Table of Contents

Introduction & Motivation

The goal of this work was to create an intelligent guidance system for a 2-player video game ice hockey team, via a neural network-driven controller that can score goals autonomously against Artificially Intelligent (AI) opponents. The game, SuperTuxKart, provides data sufficient to develop either a state-based or image-based agent. A state-based agent would utilize game state, kart state, and puck state information (location, velocity, acceleration, steering, etc.) to determine the next action for each player to take. Conversely, an image-based agent would function as a vision system from the perspective of allied players, using object detection as well as limited state information to decide the team's next actions.

We decided to implement an image-based controller by training a convolutional neural network object detection model using PyTorch's Deep Learning infrastructure. The motivation for this approach was twofold. First, an image-based controller facilitates the integration of numerous techniques explored during the course's curriculum. The information available to analyze (team members' kart states, the stationary position of goals on the map, and the images themselves) made this an applicable use of a convolutional neural network. Moreover, an image-based controller, as opposed to a state-based controller, presented what we viewed as a more interesting problem, namely, identifying the presence of an object on screen and establishing a plan of action in real-time similar to how a human would likely play the game.

To this end, a four step approach was developed, detailed in **Table 1**, to create the image-based agent controller: data collection, model development, model training, and controller tuning. Each step consisted of multiple variations with closed-loop testing, while fine tuning was performed based on examination of the performance of the entire end-to-end process.

Table 1. Summary of Project Strategy and Approach

Collect Training Data	**Build Model**	**Train Model**	**Tune Controller**
Use provided test agents to play game **Data**: image with 300 x 400 resolution and 3 color channels **Labels**: indicator for puck visible on screen with xy-coordinates of location where x, y ∈ [-1,1]	Build convolutional neural network (CNN and FCN) **Input**: Bx3x300x400 batch of images **Output**: Bx3 batch of predictions with outputs **detection** ∈ {0,1}: is the puck visible in image? **location x, y** ∈ [-1,1]: where is the puck located in the image?	Train (and tune) the model to detect puck presence and predict puck location **Loss function:** **detection_loss** using torch.nn.BCEWithLogitsLoss() **location_loss** using torch.nn.MSELoss()	Develop (and tune) controller after model has been trained **Input**: model prediction for whether puck is visible on screen, and if so, relative xy-coordinates. **Output**: one gameplay action for each player on the team.

In the following sections the methodology, considerations, and results for each step in our approach are provided, as well as a detailed analysis of the final end-to-end implementation's success or failure. In the interest of follow-up, potential avenues for further improvement are provided.

Data Collection & Labeling

The initial step was to procure a training data set, because no existing labeled set was readily available for use. Therefore, the provided agents were pitted against each other to produce frame-by-frame match data. To get a variety of behavior from the players, data was collected from several different matchups. The AI agent appeared to make the most "human-like" decisions so the highest importance was given to data collected from the AI vs. AI matchup (1500 frames). Additionally, we played AI vs' Yann agent (500 frames) and Yann agent vs. itself (500 frames).

This approach produced 12,000 images with game state, kart state, and image data. As this is too large of a data set to manually label, a programmatic approach was developed to determine whether or not a given object (in this case, the puck) was present on screen. The main challenge to generate labels was translating the world coordinates of the puck into relative coordinates with respect to the kart field of view. This was accomplished using the camera projection and view matrices, via the logic below, determined by research into video game design [1]:

> Let R be the desired relative coordinates, W be the coordinates in world space provided by the game state, P be the camera projection data, and V be the camera view data. Then:
>
> $$R = P^T V^T W$$

Finally, these coordinates needed to be converted into homogeneous clip space by dividing the positional components of the relative location by the perspective ratio, R[-1], and capping the resulting values to the range [-1, 1]. In order to assign the boolean labels, records with x, y, and z relative coordinates between [-0.97, 0.97] were labeled as on_screen = True, while all other records labeled as on_screen = False. An additional 0.03 buffer was added to the visible field of view to filter out images where the puck was not on the screen (clipped values at the end points) and instances in which there was insignificant puck presence at the edge of the field of view. In our estimation, the downside of predicting false negatives for small slivers of puck on the edge of the screen was outweighed by the reduced potential for the model to activate on small groupings of erroneous dark pixels (such as shadows, kart wheels, and the brick edges of the arena). The puck location targets were then set as the x and y values of the homogeneous clip space output.

