Deep Learning - Final Project Report

Andreas Winklmayer, Xi Chen, Khalid Moutaouakil and Minh Duy Vu

Abstract— **SuperTuxKart is a 3D open-source arcade game with a variety of characters, tracks and modes to play. In this project, we would like to develop an autopilot model for a modified version from the original game that operates through a Python interface. Our approach utilized computer vision strategy to analyze images from the games to give insight for the bot to generate action outputs. Our model outperforms several state-based agents currently available.**

I. Introduction

This project entails developing a SuperTuxKart ice-hockey player, with two distinct approaches available, the computer vision approach (image-based agent) and the reinforcement learning approach (state-based agent). These two agent types diverge in their input mechanisms and performance criteria. Nevertheless, regardless of the chosen approach, our target is to maximize the number of goals scored and secure the victory in the matches. The game evaluation will be based on 2 vs 2 tournaments, competing against AI established agents.

On the one hand, the state-based agent is usually constructed by a single deep neural network, eliminating the need for hand-designed components and ensuring a streamlined, unified control mechanism. In the context of this game, the state-based agent has access to the world state of the game, including but not limited to the information about players' karts, the opponents' karts, and the puck. Equipped with optimization techniques, the state-based agent maximizes computational resources, achieving peak performance while maintaining optimal gameplay responsiveness. There are many successful examples for the use of this type of agent in AI research. [1] [2] However, the drawbacks of this approach include the requirement of access to the game world state, as well as the availability of simulated or human players as the trainer.

On the other hand, image-based agent learns patterns and features from the game environment via the players' interface, leading to a model that is able to adapt easily. If enough image data is fed to the model, the prediction accuracy can be very high, which increases the perception of the autobot about the game environment and enables its game actions to be executed properly. In fact, image-based agents have been employed to play many popular games. The first example is StarCraft, which is a real time strategy game containing hundreds of characters (units and behaviors) and requiring enormous amount of micro- and macro-management of the game units. Despite that challenge, Deepmind's Alphastar successfully utilized Convolutional Neural Network (CNN) to process game image inputs and was able to outperform actual players to prove its capability to strategize and act in real time. [3] The second example is Super Mario Bros, for which, Pluribus was co-developed by researchers from Carnegie Mellon University and Facebook AI to solve the game. Pluribus was implemented using neural network architecture and CNN specifically to predict the game state and act based on it. [4] It managed to defeat even human professional gamers.

With the strong advantage of universality and without any special access requirement to solve the game, we considered and chose the computer vision approach. Although this agent does not receive real-time updates regarding the puck's state or the actions of opponents during gameplay, it receives visual inputs in the form of images depicting each player's in-game view (karts' camera), enabling it to deduce other important information such as the location of the puck and the positions of the opponent. We would expect to encounter solutions heavily reliant on visual processing by following this approach.

We can relate our image-based approach to self-driving cars. Self-driving cars are able to automatically steer the wheel, apply acceleration and brake appropriately based on a model. U-Net and CNN architectures are well-known for these perception tasks in autonomous driving systems. [5] CNNs can analyze input images from cameras, [6] LiDAR point clouds, [7] or radar data to detect and classify nearby objects such as vehicles, pedestrians, cyclists, traffic signs, and traffic lights. [8] By learning from large amounts of labeled data, CNNs can accurately recognize patterns and make predictions about the environment, enabling the vehicle to navigate safely and efficiently. Besides, U-Net is a popular architecture for semantic segmentation tasks, where the goal is to assign each pixel in an input image to a specific class label. In the context of self-driving cars, U-Net can be used to segment camera images to identify lanes, vehicles, pedestrians, and other objects in the environment. [9] This segmentation information is crucial for understanding the surroundings and making informed decisions about navigation and control. Some self-driving systems explore the possibility of end-to-end learning, [10] where a single neural network is trained to directly map raw sensor inputs (e.g. camera images) to driving actions (e.g. steering angle, throttle, brake). While this approach simplifies the system architecture and can potentially improve performance, it also requires large amounts of training data and careful tuning to ensure reliability.

II. Methodology

A. Data generation and preprocessing

With the central target of taking image inputs to guide the kart autopilot system, there is multiple data that can be learned, including but not limited to puck information, opponent information and environment information. Due to the

complexity of the game and the simple scope of this project, we mainly focus on puck information, including the puck center (coordinate values), puck visibility (boolean values) and puck distance from the player's camera (real values). In other words, from any given image observed by the player's point of view camera, the autobot can infer whether the puck is seen, and if yes, where its location is and how far it is from the kart.

