Final Project Report

Group 105: William Heidel, Imran I. Itani, Elena Quartararo, Hunter Sherman

Master's of Data Science / Artificial Intelligence, University of Texas at Austin

DSC 394D: Deep Learning

April 27, 2024

Introduction

Overview of the Project

The primary objective of this project is to develop an autonomous agent capable of playing ice-hockey within the open-source racing game SuperTuxKart. The challenge involves programming an AI to compete effectively in a 2 vs 2 ice-hockey tournament, aiming to score as many goals as possible against both AI and human-coded agents. This task bridges the realms of deep learning and game AI, showcasing the potential for sophisticated neural network applications in dynamic, real-time gaming environments.

Motivation

The decision to focus on an image-based agent was driven by the sophisticated visual processing capabilities typical of advanced autonomous systems, such as those found in self-driving vehicles and robotic navigation (Kanagaraj et al., 2021). Our choice was further influenced by the challenges presented by the fast-paced and visually rich environment of SuperTuxKart, which provides a fertile ground for demonstrating the practical impacts of such image-based technologies. This approach aligns with current trends in deep learning research, where integrating perceptual data into decision-making processes is crucial for developing robust AI systems. Looking specifically at AI capabilities within gaming, this is a field that continues to benefit significantly from deep learning advancements (Khanna et al., 2019). Thus, this project not only seeks to enhance gameplay through AI but also aims to demonstrate how computer vision can contribute to the broader field of real-time AI applications.

This project's relevance extends into ongoing research into the application of convolutional neural networks (CNNs) in environments where real-time data interpretation and reaction are necessary. By deploying a deep learning model capable of processing complex visual inputs and making autonomous decisions, this work contributes to the understanding of AI's potential in both simulated and real-world scenarios.

Methods

1. Data Collection

1.1 Overview

Data collection was a critical initial step to ensure that our models were trained on relevant, high-quality inputs. We collected data by utilizing the state-based agents provided in the SuperTuxKart environment, where the matchups between agents were chosen at random for each dataset. This approach allowed us to gather a diverse set of images representing various game states, which are essential for training our models to recognize and react to different in-game scenarios effectively.

1.2 Image Data Collection

We systematically captured all images from the agents involved in each game. PySTK, the game's toolkit, offers both detailed image data and coded pixel information at every time step, which we used to generate labels. The image data is encoded as a 3-dimensional pytorch tensor, the input provided to our image-based agent during gameplay. The coded pixel information is encoded as a 2-dimensional tensor, where each pixel's value (when decoded) indicates a specific in-game element (1 is a kart, 2 is the background, 8 is the puck, etc.). This pixel decoding was crucial for our target task of puck detection, as it enabled us to identify the puck in each image and label its location.

1.3 Puck Localization

Using the coded pixel data provided by PySTK, we developed a method to precisely locate the puck within each image. The puck's position was determined by creating bounding boxes – defined by the coordinates [min_x, min_y, max_x+1, max_y+1] – around the pixel positions that were labeled as puck. These labels were then used to either binarily label an image “puck” or “no puck,” or to generate a heat map of the puck location, depending on the task of interest. In either case, these labels were then used as ground truths during training.

1.4 Data Storage and Organization

For each captured image, two files were stored:

- The original image in JPEG format.
- A NumPy NPZ file containing the bounding box coordinates of the puck.

Additionally, a single CSV file was created for each dataset. Each row in this file contained 2 columns: the name of the JPEG file and its corresponding label indicating the presence or absence of the puck within the image.

To mitigate overfitting and test the generalizability of our models, we divided the collected data into distinct sets:

- CNN Training Set: Composed of approximately 15,000 images, used to train the CNN model. These images are of higher resolution (400x300) to facilitate the detection of the puck at greater distances. Each image is labeled with a binary 0/1 to indicate it contains a puck (ensuring the puck size is at least 6 pixels to qualify as detectable) or not. The balance between “puck” and “no puck” images was almost exactly 50%.

- CNN Validation Set: Similar in structure to the training set, containing around 15,000 images, used to tune the hyperparameters of the CNN and prevent the model from overfitting.
- FCN Training and Validation Sets: For the FCN model, the images were resized to 128x96 to expedite training times. This dataset includes about 10,000 training images and 5,000 validation images, exclusively containing scenes where the puck is visible and meets the minimum size requirement.
- CNN and FCN Test Sets: The same methodology described above was used to generate an additional set of data for each model, so that the models could be evaluated without bias.

