Image-based Agent for Ice-Hockey Gameplay in SuperTuxKart

CS - 394D Deep Learning
Spring 2023

Authors

Ashlyn Bain
Gabriela Lozano
John Noynay
Sailesh Kandula

Abstract

This document aims to provide a detailed explanation of the process of creating a competitive ice-hockey player in SuperTuxKart by using an image-based agent. The agent needs to process images quickly, requiring a deep neural network and optimized code to run efficiently. Our team selected image-based agents because they have advantages over state-based agents, including processing complex visual features and learning to identify relevant features independently. After initially going with a state-based agent, our team switched to an image-based agent using a convolutional neural network (CNN) to generate peaks in the images and gather game state values. Next, our team experimented with different loss functions and found that a combination between *BCEWithLogitsLoss()*, *FocalLoss()*, and *MSELoss()* provided the best results. Finally, utilizing image analysis, our team designed a player that works to gain possession and score upon detection of the puck. The player increases speed and uses calculated angles to direct the puck towards the goal when positioned near it, and moves towards the goal until the puck is detected if neither agent detects it. In this report, our team dives deeper into the agent we created as well as the steps we took to come to the specific decisions that we made.

Introduction

SuperTuxKart is a 3D racing video game that is available for open-source use. The game offers a variety of game modes, including single-player and multiplayer options, and provides players with numerous character, kart, and track options for an immersive racing experience. This popular racing game is compatible with various operating systems, including Windows, macOS, Linux, and Android.

Drawing from the gameplay for the ice-hockey game, which involves strategies for chasing, positioning, and scoring the puck against the opposing team. The underlying mechanics of the ice-hockey game are similar to that of Rocket League, where the objective is to outscore the opposing team. By leveraging our firsthand experience

with the game, we aim to build an AI player that can effectively navigate the game environment, make strategic decisions, and score goals.

Under the hood, SuperTuxKart uses convolutional neural networks (CNNs) to train agents to recognize and respond to visual cues within the game. For example, CNNs can be trained to identify the position of the puck, the location of other players, and other key features of the game environment. This enables the agent to make more informed decisions about how to move and where to aim, making it a more competitive player.

Compared to state-based agents, image-based agents have various advantages that make them a preferred choice in many scenarios. One significant benefit is their suitability for environments with complex visual features that are difficult to represent in a state vector. This provides a more comprehensive and precise understanding of the environment, leading to superior performance. Unlike state-based, image-based agents learn to identify relevant features independently, without depending on pre-defined state representations.

After researching the ice-hockey gameplay in SuperTuxKart and the key differences between image-based and state-based agents, our team decided that using image-based agents was the best approach in developing a competitive player. The use of image-based agents would enable us to process visual information from the game environment, allowing our agents to learn and make informed decisions about the location of the puck and the other players.

Initially, our team attempted to create a state-based agent but struggled to implement a single deep neural network without handwritten components. We switched to an image-based agent, utilizing a convolutional neural network (CNN) to generate peaks in the images and gather game state values.

Code

The following sections lay out the contents of the code that controlled, modeled, and trained the designed player.

Player

The player designed by our team is driven by a Team class that initializes variables, initiates a new match, and employs image-based processing for decision-making for each player. In each step, the *act()* method is invoked to determine the next action for each player. This method utilizes a pre-trained object detection model to extract the image and state of each player and detect the puck. Once the puck is detected, it decides whether to pursue the puck or remain in the

current position, based on the relative location, in coordinates, and direction of each player's kart with respect to the puck.

Controller

Our team conducted extensive research on similar games to determine the best approach for our game. After careful consideration, we decided that having one agent focused on scoring goals and the other on preventing the opposing team from scoring would provide a comprehensive understanding of the game and cover both offense and defense.

To implement this approach, we designed and built a CNN that processed real-time images generated by the agents. This allowed us to quickly detect and determine the location of the puck and assign our agents to their respective tasks.