We started the project by gathering images, depth maps, semantic segmentation arrays, and game states from the game. These data were collected by simulated games between teams of six available state-based agents. The images were collected with the ground truth labels exported from the game state. Since there are more than 10 different types of karts with different appearances and behaviors, our simulated games for data collection also used various kart types, e.g. "Tux", "Sara the racer", etc. This allowed us to gather images from a variety of game contexts. Overall, more than 45,000 images were collected in total for our train and test datasets. We then tried two different approaches, namely, an image analysis approach and a game state-based approach, to label the images concerning the binary classification "puck" vs "no puck" in the image, the (x, y) coordinates of the puck inside the images, and the distance of the puck from the camera.

In the former approach, we applied several image processing algorithms. We used the semantic segmentation arrays provided by the game API to determine the number of pixels that belonged to the puck in each image. If it surpassed a set threshold, we classified the images as "containing puck" and vice versa. Once the puck was identified in the image, the mean center (x_c, y_c) of the puck pixels was then used as the (x, y) coordinates for the puck. As for the puck-camera distance, we first checked whether the center pixel actually belonged to the "puck" class, as occlusions of the puck by a kart could make it possible that this was not the case. If there was no occlusion, we used the value from the depth map at (x_c, y_c) as the ground truth depth of the puck. The depth map was also gathered from the game API. In cases of occlusions, we first found the nearest pixel around (x_c, y_c) that belonged to the puck class in the semantic segmentation array. Afterwards, we used the value from the same respective location of the depth map as a suitable approximation of the puck-camera distance.

In the latter approach, we loaded the data from the internal PySTK game state to gather the label ground truths. [11] The game state offered the puck and kart locations in 3D world coordinates, as well as the view and projection matrices of all karts. For the puck centers, we used the view **V** and projection **P** matrices to transform from 3D world coordinates c_{world} to coordinates c_{view} relative to the camera [12] via the mathematics transformation below:

$$c_{view} = \mathbf{V} * \mathbf{P} * c_{world}$$

The resulting c_{view} contains the puck location in the accompanying image, as well as the puck distance as an additional result. We then determined the puck visibility (boolean) label by checking the logic if c_{view} is within the image boundary coordinates $\in$ [-1, 1], and if the normalized distance is a positive number $\in$ [0, 1].

The achieved labels for puck visibility, puck distances, and puck center coordinates were almost identical between the two mentioned approaches, e.g. the calculated puck center coordinates deviated on average less than a single pixel between the approaches. However, there still exist a few notable differences. Firstly, the two puck-camera distance estimations always had a discrepancy, because they actually measure slightly different things. While the value from the depth map of the image analysis approach measures the distance from camera to puck surface, the depth from the game state actually measures the distance to the puck center. Secondly, many cases in which the center coordinate estimates had a discrepancy, were generally from images that only contained a part of the puck, e.g. when almost half of the puck was outside the viewing frame. In such scenarios, the image analysis approach that approximated the puck center with the mean of the segmentation array had a large and unavoidable systematic error. Thirdly, there were similar problematic instances for the puck visibility label derived from the game state approach. When the puck center from the game state was just outside the image, even if there were still quite obvious parts of the puck inside the images, the game state approach labeled it as no puck identified. It was our assumption that this would negatively influence the gameplay of the autobot.

Therefore, we finalized our method to collect the ground truth labels by combining the results from both mentioned approaches. On the one hand, the binary classification label is taken from the image analysis approach. On the other hand, we used the more accurate puck center coordinates from the game state, except in cases where the image analysis and game state labels diverge, then we used the coordinates from the image analysis. The same strategy applied to the puck-camera distance: the game state value was used as the base case, except when the puck labels disagree. Then, the distance value from the depth map was adjusted to approximate the puck center instead of puck surface, and this value was used for the distance ground truth.