Generated Image | Colored Plot of Pixel-Encoded Image

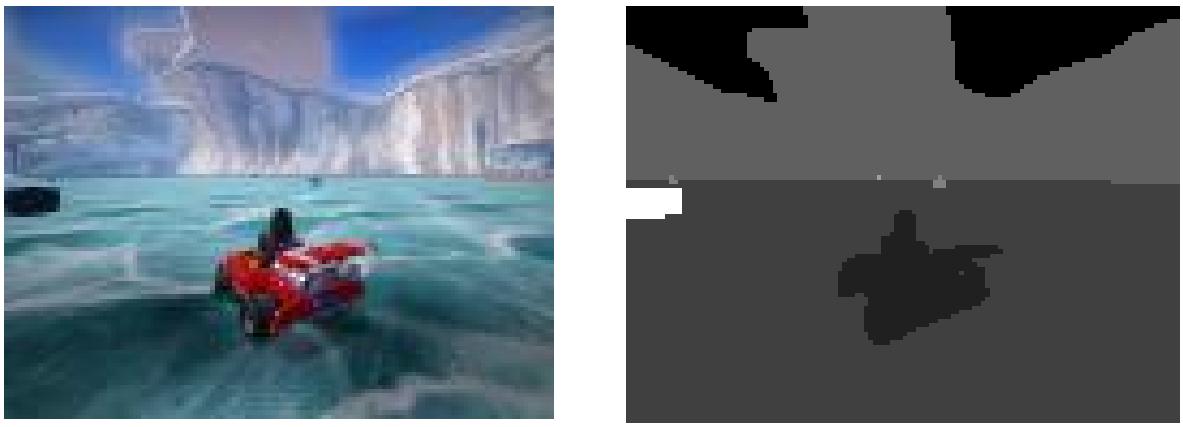

Figure 1: An example of an image and its colored-coded object labels generated for training.

2. Controller

2.1 Overview

The controller is a critical component of our image-based agent, responsible for interpreting the outputs from the neural network models and translating them into actionable decisions within the game environment. Our controller strategy involves various stages, each tailored to different game states and conditions.

2.2 Initialization Phase

During the initial 32 frames, the controller drives the kart straight to the center of the rink for the initial face-off. This phase is critical for positioning the kart in anticipation of the puck's movements, and competing for the puck early on to prevent the opponent from an easy, immediate goal. Any pickups available are fired in this phase, as they may give us an advantage in beating the opponent to the puck.

2.3 Puck Detection and Tracking

The controller continuously monitors our players' viewpoints via the game provided images to detect and respond to puck presence and location. Using a convolutional neural network (CNN), the controller assesses whether the puck is visible in the captured image. If the puck is detected, the fully convolutional network (FCN) determines the puck's precise center point. This point is adjusted slightly to produce the optimal aim point for steering with the objective of hitting the puck towards the opponent's goal or away from our own.

- Puck Steering: The controller calculates the steering angle based on the relative position of the puck, utilizing trigonometric functions to determine the optimal path to intercept or follow the puck.
- Goal-Oriented Movement: Once near the puck, the controller adjusts its strategy to push the puck towards the opposing goal, factoring in the orientation and position of both the kart and the puck.

Given the frequency of false negatives in our models, puck detection logic includes memory of previous detections, allowing us to continue pivoting the kart towards previous known puck locations.

2.4 Handling Game Resets

The controller detects game resets, signified by significant positional jumps between time steps. This triggers a strategy reset to reinitiate the initial driving phase and clear historical data to adapt to the new round conditions.

2.5 Unsticking Mechanisms

During gameplay, karts may become stuck against walls or within goals. The controller identifies these scenarios by monitoring proximity to the rink's edges and the kart's orientation. If a collision course is detected, the controller engages reverse maneuvers for several time steps to correct the kart's path and resume normal operation.

2.6 Search and Recovery

If the puck is not detected and there's no recent confidence in its sighting, the controller initiates a search pattern:

- Circling: The karts circle the rink to relocate the puck, using a calculated steering adjustment to maintain a systematic search pattern. The center point of the two kart's circling pattern is offset to cover more of the rink.
- Recovery Actions: If the puck was recently observed but is no longer visible, the controller may execute a brief reverse to reacquire the puck, based on the last known trajectory and confidence scores from the FCN.

2.7 Search with Known Position

Our players are able to communicate a rough puck position via a shared state object. Any time the puck is seen, its approximate location on the rink (signified by compass coordinates) is updated. This provides a direction for our karts to travel when the puck is not detected on screen, with better results than circling alone.