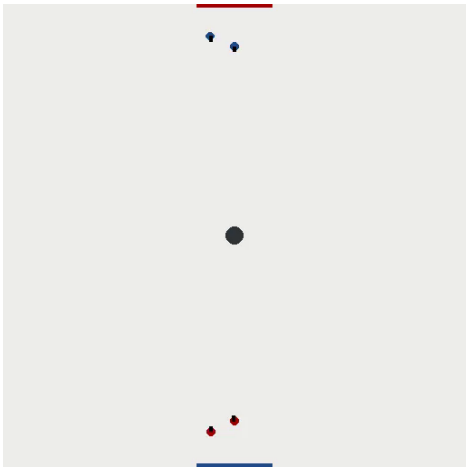

Figure 1

The training set that we created contained 1,200 images. This allowed us to accurately and efficiently train our model. An example is shown above in *Figure 1*. In this figure it is evident the puck is located in the center of the rink and the coordinates of and direction to that point were also generated. In *Figure 2*, which is another generated training image, you can see that the puck is located on the bottom right portion of the image. The respective coordinates of and direction to that puck location were also recorded.

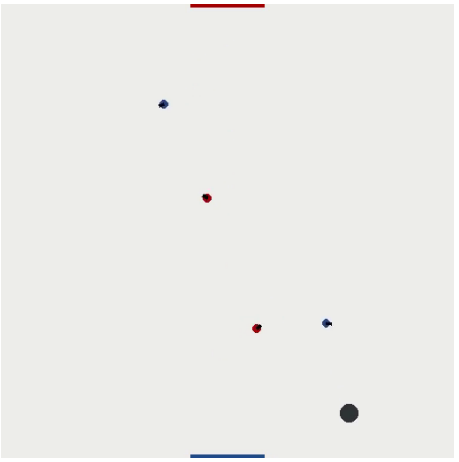

Figure 2

Our controller worked by choosing the agent closest to the puck and designating it as offense, while the other agent was assigned to defense. The offense agent angled itself to hit the puck into the opposing team's goal and used a boost to score every time it was within a certain distance from the goal. We then realized that assigning one agent to offense and the other to defense wasn't the most effective strategy. We began experimenting with more dynamic strategies, such as having both agents focus on offense

and defense simultaneously and switching roles on an as needed basis.

To facilitate this new approach, we added features to our CNN that allowed it to track the movements of both the puck and the opposing team's agents in real-time. We then trained our agents, allowing them to learn optimal actions through trial and error without the need for hand-crafted features or explicit instructions. Despite struggling with occlusions and fast movements and the requirement of significant data and computational resources for effective training, we refined and improved our agents' strategies through experimentation and analysis, ultimately developing an effective system that could win matches.

Model

Upon analyzing our gameplay and similar games, we came to the conclusion that we should have one agent concentrating on maneuvering the puck into the opponent's goal, while the other agent focused on identifying and relaying the puck's exact location to the scoring agent. We decided this because of the belief that a solo agent equipped with flawless knowledge would outperform two agents inundated with misinformation.

Our team created a detector model for object detection by combining a convolutional neural network (CNN) with an upsampling network. The model, named *Detector*, is a module consisting of two nested classes, *BlockConv* and *BlockUpConv*. *BlockConv* defines a 3-layer residual convolutional block, and *BlockUpConv* defines a 3-layer transposed convolutional block. To extract local maxima (peaks) from a 2D heatmap generated by the model, we employed the *extract_peak* function.

Train

The code written to train our created model involves testing various combinations of learning rates, optimizers, loss functions, and input sizes. The ultimate goal was to identify the combination of hyperparameters that would produce the lowest validation loss. We tested out different loss functions and concluded that a combination between *BCEWithLogitsLoss()*, *FocalLoss()*, and *MSELoss()* gave us the best results for our project. The camera inputs were processed through these networks, enabling our agents to accurately detect the puck's location and respond accordingly with agility, granting us an edge over our opponents and facilitating smooth navigation of the ice-hockey rink.

Methods

As a team, we have thoroughly explored various methods for making use of an image-based agent. One approach we have considered is

utilizing deep reinforcement learning techniques to train the agent directly from raw image data. This method allows the agent to learn optimal actions through trial and error without the need for hand-crafted features or explicit instructions.

Another method we have considered is hand-crafted feature extraction followed by traditional machine learning techniques. In this approach, we would utilize image processing techniques such as edge detection, color segmentation, and template matching to extract meaningful features from the images. We would then train our machine learning model to predict the optimal actions to take based on these extracted features.

Although each of these methods has its own set of advantages and disadvantages, we also need to consider potential issues. Object detection and tracking methods may struggle with objects hidden from view and fast movements, which can lead to inaccurate predictions.

