable. Testing was conducted to evaluate the potential benefits of utilizing two different

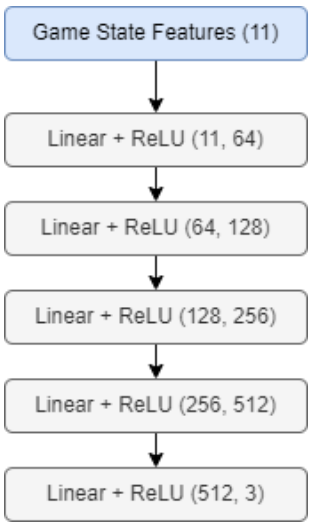

Figure 4: Action Network Structure

loss functions for the different output types. The results from these two loss functions could be combined into a single loss for the model.

Mean Square Error (MSE) was to be used on the first two outputs. Since the network of the imitation learning state agent would utilize linear layers it seems like using MSE would be the best choice for the continuous variables. As the name implies MSE squares the error between the model's predicted value and the target value. The MSE is always differentiable which lends itself to work well with gradient based optimizers such as ADAM or SGD [9]. The model also strongly penalizes predictions that vary greatly from the target which helps result in a more precise and accurate model. [10]

Binary Cross-Entropy (BCE) was the loss function which was going to be used for the binary output. Similarly to mean square error, BCE heavily penalizes higher errors in order to push the model to adjust accordingly.The difference here is that BCE uses probabilities that the output is assigned to a specific category, either 0 or 1 [11]. These probabilities represent the model's confidence level regarding the classification of the output given the input features.

3. Results

3.1 Vision Learning Results

With our image data and labels ready, we were able to train our CNN model to identify the puck within images at a 90% accuracy when it was in view. However, if the puck gets lost, getting the historical locations of the puck was not good enough to help the agent relocate the puck. Given our time constraints and what we felt was needed to progress this approach, we decided to switch to the state agent. Despite that, we learned many valuable lessons to apply to the later imitation design. Designing the simulated game environment for gathering image training data was very similar to the data generation of

later imitation learning. When we designed the dummy agent, we understood the features extracted from the states. Overall, these designs and thoughts have been applied to the imitation learning later and were proven to be effective.

3.2 Reinforcement Learning Results

The challenge that reward function shaping poses cannot be understated. In our initial attempts, we tried to fine tune the entirety of the reward function at the onset based on the referenced literature. However, hand adjusting the weights of each of the step awards based solely on reviewing the output proved to introduce new and interesting unintended behavior. For instance, during one of our training runs we saw that the agent's average reward score was steadily improving during each epoch. However, upon review of a sample game, we realized that the agent had learned to focus on getting to the goal as fast as possible. In order to do that, the agent had unexpectedly learned to avoid the puck and drive around it so that it would not get stuck in a deadlock when the other team approached it as well.

We then switched tactics and attempted curriculum learning [12] in order to help guide it towards a particular goal. The curriculum we intended to work through had the following steps:

1. Learn how to turn towards the puck (no opponent)
2. Learn how to move towards the puck (no opponent)
3. Learn how to move the puck and stay close (no opponent)
4. Learn how to move the puck to the goal (no opponent)
5. Learn how to score a goal

Once our agent learned that, we would introduce an opponent. Unfortunately, we did not get past step 2, when our agent learned instead that it received a fine enough reward by just getting close enough and racking up points.

3.3 Imitation Learning Results

Initially, we trained the model with the smaller dataset without optimizations, but the resulting agent was only able to score an average of 7 goals over 32 games. To improve performance, we adjusted the loss function and augmented the dataset by adding more "score" adjacent data points and noise to the input features. We also experimented with different hyper-parameter settings, such as modifying the optimization function (e.g., increasing the learning rate), but these changes led to seemingly insignificant improvements. Keeping all other factors constant, the removal of the BCE function led to an immediate decrease of 60% in the lowest loss previously achieved when using both loss functions.