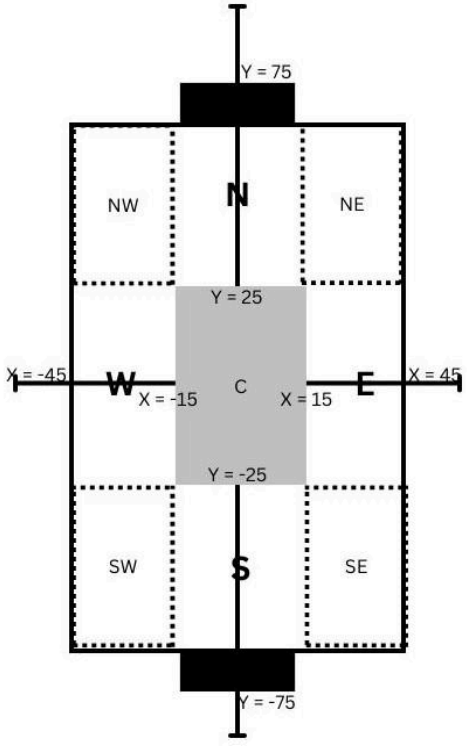

Figure 2: Sextant locations used to communicate potential puck locations.

3. Models and Training

3.1 Overview

This section describes the neural network models used in our project: a Convolutional Neural Network (CNN) and a Fully Convolutional Network (FCN). Both models play critical roles in processing image data to detect the puck's position and guide the autonomous agent's in-game actions accordingly.

3.2 CNN Model

3.2.1 Functionality

The CNN model is designed to identify the presence or absence of the puck within the game environment. It operates on the principle of feature extraction from images, which are then used to predict whether or not the puck is in the image.

3.2.2 Architecture

The CNN architecture is built using multiple layers to enhance its feature extraction capabilities:

- Initial Layer: A 7x7 convolutional layer with stride 2 and padding 3, followed by ReLU activation and a max pooling layer. This layer is designed to capture the initial set of features from the input images while reducing their dimensionality.
- Convolutional Blocks: Each block consists of a 3x3 convolution, batch normalization, ReLU activation, and a dropout of 25%. These blocks are stacked with increasing filters (16, 32, 64, 128), allowing the network to learn more complex features at each level.

- Classification Layer: The final part of the CNN is a linear layer that classifies whether the puck is present in the image. This layer uses the features extracted by the convolutional blocks to assign the image to either class 0 (no puck) or class 1 (puck).

The network uses batch normalization to improve training speed and stability, and dropout to prevent overfitting. To further improve performance, the input was normalized before being passed through the network. The mean and standard deviation of the training dataset was calculated outside of the model and coded into the CNN. See Figure A1 in Appendix A for a visualization of the model.

3.2.3 Training

- Loss Function: Cross Entropy Loss for binary classification.
- Optimizer: Adam, with a learning rate of 0.001 and weight decay of 0.0001. Stochastic Gradient Descent (SGD) was also tested, as well as different values for the learning rate and weight decay. The values mentioned above resulted in the highest accuracy on the validation dataset.
- Data Augmentation: To enhance the model's ability to generalize, we employed random horizontal flips during training. However, we avoided color jitter as the game's consistent color scheme does not require such augmentation, and adding jittering seemed to actually hurt model performance.
- Training Procedure: The model was trained over 20 epochs with a batch size of 128. Learning rate adjustments were made based on the plateauing of accuracy metrics,

specifically reducing the learning rate if there was no improvement in accuracy for 5 consecutive epochs.

- Performance Metrics: The model's performance was evaluated based on accuracy, IOU, precision, recall, specificity, and F1 score. Most notably, the model achieved approximately 91% accuracy on the validation set.

3.3 FCN Model

3.3.1 Functionality

The Fully Convolutional Network (FCN) is designed for precise localization tasks within our project, essentially acting as a classifier for every pixel. This model excels in generating spatially dense predictions from the input images, crucial for real-time decision-making in gameplay. Within the game, the FCN is used to locate the center of the puck if the CNN determines the puck is in the player image.

3.3.2 Architecture

The architecture of the FCN model includes several key components designed to enhance its ability to accurately interpret complex spatial relationships within images. Its design is heavily inspired by U-Net architecture, which known for its efficiency in performing semantic segmentation tasks accurately, even on smaller datasets (Ronneberger et al., 2015). The most notable differences in our FCN's architecture vs. U-Net's are the smaller number of layers (our maximum channel size is 256 instead of 1024) and the addition of residual connections at each downsampling block. See Figure A2 in Appendix A for a visualization of the model:

- Initial Layer: The network starts with a normalization of inputs (similar to the CNN) followed by a 3x3 convolutional layer with 32 output channels, setting the stage for deeper feature extraction.
- Downsampling Blocks: Sequential blocks that double the number of filters at each stage (64, 128, 256). Each block consists of:
    - A MaxPooling layer to reduce spatial dimensions and increase the receptive field.
    - Residual connections that help preserve information across layers and improve gradient flow during training (He et al., 2015).
- Upsampling Blocks: These blocks halve the number of filters (128, 64, 32) and include:
    - ConvTranspose layers for spatial expansion.
    - Skip connections from corresponding downsampling blocks to reintegrate earlier spatial features, crucial for precise localization.
- Final Layer: A 1x1 convolution that outputs the logits, representing the puck's predicted locations as a heatmap.