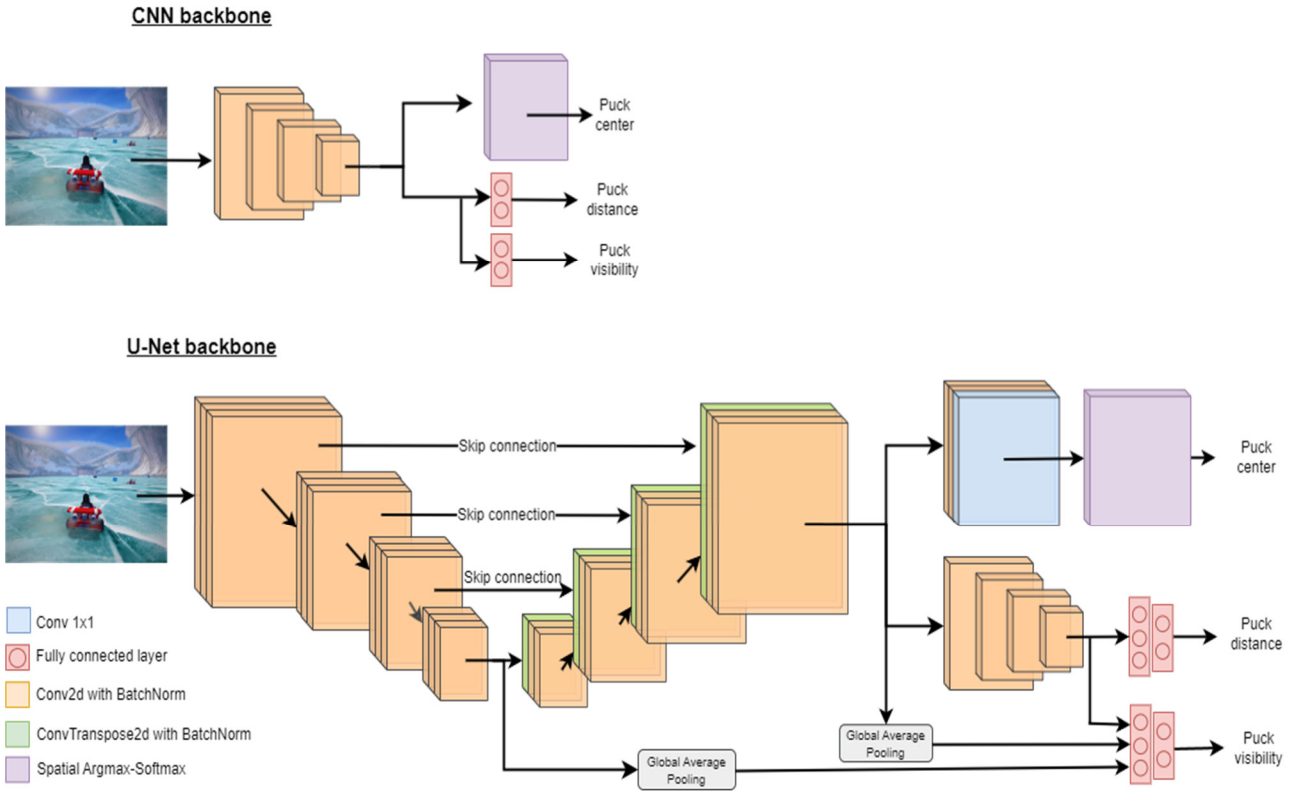
With these data collected, before feeding them to the training process, they were also pre-processed with certain transformations, including resize (if applicable), augmentation, and input normalization. Image resizing primarily helps to reduce training time to identify the lead case among different deep network candidates. Global input normalization using training set mean and variance helps to standardize the data and improve the convergence. [13] Image data augmentation with random horizontal flip and color jitter (with brightness, contrast and saturation) generalizes the training by exposing the model to new data (very similar but randomly distinct training examples) for every epoch, this reduces overfitting problem for training, especially with a large number of epochs. [14]

B. Proposed vision network architectures

In this study, we propose a multi-head network architecture for puck detection featuring two different investigated backbones: U-Net [15] and typical convolutional neural network (CNN) (Figure 1). Our study's scope is to design and evaluate the performance of both types of these networks for simple object detection and classification tasks.

To begin with, the U-Net architecture consists of encoder and decoder parts, which respectively extract relevant features from input images and build segmentation masks from the features. The encoder contains 4 blocks of convolutions for

Figure 1. Model training & validation visualization



down-resolution of the input. Similarly, the decoder also contains 4 blocks of up-convolution to enlarge the compressed information and to produce a full-resolution heatmap (same as the original). The number of intermediate channels in these convolutional layers varies between our different models and more details about our model optimization can be found in Table 1. The architectural decision to use a U-Net as the backbone for an object detection model, was inspired by the similar Hourglass model for pose estimation. [16]

Within each block of convolutions, we employ batch normalization layers. Batch normalization was reported to help dramatically accelerate the training process of a neural network by reducing internal covariate shift, which was achieved through automatic standardization of tensors within a batch. [17] The use of batch normalization in every single structural block of our network reduces initialization dependency as well as the risk of divergence during optimization.