Before removing the BCE loss, observations showed that the BCE loss decreased at a faster rate than MSE. While MSE continued to decrease steadily during training (in Chart 1), BCE would plateau and in some cases even increase. This behavior hindered the overall loss reduction, hinting potential overfitting on the binary output and impeding the model's improvement. Efforts could have been taken to establish a weighting system to prioritize the MSE while allowing BCE to continue to work with the binary output while not hindering the models overall performance. This idea was not explored further due to the time limitation and other work still to get done.

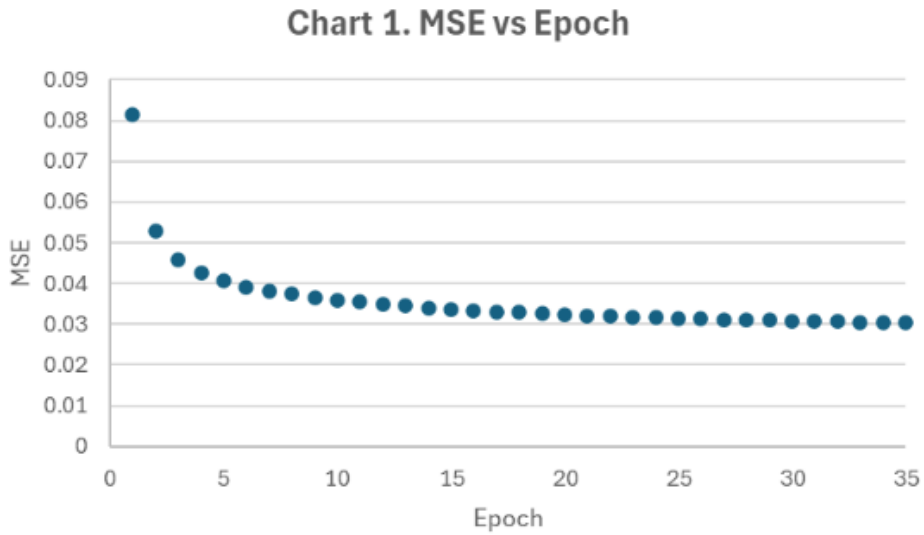

Figure 5: Imitation Learning MSE Loss Results

3.3.1 Data Augmentation

The initial imitation learning proved to be a simple and efficient supervised learning network, but there were also some notable disadvantages of this approach. When we trained the model the first time, despite a MSE decrease from 0.08 to 0.03, the total goals scored by our agent underwhelmed, ranging from just 0 - 7 total goals scored. Manual review of the game recordings further confirmed our agent's sub-par play as we could see our agent occasionally wandering aimlessly or at a standstill. The initial model proved unstable as well scoring anywhere from 50-100+% less goals depending on the machines in which we tested. Specifically, the average number of goals we ever scored across 32 games with this model was 7.

To improve these results, we pursued data augmentation believing that an improved dataset could increase our model's performance and consistency. We enhanced our data by doubling the number of games in which either team scored hoping to reduce the learning done in scoreless games and to increase the learning done in higher scoring games. After training with this version of our enhanced data, our scores marginally increased, but also with some improved consistency. Specifically, we saw our average number of goals scored across 32 games increase to 12.

Our next improvement was to triple the number of frames in our dataset that captured images just before and after any goals were scored (100 frames before and after any score change to be exact). This heavily increased the amount of learning done in goal scoring situations which was the most critical game strategy for our agent to learn. Finally, we added 10% random noise to each of the 11 features. This "noisy" data and the original augmented data were then used together to train the model. This combined with a doubling of training time to 200 epochs, the MSE decreased back down to 0.03 as previously achieved with the final model scoring more efficiently and consistently than ever before yielding a maximum of 19 total goals scored across 32 games.

3.3.2 Expansive Data Set

With our new understanding of how expanding the dataset improves performance, we came up with a new hypothesis - what would happen if we fed a massive dataset that more fully represented all possible state and action spaces for our expert. We fed our dataset consisting of 8.7 million data points, which included randomly generated starting puck locations as our noise, and immediately saw a jump in performance. By providing all of this data, which included both scoring and non-scoring scenarios, our model was able to learn how to best emulate the Jurgen agent. After only 9 epochs, our agent was able to score on average 28 goals out of 32 games, which was 1 goal shy of matching the average observed by our expert.