3.3.3 Training

- Loss Function: Focal loss was chosen over Binary Cross Entropy loss due to its effectiveness in handling class imbalance between foreground and background by focusing more on hard-to-classify examples (Lin et al., 2018). Parameters for focal loss included alpha (0.25) and gamma (2).
- Optimizer: Adam with a learning rate of 0.001 and weight decay of 0.0001. This setup is optimized for fast convergence while preventing overfitting through regularization.

- Early Stopping: Implemented to halt training if the validation loss does not improve by more than a threshold of 0.0001 over 10 epochs, ensuring that the model does not overfit to the training data.
- Batch Size and Epochs: The model was trained using a batch size of 64 for 50 epochs, balancing computational efficiency with adequate gradient estimation per update.
- Data Handling: Normalization of inputs without the use of dropout or color jitter, focusing on consistent training conditions reflective of the game's environment.
- Performance Metrics: The primary metric for training efficacy was the reduction in loss, which significantly decreased to about 2.5e-4 by the final epoch, indicating effective learning and model convergence.

Results

4.1 Performance of Models

4.1.1 CNN

To measure the performance of the CNN, our main metrics were Accuracy and Recall. Accuracy is the most intuitive measure, as it is simply the proportion of images labeled correctly by our model. During training, we plotted both the training and validation accuracy at each step:

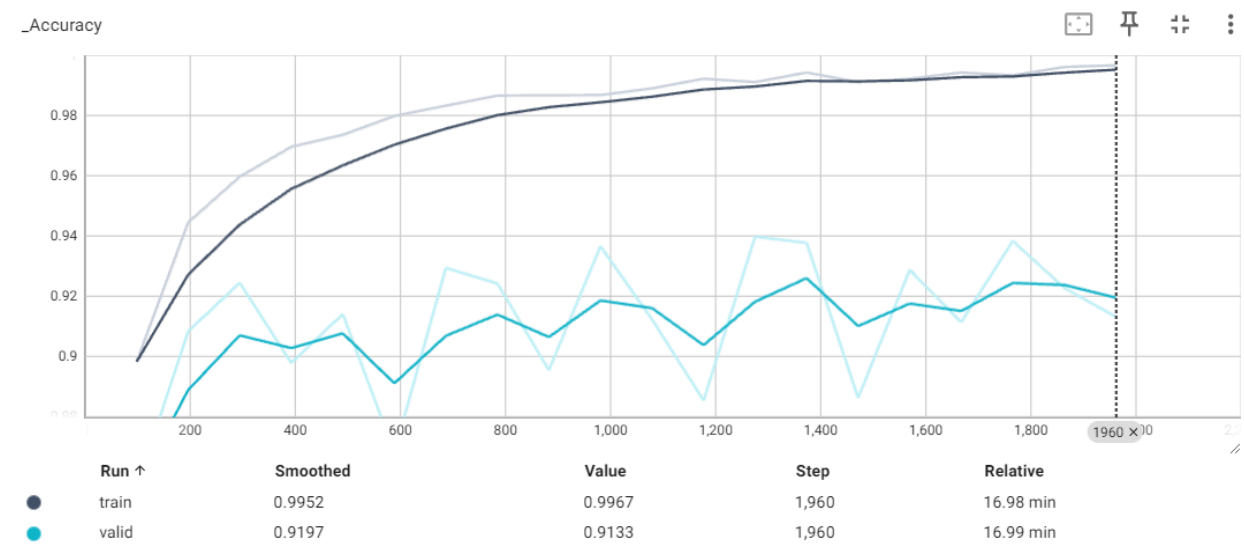

Figure 3: A plot of calculated accuracy per step (training set) and epoch (validation set).

While assessing agent performance, it became apparent that false negatives were the most important problem with the model. We therefore also evaluated our model on recall, the ratio of all positive examples in the dataset being labeled correctly:

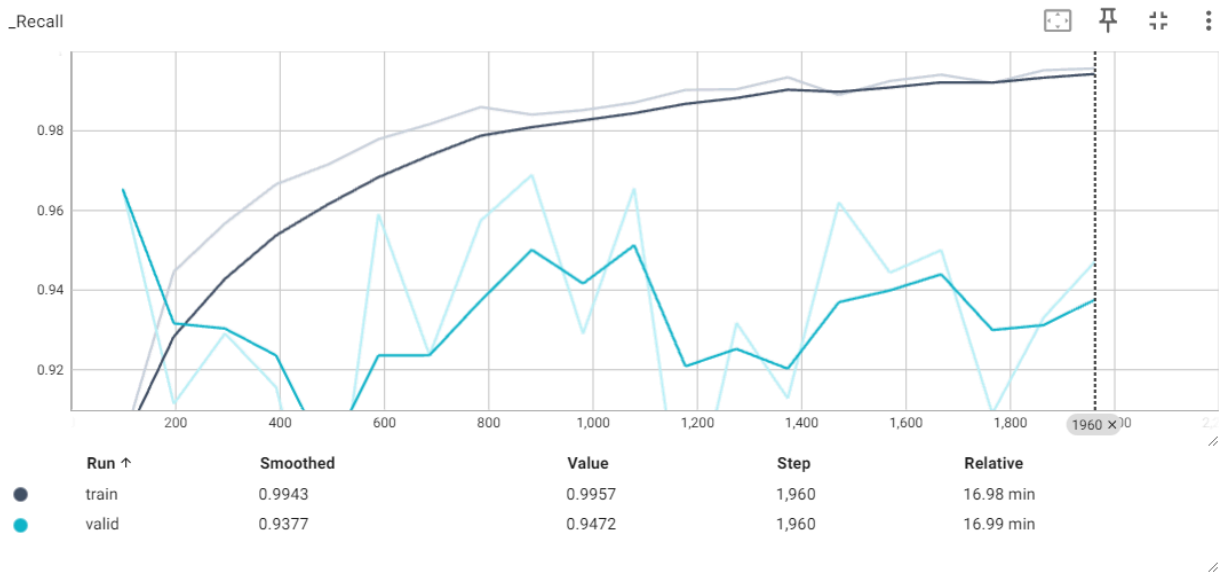

Figure 4: A plot of calculated recall per step (training set) and epoch (validation set).

Our best performing model was consistently over 90% for both accuracy and recall on the validation set. Other measures of model performance were assessed, such as IOU, precision, F1 score, and specificity, but accuracy and recall were deemed the most important metrics to use as feedback.

After the best CNN model was trained, we evaluated these metrics on a test set of 5,000 images generated from matchups between AI agents that had not occurred in the training or validation dataset generation. We then plotted the confusion matrix of the results and calculated the metrics described above:

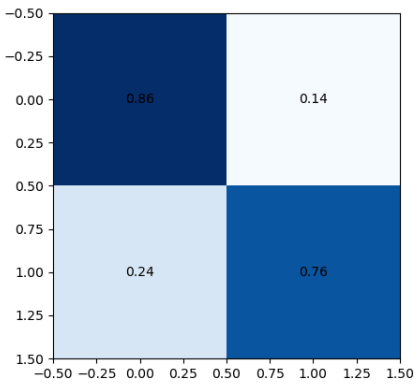

Figure 5: Confusion Matrix on Test Set

Metric	Score
Accuracy	0.801
Recall	0.758
Precision	0.883
F1 Score	0.816
IOU	0.666
Specificity	0.860

Table 1: Performance Metrics on Test Set

Overall, these results were satisfactory, but not stellar. A model performing upwards of 0.9 in both Accuracy and Recall would have been ideal, but the current model performed reasonably well in practice.

4.1.2 FCN

To measure the performance of the FCN, we plotted both the loss over time and image comparisons of the model predictions vs. ground truth labels. Initially, Binary Cross Entropy Loss was used, but neither the loss nor prediction comparisons were particularly compelling. After switching to focal loss, both the overall loss values and predictions showed significant improvement:

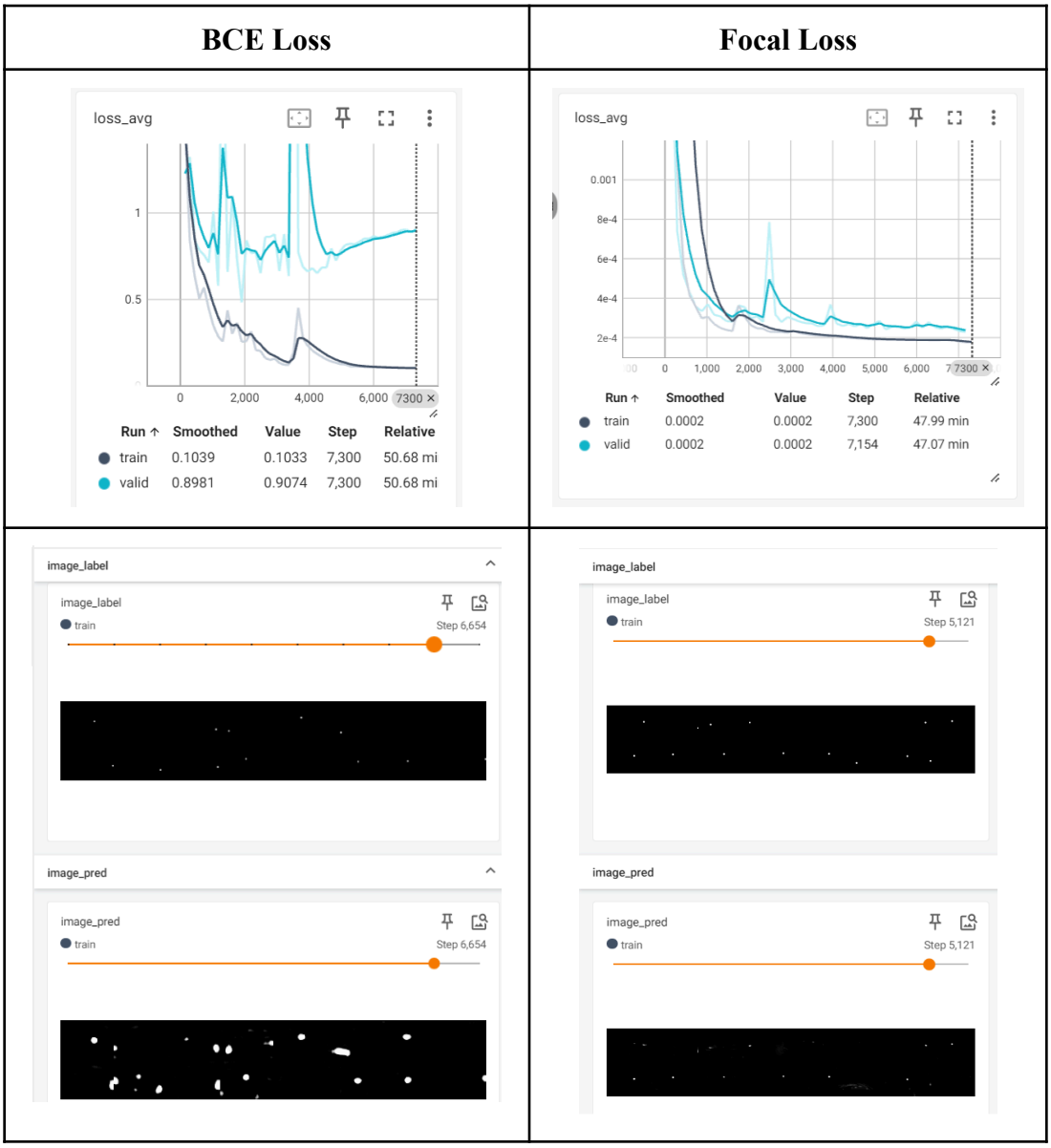

Figure 6: Comparison of Loss and Predictions using BCE vs. Focal Loss

Focal loss on the validation set was extremely low by epoch 50, with a value of about 0.0002. A visual comparison of the images between the ground truth labels and the predicted values were also very encouraging. For these reasons, we determined our FCN to be satisfactory.

When using the Image Agent, the heatmap predictions from this FCN are then fed into a detector function which returns the most likely centerpoint of the object we are trying to detect: the ice hockey puck. This point becomes the aim point that determines how our kart drives. As a final check, we calculated the average euclidean distance (in pixels) between the ground truth aim point and predicted aim point on a test set of 5,000 images. The average distance between these two points was just under 15 pixels, which is highly accurate in a 128x96 pixel image. This increased our confidence in our puck detector as we built out the controller.

4.2 In-Game Performance

4.2.1 Agent Scoring

The main goal was for our agent to score at least one goal in each game. Unfortunately we fell well short of this, only scoring an average of about 0.32 goals per game. Analyzing situations where our agent was and was not successful, we found that our agent particularly struggled against agents that were able to play defense, as we had no strategy to avoid them or thwart their attempts to block us. Many of our procedures worked very well in a vacuum. For example, we very rarely got stuck in one spot, and we were often able to find the puck after briefly losing it. However, once in the mode of circling the rink to look for the puck, our agent struggled to identify the puck with enough distance to appropriately correct steering. That is, it was often the case that once our agent recognized the puck, we were going past too fast to steer towards it, and would end up losing it again.

4.2.2 Detector Accuracy

Because our puck detector plays the main role in determining the Image Agent's course of action, we used image printouts of each frame for our agent to see whether or not it was detecting our puck accurately (see Figure 7). The detector was very reliable at detecting both the presence of the puck and its corresponding aim point when the puck was close, yet it struggled with detections where the puck was farther away. This is likely because in those cases the puck's small size makes it much more difficult for our models to detect.