Additionally, specifically for these U-Net models, skip connections were utilized. These skip connections enhance training efficiency of deep networks without gradient degradation problems by using an identity mapping, where the input of a layer is added to the output of a different layer. [18] To the best of our knowledge, skip and the related residual connections are not only incorporated into U-Net design but also other modern networks, such as ResNet models, [19] [20] LSTM models [21] and transformer models. [22]

In terms of the activation layers, it is also worth noting that in our optimized model, instead of the commonly used Rectified Linear Unit activation (ReLU), we deployed Sigmoid-weighted Linear Units (SiLU). The activation function of SiLU has a non-zero gradient for negative numbers that enable a self-stabilizing property during backpropagation, because now weights with negative activations can be updated. Therefore, it outperforms ReLU in many deep networks and is a common practice in computer vision. [23]

In contrast to the U-Net architecture, the CNN network has a simpler design, which consists of 4 blocks of convolutional layers that generate a heatmap of reduced size compared to U-Net. The main purpose of evaluating the CNN is to compare model performance with U-Net and from that, we can select a suitable model for the prediction of puck information that has an acceptable level of accuracy as well as a reasonable requirement of computational resource, since we are constrained with a 50 milliseconds limit for 1 full cycle of action from inputting the images to model to deciding the action for the kart.

Behind the backbone, we added the three heads of our model. For the puck visibility classification head, we strive for efficiency by reusing already calculated intermediate activations for the regression heads. Intermediate activations were flattened, concatenated, and only two further fully connected dense layers (FCs) followed. Dropout was employed in these linear layers during the training process to reduce overfitting. [24]

Our model can be best described as the so-called multiple-head (multiple-task) network that was deployed for the prediction of multiple labels using the same convolutional network backbone. [25] [26] In our scenario, puck visibility is first predicted using a binary linear classifier and hence it is trained with a BCE loss function. Second, puck center coordinate is predicted using a SoftMax algorithm to detect the

peak of the single channel heatmap derived from the convolutional network. Lastly, puck-to-camera distance is predicted by a combination of multiple linear layers. It is worth noting that we also examined the addition of a few convolutional layers after the heatmap generation (for the puck center and puck distance regression heads) to reduce the number of parameters in the FCs, with the hope to accelerate computing efficiency.

Subsequently, these deep network models were trained using MacBook Pro with M2 chip including 8-core CPU with 4 performance cores and 4 efficiency cores, 10-core GPU and 16-core Neural Engine.

C. Controller design and logics

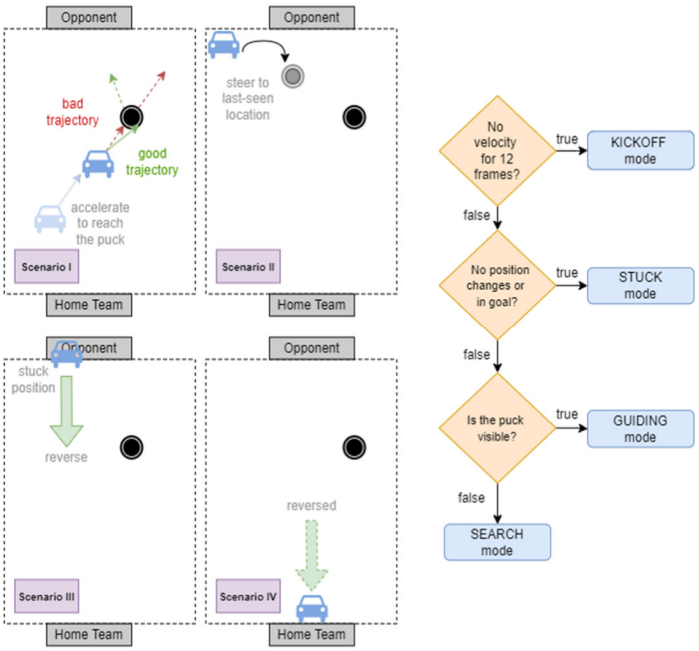
With the puck information in hand, we designed the controller in a four-mode operation that includes kick-off mode, stuck mode, puck-searching mode and puck-guiding mode. The playground has some tricky "dead" spots at which the karts are usually unable to move forward and get stuck, such as the corners of the field and the area inside the goals. First and foremost, to define a kart in a stuck position, we utilized a back-tracking algorithm that considers its recent velocity as well as its center location. If there is no change in both values within a given timeframe (i.e. previous 5 game steps), the kart will be put in stuck mode operation. As can be seen in Figure 2, we have 2 different strategies to tackle this issue, which are illustrated in scenario II, III and IV. One of the smart ways is to get the kart to memorize several last seen locations of the puck (the last puck center coordinates when it was still present in the player's view). This short-term memory is very useful for controlling a stuck kart to steer towards the direction with high probability of having a puck present. A failed attempt of this would lead to a reverse movement (triggered by zero acceleration and break is true) until the kart reaches its home team goal, at which, it switches into puck-searching mode again.