Initially, a precision of one-tenth was used for location targets. This was subsequently changed to the hundredths in order to increase pixel-wise precision from 20x15 to approximately 2x2. Keeping in mind the goal of steering to hit the puck at an angle that propels it towards the opponent's goal, the initial 20x15 pixel-wise precision did not provide a sufficient level of granularity.

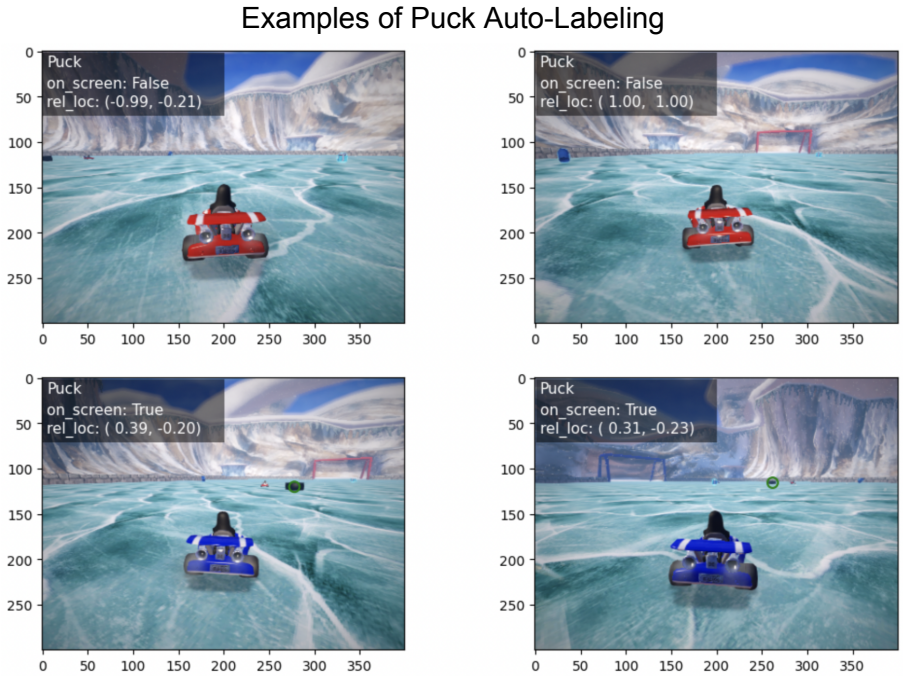

Figure A. Four examples of the auto-labeling for whether the puck is on screen using relative x,y coordinates. Green circles highlight the labeled location of the puck, provided it is visible.

Figure A highlights four different test cases of the automated puck labeling. The top left image showcases the 0.03 edge buffer used when creating labels. Despite the puck being barely visible on the left edge of the image, a label of False was assigned because the relative x location (-0.99) is beyond the buffer. The top right image is correctly labeled because it contains no puck, while the bottom two images highlight the capability of the automatic labeling to accurately pinpoint both near and far puck presence.

Additionally, the same automated data labeling used for puck location was applied to enemy goal and enemy kart locations in an identical format. This yielded a versatile and robust training dataset with 15 total labels that could be used to develop a single or multi-object detector. Ultimately, due to loss function considerations and the prioritization of accurate puck detection, only the puck labels were used in a single object detection model. Further details about the potential for a multi-object detector for puck, kart, and goal posts is discussed in the Future Improvements section.

Because AI-based gameplay was leveraged to generate the training data, there was a possibility of repeating game loops that could result in inadequate coverage of the domain of potential states in a random game. Therefore, additional validation was performed to ensure

puck and kart locations were distributed across a range of possible scenarios, detailed in **Figure B** below.