Image with Puck Detection

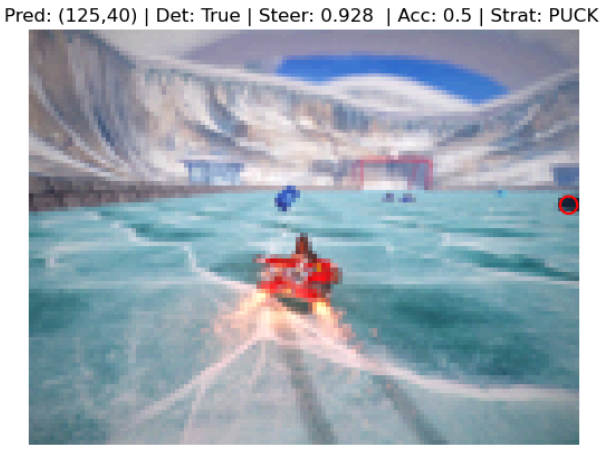

Image with no Puck Detection

Figure 7: Examples of image printouts when the Agent detects the puck (left) and when it does not (right).

Conclusions

Summary of Outcomes

Our project's main objective was to design an autonomous agent capable of effectively playing ice-hockey within the SuperTuxKart environment. While our approach was innovative and grounded in deep learning principles, the actual game results – specifically the number of goals scored – were below our expectations. This shortfall primarily stemmed from limitations in our puck detection system, which often lost track of the puck due to false positives and negatives. These detection errors frequently triggered the agent's search and recovery mode, diverting it from active play to search for the puck, thereby diminishing its effectiveness in the game. This was particularly noticeable at distance.

Addressing Failure and Alternative Methods

Reflection on Tactics and Strategic Gameplay

One significant insight from our project's outcome is the importance of tactical gameplay in addition to technical prowess. In particular, we are interested in how performance could be improved by giving the two players on our team differing strategies. Inspired by successful strategies in similar games, such as Rocket League, we propose a "rotation" tactic for future implementations (Thanovic, 2021). By alternating the roles of our karts between offense and defense, we can cover more area on the field, enhance our defensive stability, and increase offensive opportunities. Such strategic positioning could also mitigate the impact of puck detection errors by maintaining continual visual contact with the puck, thus reducing the frequency and duration of search and recovery modes.

Technical Improvements and Future Directions

The primary technical challenge was the integration of CNN and FCN models, which did not perform optimally in tandem. The CNN was tasked with binary puck detection, while the FCN localized the puck. The misalignment between these tasks often resulted in conflicting signals about the puck's presence and location, leading to the aforementioned false detections. There are a couple possible enhancements we would explore if we were to continue this project.

First, we could enhance our training regime of the CNN to improve performance. The disparity between accuracy on the validation set (~91%) versus the test set (~80%) indicates that there may be “difficult” examples that were underrepresented in our initial training/validation datasets. Based on observations of the model in game scenarios, such “difficult” examples may include when the puck is far away, when it is off to the side instead of straight ahead, or when other objects that resemble the puck are in the frame. Adjusting the training process to focus more on such scenarios could better prepare the model for the complexities of the game environment.

Second, we could attempt to develop a more robust unified detection framework. Initially, we developed our two-step approach to finding the puck (feeding the image through the CNN first and then the FCN) due to concerns about false positives from the FCN. Because the FCN was only trained on images that contain the puck, it tended to be confident about a center point whether or not the puck actually appeared in the image. However, as mentioned above, the CNN did not do as good a job at rooting out these false positives as initially hoped, and also introduced a lot more false negatives. Additionally of note, the FCN was trained on compressed images, as we were constrained by compute power, which likely causes a loss of information. We posit that

by training the FCN on more varied data – specifically including images without the puck – and training on full size images, we could potentially produce a model that streamlines the puck detection pipeline and ultimately reduces error rates.

Third, for the sake of time and accuracy, our models were trained on a single class, the puck. While training on goals or team karts may not have been useful given the static or provided coordinates for these, training with opponent or pickup classes could have provided us with opportunities, given enough time, to experiment with additional controller strategies. Such strategies could include more aggressive defense or attack against opponents, or having one of the karts play permanent defense.

Lastly, we could expand the model's capabilities to take advantage of more than just the image information that is available at each time step. We could implement more advanced object tracking algorithms that could maintain the identity and location of the puck even when direct detection fails temporarily. We could also incorporate deep reinforcement learning to allow the agent to learn optimal strategies through trial and error, adapting to various gameplay styles and opponent behavior dynamically.

Final Remarks

In conclusion, while our project did not meet all its intended goals, it has provided valuable insights into the integration of deep learning models within interactive game environments and highlighted the importance of combining strong technical solutions with strategic gameplay. The

lessons learned from this project will guide future enhancements for more robust and strategic AI players in simulated environments. We hope that the proposed adjustments to both strategy and technology would not only improve the agent's performance in subsequent iterations but also contribute to broader research in AI-driven gaming and simulation.