4. Conclusion

In conclusion, we found that imitating the Jurgen agent, our chosen expert, was the most efficient and effective approach to meeting our objective of scoring at least one goal per game. Learning from an expert often proves to be the best way to reach a baseline behavior, even if the performance ceiling is capped at the same point as the expert. While the upfront cost of generating 8.7 M data points may be daunting, when done correctly it ensures that you have mapped enough of the expert's action-space value pairs for solving our objective.

While we decided to move away from vision learning as a solution to this project, we learned lessons that gave us ideas to explore in the future. With more time to study the mechanics of the game itself, we recognize that we could develop a significantly better dummy agent that was better at, if not close to masterful of, the game on its own. From here, generating a large amount of data would be greatly simplified as simulated games with an expert agent would be able to generate a much stronger dataset of images as those games would capture substantially more game states and significantly more effective game strategies.

Moreover, successfully developing a model that emulates the Jurgen agent opens up new avenues of approaches to the challenges we encountered during the project. For

instance, instead of manually crafting a reward function, we could have instead approached it from an inverse reinforcement learning perspective [13]. In this methodology, instead of developing a reward algorithm to shape our policy, we could use the expert agent's policy to derive a reward function through a model-free approach. In this way, we could learn step by step how the agent interacts with the environment and update a complex reward function accordingly.

Furthermore, much of the work in this project was spent on understanding the data, generating datasets in various methods, and screening out the critical data. While most research focuses extensively on the algorithmic and processing aspects of deep learning (and for good reason), the quality of the dataset gathered is perhaps the most important aspect to success. Spending the time to ensure a strong data generation and labeling process is paramount, and was a key component to our successful completion of this project.

5. References

[1] The AlphaStar team, "AlphaStar: Mastering the real-time strategy game StarCraft II," Google DeepMind, Jan. 24, 2019. https://deepmind.google/discover/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii/

[2] Ronneberger, O., Fischer, P., and Brox, T., "U-Net: Convolutional Networks for Biomedical Image Segmentation", International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 234–241. Springer, 2015.

[3] Zhou, X., Wang, D., and Krahenbuhl, P., "Objects as points", arXiv:1904.07850, 2019.

[4] A. Anwar, "What are Intrinsic and Extrinsic Camera Parameters in Computer Vision?," Medium, Mar. 01, 2022.

[5] N. Walo, F. Perez, C. Advisor, and N. Perraudin, "Seer: Reinforcement Learning in Rocket League," 2022. Available:

[6] Kanervisto, Anssi, Scheller, Christian, Hautamaki, Ville, "Action Space Shaping in Deep Reinforcement Learning arXiv:2004.00980v2 [cs.AI] 26 May 2020

[7] M. Pleines et al., "On the Verge of Solving Rocket League using Deep Reinforcement Learning and Sim-to-sim Transfer." Accessed: Apr. 8, 2024. [Online]. Available: https://arxiv.org/pdf/2205.05061

[8] S. Padhye, "Building a Reinforcement Learning Agent that can Play Rocket League,"

Medium, Mar. 25, 2023. (accessed Apr. 8, 2024).

[9] Amanatullah, "Demystifying Loss Functions in Deep Learning: Understanding the Key Metrics for Model Optimization," Medium, May 24, 2023. (accessed Apr. 15, 2024).

[10] R. Alake, "Loss Functions in Machine Learning Explained," www.datacamp.com, Nov. 2023. (accessed Apr. 19, 2024).

[11] S. Saxena, "Binary Cross Entropy/Log Loss for Binary Classification," Analytics Vidhya, Mar. 03, 2021. https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/

[12] S. Narvekar et al., "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey," Journal of Machine Learning Research, vol. 21, pp. 1–50, 2020, Accessed: Apr. 15, 2024. [Online]. Available: https://arxiv.org/pdf/2003.04960

[13] S. AI, "A brief overview of Imitation Learning," Medium, Sep. 19, 2019. https://smartlabai.medium.
brief-overview-of-imitation-learning-8a8a75c44a9c