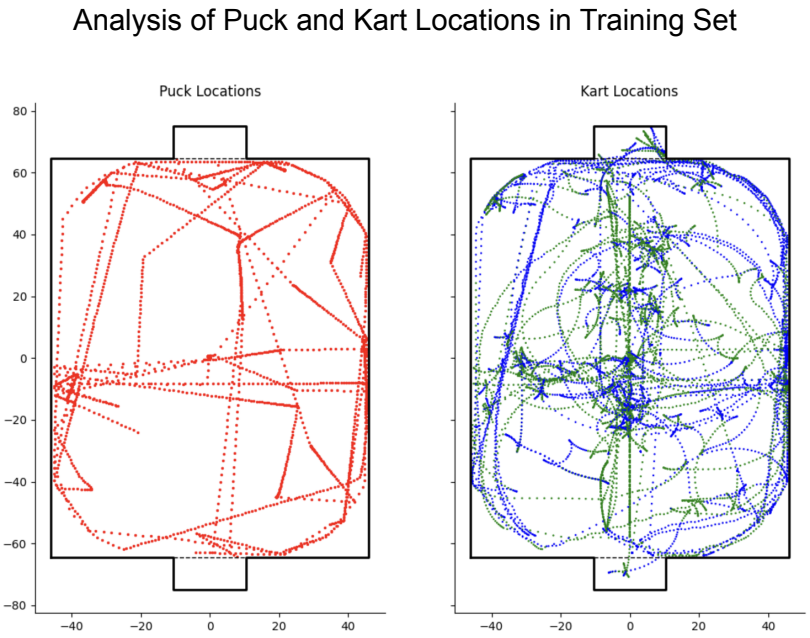

Figure B. Left contains training puck locations. Right contains training kart locations, colored green if the puck was visible and blue if it was not.

This analysis shows that the training sample had an adequate level of variability such that any model developed from it should be equipped to handle all possible puck and kart locations. With this 12,000 image, labeled data in proper format and validated for domain coverage, a sufficient foundation was established to begin model development.

Modeling Approach

Model Design & Architecture

Two model architectures were developed and tested: a convolutional neural network (CNN) and fully-connected convolutional network (FCN). Initially the CNN framework was intended for single object detection and the FCN for improvement via multi-object detection and/or a densely labeled data set. While multi-object detection and densely labeled data were both explored, neither approach proceeded to the model training phase. However, the CNN and FCN approaches were still both utilized for puck detection to determine which provided the highest accuracy. This section details the architecture for both the CNN and FCN, discusses considerations for choice of loss function and optimizer, and finally provides an analysis of training performance of each model.

For both designs, the input and output requirements were the same. The networks converted an RGB color image of the playing field captured from a player's field of view into two predictions:

- indicator, $I \in \{0, 1\}$, for whether the puck is present on screen

- $(x, y) \in [-1, 1]$, relative screen coordinates of the puck center

The convolutional neural network developed, detailed in **Figure C**, contains a simple sequence of convolutional layers that is structured to satisfy the trade-off principle such that each time an image input is reduced to half its original dimension, the number of channels doubles. The design of CNN was built with consideration of common practices shared in this course, as well as recommendations summarized by Alzubaidi et. al. [2].

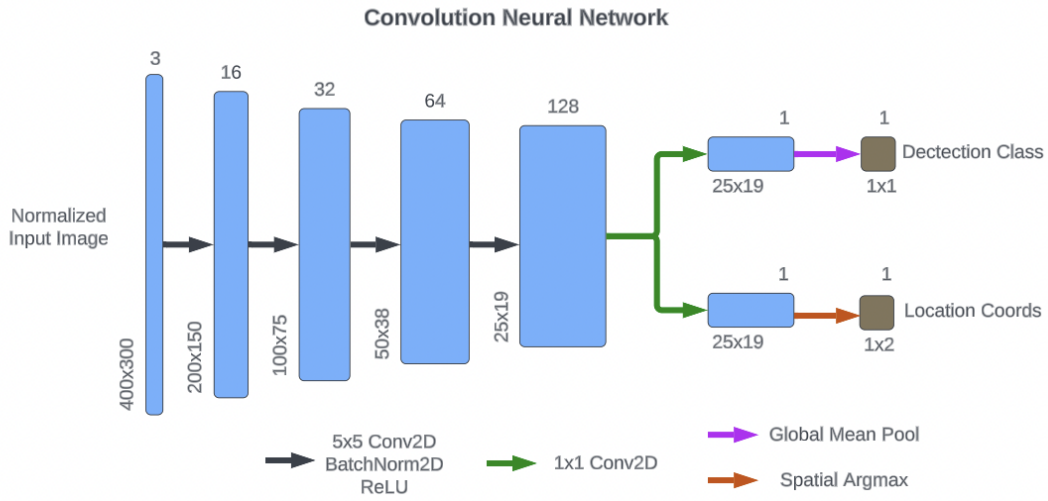

Figure C. Structure of Convolutional Neural Network (CNN)

The network was composed of six total layers. The bulk of the feature mapping occurs in the initial four layers, each consisting of a 5x5 convolution with a stride of 2 and padding of 1, a 2D batch normalization layer, and a final rectified linear unit. A convolution size of 5x5 was chosen to increase the receptive field of the downstream features to account for a large image resolution. These four layers alter the original input dimensions of 3x300x400 into a 128x25x19 feature space. This is finally followed by two independent processes, a classification layer for detection and a 2D output layer to create the xy-coordinate position.