It is important to note that during the game kick-off, all karts are at zero speed and no change in location, hence, it could be mistaken with stuck condition. Therefore, we designed a special state for the kick-off in order to differentiate it from other stuck scenarios. In kick-off state, the kart would move toward the puck with highest acceleration, instead of following stuck operations.

In the mode of puck-searching, the kart is moving around and gets the puck in its vision before it quickly accelerates to reach the puck's location. (Figure 2 – scenario I). Puck-guiding mode is designed based on a simple but logical algorithm that by drawing a vector from the kart center to the puck center, if the opponent's goal (target) is on one side of this vector, the kart should approach the puck slightly on the other side of this vector. These repetitive actions carry on until the kart guides the puck to the target. No matter where the locations of the kart and the puck are, this strategy still applies as long as the puck is within the kart's view.

Last but not least, given the primary objective of the game is to score as many goals as possible, we strategized both of our karts to play in an offensive manner, which is to keep searching for the puck and guiding it towards opponent's goal and no defensive mechanism was applied in the current version of our autopilot model.

Figure 2. Controller design



III. RESULTS AND DISCUSSIONS

A. Network training and performance

The five deep networks listed in Table 1 were subjected to a training and validation process using an initial learning rate of 10^{-3} with reduce on plateau scheduler, a batch size of 32, an epoch number of 150, and an Adam optimizer with weight decay set at 10^{-5}. Since our multi-head networks contain 3 loss functions, the loss weight parameter for each was set equally. During the training process, although all of the models yielded relatively good results, we still experienced some minor issues.

The first training problem is that initially using U-Net$_2$ we achieved poor prediction on puck-camera distance. It can be attributed to how we first default labeled puck centers as [-2, -2] for the case where the puck is not visible in the image. We troubleshot this issue by creating a mask to exempt instances with puck visibility equaling False from the loss function for training the other tasks (puck center and puck distances).

TABLE I. MODEL TRAINING COMPARISON

Model	Network training validation details		
	Model architecture	***Total Loss***	***Accuracy***
U-Net$_1$	(input = 3 x 144 x 192) D_{16}-D_{32}-D_{64}-U_{64}-U_{32}-U_{16}-Ln	0.052	0.992
U-Net$_2$	(input = 3 x 144 x 192) D_{16}-D_{32}-D_{64}-U_{64}-U_{32}-U_{16}-Ln	0.023	0.994
U-Net$_3$	(input = 3 x 144 x 192) D_{32}-D_{64}-D_{96}-D_{128}-U_{128}-U_{96}-U_{64}-U_{32}--Cvs-Ln	0.003	0.971
U-Net$_4$	(input = 3 x 216 x 288) D_{32}-D_{64}-D_{96}-D_{128}-U_{128}-U_{96}-U_{64}-U_{32}--Cvs-Ln	0.001	0.991
CNN$_1$	(input = 3 x 144 x 192) Cv_{16}-Cv_{32}-Cv_{32}-Cv_{32}-Ln	0.079	0.988

D: Down-convolution, U: Up-convolution, Cv: Convolution, Ln: Linear blocks

The second training problem observed was for the case of U-Net$_3$, due to its small input size (300x400 resized to 144x192), the input resolution was not acceptable for predicting images with a very small puck (the puck is far away from the camera). This leads to several cases of false negative classification of puck visibility. Hence, we improved this in the U-Net$_4$ version, which shares the same network architecture but takes a larger input size of 216x288. We observed a notable increase in classification accuracy.

Furthermore, despite the fact that CNN$_1$ is the simplest model among all, the validation loss from this model was much higher compared to the rest of the U-Net architecture. Given that we tested our most computationally heavy U-Net (U-Net$_4$) and verified the calculation speed was far below the threshold of 50 milliseconds per step of action in game, we did not select CNN$_1$ for this reason.

Overall, from Figure 3 we can see that U-Net$_4$ is the most promising model since it has significantly lower total loss than the rest. This is attributed to very good puck center and puck distance prediction, with high accuracy of classification on puck presence. Numerical data provided in Table 1 reflects this observation. As a result, we selected this model as our optimized predictor to proceed with the controller design step.