References

K. He, X. Zhang, S. Ren, & J. Sun. Deep Residual Learning for Image Recognition. (2015). https://arxiv.org/abs/1512.03385.

Kanagaraj, N., Hicks, D., Goyal, A. et al. Deep learning using computer vision in self driving cars for lane and traffic sign detection. Int J Syst Assur Eng Manag 12, 1011–1025 (2021). https://doi.org/10.1007/s13198-021-01127-6.

A. Khanna, B. A. Manoj, Y. T., G. M. Parihar, R. Maurya and R. B., DeepStreak: Automating Car Racing Games for Self Driving using Artificial Intelligence. *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2019, pp. 420-425, doi: 10.1109/I-SMAC47947.2019.9032527.

T. Lin, P. Goyal, R. Girshick, K. He, & P. Dollár. Focal Loss for Dense Object Detection. (2018). https://arxiv.org/abs/1708.02002

Ramsundar, B., & Zadeh, R. B. (2018). TensorFlow for deep learning : from linear regression to reinforcement learning (First edition.). O'Reilly Media.

O. Ronneberger, P. Fischer, & T. Brox. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. https://arxiv.org/abs/1505.04597v1

Thanovic. “The Ultimate Rotations & Positioning Guide for Rocket League.” YouTube, 9 Apr. 2021, www.youtube.com/watch?v=xiHHbvhRNBU.

Appendix A

CNN Structure

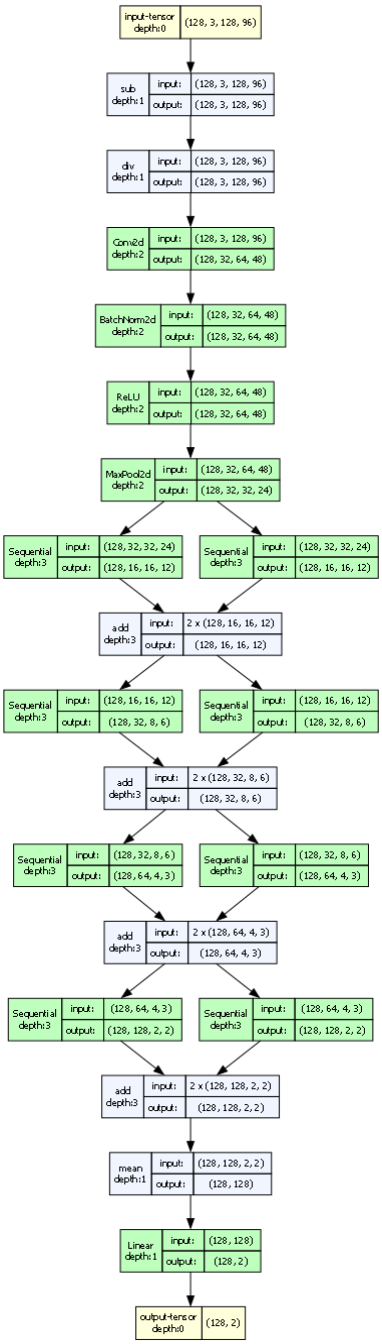

Figure A1: Visualization of CNN Structure

FCN Structure

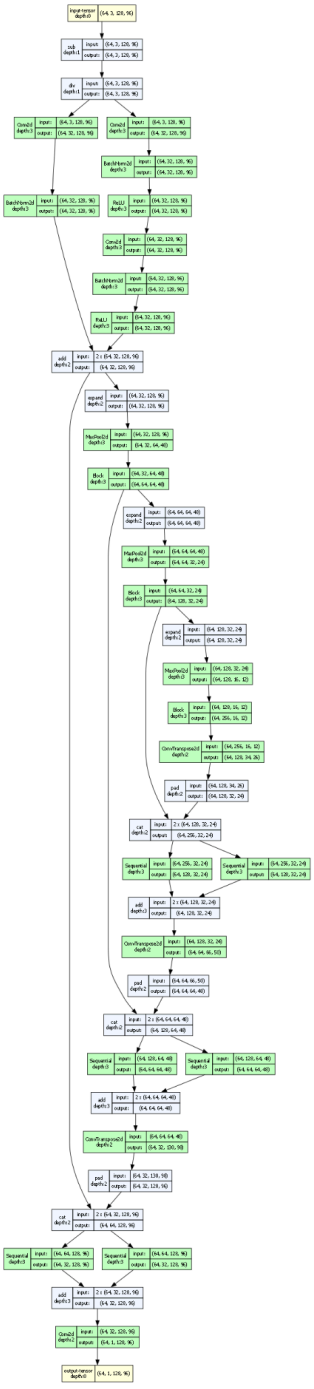

Figure A2: Visualization of FCN Structure