The classification layer consisted of a 1x1 convolution followed by global mean pooling, which outputs a logit value that was then thresholded at 0 to determine positive detection and negative detection classes. The coordinate output layer consisted of a 1x1 convolution followed by a spatial argmax computation to produce 2D positional coordinates in the [-1, 1] domain.

The FCN design was modeled after the common U-net architecture [3] in which a sequence of down convolutional blocks reduces the dimensionality while increasing the number of channels, followed by a sequence of up convolution blocks that increase the dimensionality while reducing the number of channels. Residual connections and skip connections were also utilized in order to provide more robust feature activation and smooth gradient flow for the deeper network.

The down-convolutional block is illustrated in **Figure D**. The following are key architectural components of the down block.

- 2D convolutions: provides transformation of input feature space.
- Batch normalization: prevents vanishing or exploding gradients during a forward pass by normalizing (shifting and scaling) the inputs of each layer
- Residual connections: facilitates smooth gradient flow by providing a direct path for data inputs to reach (skip to) later parts of the network.
- ReLU activation: introduces non-linearity to the network

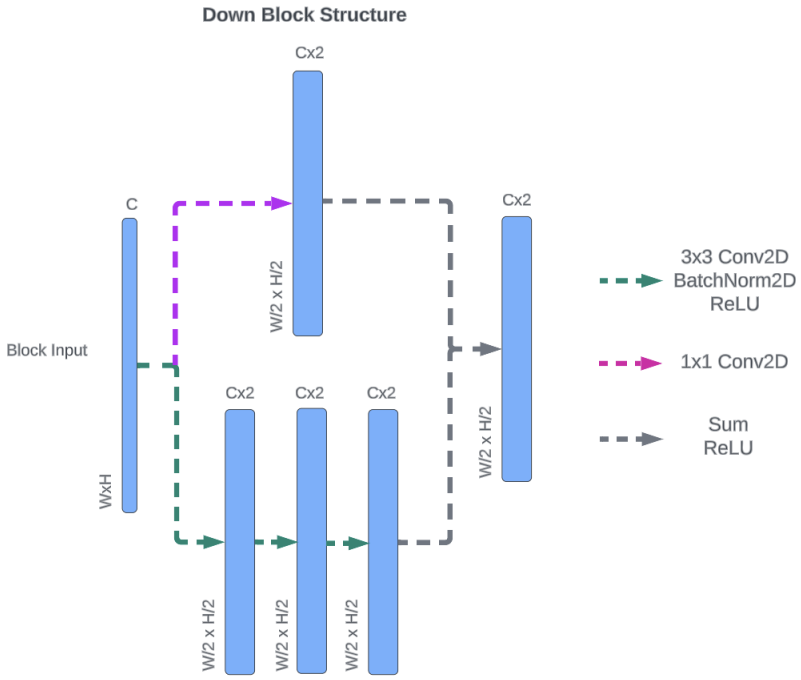

Figure D. Structure of a Down Block in a Fully Convolutional Neural Network (FCN)

The up blocks were made up of a single 3x3 convolution with a stride of 2 and padding of 1 followed by a ReLU. Finally, the same classification layer and xy-coordinate output layer were used in both the CNN and FCN. The full FCN structure is detailed in **Figure E** below.