B. Controller evaluation

Our controller was then evaluated and benchmarked against the available state-based agents. At the same time, we also created a dummy image-based agent which follows no specific strategy and only takes random action every step (random steer, random acceleration, etc.).

Our metrics to benchmark the performance of those agents includes the average goals scored, the average stuck time of the karts and the average puck-kart distances. Besides the obvious success measurement by the number of goals scored, our rationale is that a good controller should have minimum amount of stuck time (which means every time the kart got stuck at "dead" spots, it could get out as soon as possible) and a minimum total puck-kart distances (which means the karts should always trace the puck closely).

Based on these metrics, we evaluated all the agents by simulating three 2 vs 2 games between each pair of agents (63 games in total). The results were demonstrated in Figure 4. Based on the average number of goals scored, the provided AI agent performed the best, followed by Jurgen agent and our developed image-based agent. Clearly, the dummy agents scored the least and mainly their goal came from own goals by its opponents.

Figure 3. Model training & validation visualization

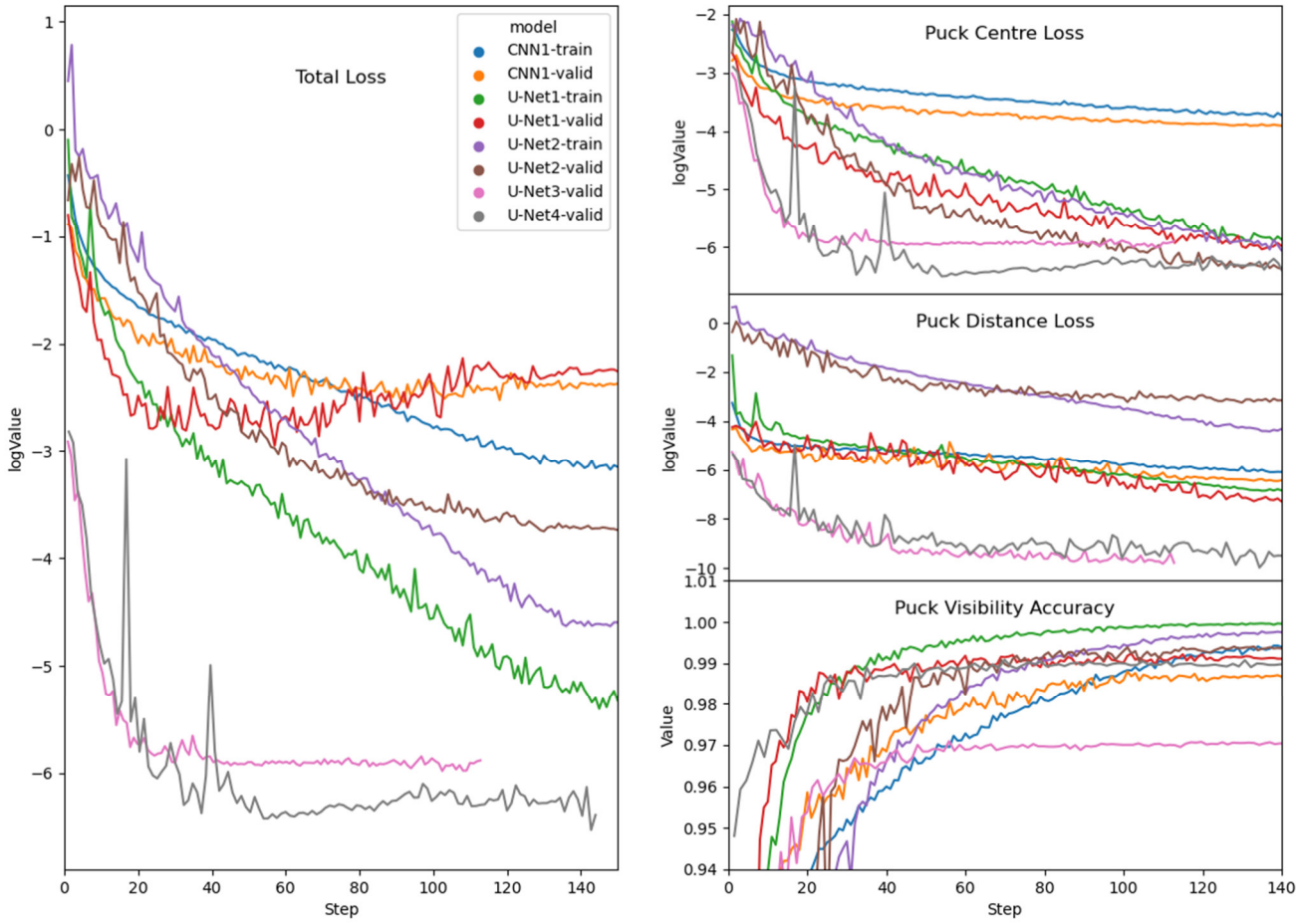
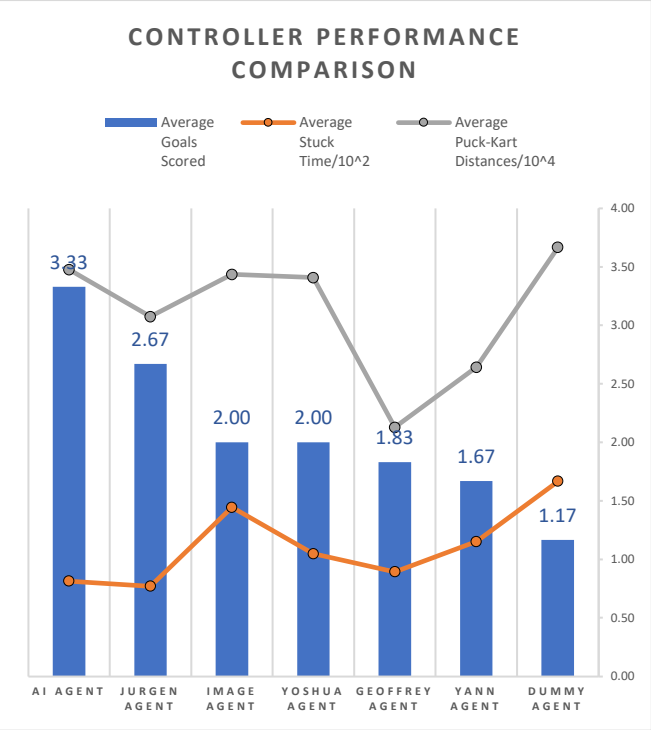