Our team's final approach to designing the player involved utilizing image analysis to identify the presence of a puck within the game environment. Once both agents detect the puck, they collaborate to gain possession and score. If they are close to the goal, they increase their speed and use calculated angles to direct the puck towards the goal. In situations where only one agent detects the puck, it will pursue it while the other agent defends the goal. If neither agent detects the puck, they move closer to their goal until the puck is identified. Additionally, the player is designed to rescue and redirect any stuck agents, enabling them to continue searching for the puck as if they had already detected it. These strategies highlight the player's ability to make informed decisions in complex game scenarios.

Results

Our final method ended up being a mix of using single offense/single goalie and double offense which led to our model being very complex. The complexity of our model could be a reason for our low score since it would require more time. We could have avoided this by sticking to one of the two variations, although defense is important we believe that going the route of only having a double offense setup could have given us a better score.

Below show the images that were generated from our image-based agent during the ice-hockey gameplay:

Figure 3

The image above, *Figure 3*, shows the player driving based on visual cues. The puck can be seen in the below image, *Figure 4*, where a mask is applied for better communication visually. The agent is able to fully see the puck and automatically make its own decisions based on that.

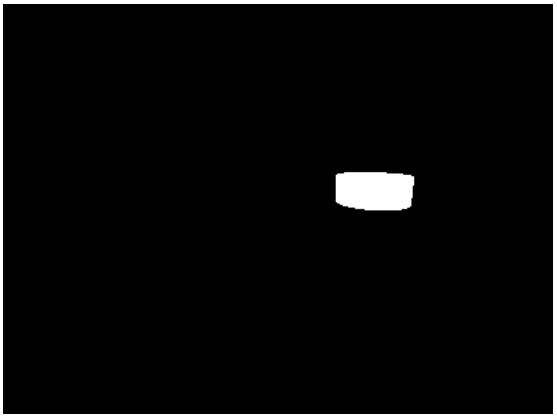

Figure 4

With more time we would be able to optimize our model further and be able to try more variations that could bring better results. We also tried a few different loss functions but the loss that worked best was a combination of *BCEWithLogitsLoss()*, *FocalLoss()*, and *MSELoss()* as shown below:

```
if gamma == 0:
                    loss_centers =
torch.nn.BCEWithLogitsLoss().to(device
)
                else:
                    loss_centers =
FocalLoss(gamma=gamma,
reduce=True).to(device)
                loss_sizes =
torch.nn.MSELoss(reduction='none').to(
device)
```

Having more time to tune some of the hyperparameters and experiment with different karts and the kart features would have been beneficial for us as well.

Conclusion

In summary, we were tasked with creating a strong ice-hockey player for SuperTuxKart, and we decided to use an image-based agent for several reasons. Image-based agents are better suited for processing visual information and can learn to identify key features on their own. To achieve this, we used a deep neural network and optimized our code to ensure efficient processing of images.

Our primary goal was to design a player that could effectively gain possession of the puck and score. We used image analysis to improve the player's speed and ability to calculate angles to direct the puck towards the goal when in its vicinity. If the puck was not detected, the player would move towards the goal until it was

found. We also equipped the player with the ability to rescue and redirect any stuck agents to keep searching for the puck.

Throughout our work, we provided detailed explanations of the steps we took to arrive at our decision. We also provided an overview of SuperTuxKart and how convolutional neural networks are used to train agents to recognize and respond to visual cues within the game.

References

Ezbutd. (n.d.). *EZBUTD/Supertux-ai: Writeup*. GitHub. Retrieved April 27, 2023, from https://github.com/EZBUTD/SuperTux-AI

Deep learning project of a supertuxkart ice-hockey player. (n.d.). Retrieved April 28, 2023, from https://www.researchgate.net/publication/357897741_Deep_Learning_Project_of_a_SuperTuxKart_ice-hockey_player

Discover. SuperTuxKart. (n.d.). Retrieved April 27, 2023, from https://supertuxkart.net/Discover

Mandal, M. (2023, April 27). *Introduction to convolutional neural networks (CNN)*. Analytics Vidhya. Retrieved April 27, 2023, from https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

Karts/Controller. SuperTuxKart. (n.d.). Retrieved April 27, 2023, from https://doxygen.supertuxkart.net/group__controller.html