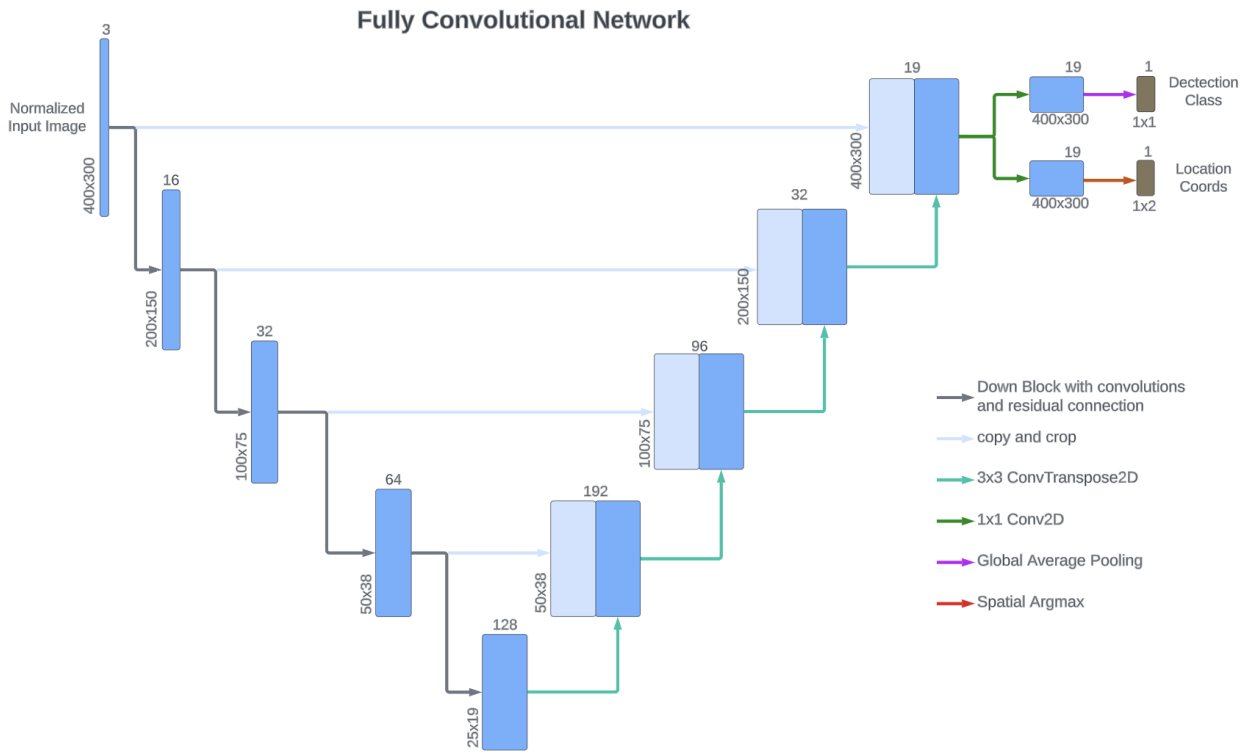

Figure E. Fully convolutional network consisting of four down blocks with residual connections and four up blocks with skip connections.

With the architecture of both the CNN and FCN in place, the next consideration was on the training setup. The dual output format of both models, containing a boolean classification and two-dimension positional values, posed an interesting optimization challenge. The overall loss function needed to incentivize the model to update in a manner that balanced both the discrete class assignment and continuous coordinate values. Thus, a manually tunable loss function was composed that enabled this balancing, in a similar style successfully used by Yue et. all [4] to identify whether a drug-drug interaction would occur and the resulting synergy score, detailed below:

$$Total\ Loss\ =\ BCEWithLogitsLoss(I_p)\ +\ \alpha\, I_r\, MSELoss(x, y)$$

Where α is a tunable parameter to balance the class and location losses, I_p is the detection classification of the model, I_r is the true class label, and x,y are the coordinate predictions. The MSELoss was only considered for records where the puck existed on screen, as it would be illogical to have the model attempt to improve the xy-coordinate prediction when the object does not exist.

For both models, the Adam optimizer was used due to its success on smaller datasets and training stability with respect to hyperparameter selection, as detailed by Kingma & Ba [5]. The

decrease in generalization using Adam was not of significant concern, as the environment of future gameplay would be very similar to the training data set (i.e. no large variance in arena styling, team colors, or puck graphic). Lastly, the learning rate was initialized with the commonly used value, 0.001, and scheduled to decrease by 50% every 10 epochs to prevent progress from stagnating.

Model Training & Selection

Both models were trained on the full 12,000 image data set, under various parameter settings, and evaluated on detection accuracy ($\frac{correct\ detections}{num\ samples}$) and location accuracy (using R^2 of the xy-coordinate as a proxy). The best performing CNN and FCN models are shown below in **Figure F** and **Figure G**.