Figure 4. Controller evaluation chart



Our image-based agent could have scored better if the puck-tracking mechanism was better and stuck time was minimized. Although its performance was among the top 3 best players, it should be noted that the results may statistically vary if we replay several games. Definitely, there are multiple ways to upgrade our agent's performance, which will be mentioned in the next section.

IV. Conclusion

In conclusion, our designed autobot for the game SuperTuxKart (ice hockey map) gained a certain level of success, as proven by the number of goals it was able to score against established AI agents. However, there are still potential improvements to our model as well as our controller which could have perfected the bot. Firstly, while using the trained model U-Net$_4$, we observed quite a number of false positive predictions. Our agent sometimes misclassified the goal post as the puck. In order to solve this problem, we may increase the resolution of input further or increase the training set size by augmenting the training set with the observed false positive cases. We may also try to increase the dropout rate or use 2D dropout for the convolutional layers (the current model only used 1D dropout in the FCs part). Secondly, we realized that our predicted distance information was underutilized in the controller. This piece of information can help to unlock the world coordinate of the puck, which, subsequently, aids in guiding a better course of actions.

Furthermore, future implementations of our work can also be improved with some changes in the design. First, since this game operates in 2 vs 2 mode, the two players under our control can actually share knowledge of the puck information, so that in the event of one getting stuck or unable to locate the puck, it can receive the intel from its partner and proceed toward the puck's location. Second, the information about the opponent team was totally neglected in our design, this can be developed in such a way that our karts learn to differentiate and take hostile actions toward the opponent team. Doing this would definitely slow down the opponents. Third, we also ignored the use of boost-up (bonus) in-game items. Leveraging these would certainly add benefits to the team and gain advantages over the opponents.

Acknowledgment

We would like to thank Dr. Philipp Krähenbühl for his comprehensive lectures in deep learning, computer vision and reinforcement learning. Furthermore, we would like to express our gratitude to the teaching assistant team for their continuous efforts to clarify and to guide us throughout this project.

References

[1] J. Max, M. C. Wojciech, D. Iain, M. Luke, L. Guy, G. C. Antonio, B. Charles, C. R. Neil, S. M. Ari, R. Avraham, S. Nicolas, G. Tim, D. Louise, J. Z. Leibo, S. David, H. Demis, K. Koray and G. Thore, "Human-level performance in 3D multiplayer games with population-based reinforcement learning" *Science,* vol. 364, no. 6443, pp. 859-865, 2019.

[2] M. Volodymyr, K. Koray, S. David, G. Alex, A. Ioannis, W. Daan and A. R. Martin, "Playing Atari with Deep Reinforcement Learning" *Computing Research Repository,* vol. 1312, p. 5602, 2013.

[3] S. Farooq, I. Oh, M.-J. Kim and K. Kim, "StarCraft AI Competition: A Step Toward Human-Level AI for Real-Time Strategy Games" *AI Magazine,* vol. 37, no. 6, pp. 102-106, 2016.

[4] D. George, C. Eirini and K. I. Dimitris, "E pluribus unum interpretable convolutional neural networks" *Scientific Reports,* vol. 13, p. 11421, 2023.

[5] I. Sonata, Y. Heryadi, L. Lukas and A. Wibowo, "Autonomous car using CNN deep learning algorithm" *Journal of Physics: Conference Series,* vol. 1869, no. 1, p. 012071, 2021.

[6] B. S. K. Manju and M. Sowjanya, "An Experimental Analysis on Self Driving Car Using CNN" *International Research Journal of Engineering and Technology,* vol. 09, no. 04, pp. 7-12, 2022.

[7] V. Vaquero, A. Sanfeliu and F. Moreno-Noguer, "Deep Lidar CNN to Understand the Dynamics of Moving Vehicles" in *IEEE International Conference on Robotics and Automation*, 2018.

[8] P. Svetlana, L. Nico, K. B. Ashok and J. M. Zöllner, "Traffic Light Recognition using Convolutional Neural Networks: A Survey" arXiv, 2023.

[9] C. Ziyi, L. Ruixiang, L. Jonathan, D. Jixiang and W. Cheng, "U-Net Based Road Area Guidance for Crosswalks Detection from Remote Sensing Images" *Canadian Journal of Remote Sensing,* vol. 47, no. 1, pp. 83-99, 2021.

[10] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars" in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[11] P. Krähenbühl, "Python SuperTuxKart" 2019. [Online]. Available: https://pystk.readthedocs.io/en/latest/state.html.

[12] H. A. Song, "OpenGL Projection Matrix" 2024. [Online]. Available: https://www.songho.ca/opengl/gl_projectionmatrix.html.

[13] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems" *IEEE Transactions on Nuclear Science,* vol. 44, no. 3, pp. 1464-1468, 1997.

[14] X. Mingle, Y. Sook, F. Alvaro and P. Dong Sun, "A Comprehensive Survey of Image Augmentation Techniques for Deep Learning" *Pattern Recognition,* vol. 137, p. 109347, 2023.

[15] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation" in *Medical Image Computing and Computer-Assisted Intervention -- MICCAI 2015*, 2015.

[16] J. Tompson, R. Goroshin, A. Jain, Y. LeCun and C. Bregler, "Efficient object localization using Convolutional Networks" in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[17] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" in *Proceedings of Machine Learning Research*, 2015.

[18] H. Kaiming, Z. Xiangyu, R. Shaoqing and S. Jian, "Deep Residual Learning for Image Recognition" *Computing Research Repository,* vol. 1512, p. 03385, 2015.

[19] H. Kaiming, Z. Xiangyu, R. Shaoqing and S. Jian, "Deep Residual Learning for Image Recognition" *Computer Vision and Pattern Recognition,* vol. 1512, p. 03385, 2015.

[20] X. Saining, G. Ross, D. Piotr, T. Zhuowen and H. Kaiming, "Aggregated Residual Transformations for Deep Neural Networks" *Computer Vision and Pattern Recognition,* vol. 1611, p. 05431, 2017.

[21] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory" *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[22] D. Yihe, C. Jean-Baptiste and L. Andreas, "Attention is Not All You Need: Pure Attention Loses Rank Doubly Exponentially" *Computing Research Repository,* vol. 2103, p. 03404, 2021.

[23] E. Stefan, U. Eiji and D. Kenji, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning" *Neural Networks,* vol. 107, pp. 3-11, 2018.

[24] S. Nitish, H. Geoffrey, K. Alex, S. Ilya and S. Ruslan, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" *Journal of Machine Learning Research,* vol. 15, no. 56, pp. 1929-1958, 2014.

[25] V. Ashish, S. Noam, P. Niki, U. Jakob, J. Llion, N. G. Aidan, K. Lukasz and P. Illia, "Attention Is All You Need" *Computing Research Repository,* vol. 1706, p. 03762, 2017.

[26] P. Michel, O. Levy and G. Neubig, "Are Sixteen Heads Really Better than One?" in *Advances in Neural Information Processing Systems*, 2019.