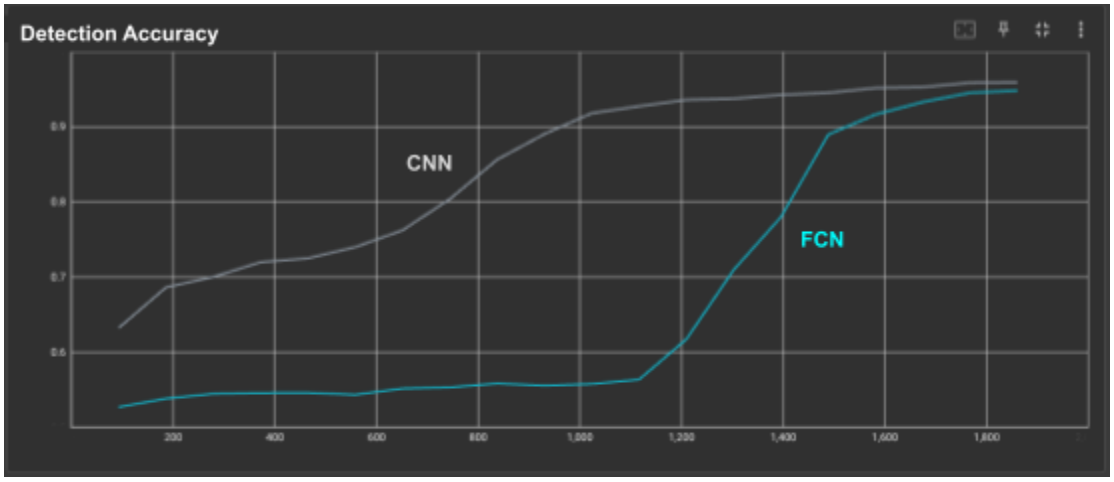

Figure F. Accuracy of puck detection for CNN (gray) and FCN (blue). Puck detection accuracy was slightly higher for CNN than for FCN during the first 20 epochs of training.

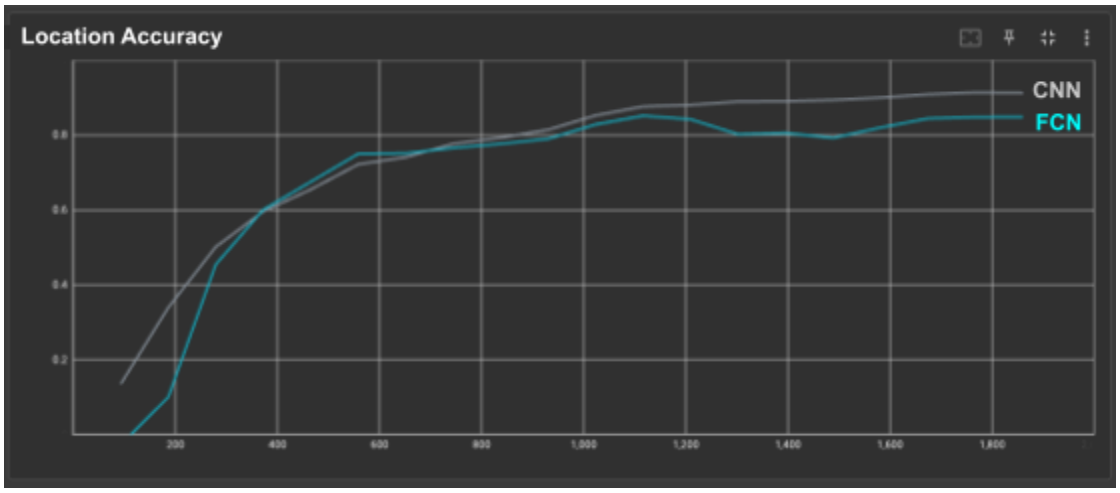

Figure G. Accuracy of puck location for CNN (gray) and FCN (blue). Puck location accuracy was slightly higher for CNN than for FCN after the first 20 epochs of training.

There are a few interesting insights from these graphs that should be highlighted. First, both models were able to reach high accuracy thresholds for both the detection and location outputs. Next, these results showcase the interaction between the two losses, BCEWithLogitsLoss and MSELoss, and the impact it can have during training. The FCN detection accuracy remained almost unchanged for the first 1,000 batch iterations, while the location accuracy continuously improved. Then, once location accuracy leveled-off, the detection accuracy began to climb for the final 850 batches. The FCN seemingly prioritized the two objectives independently, first optimizing for location and then for detection. Conversely, the CNN approach seems to be able to make steady progress simultaneously for both the detection and location outputs. Comparing both models, the CNN slightly outpaced the FCN in both detection and location accuracy. The marginally better performance, as well as the lower computation requirements, led to the CNN model being selected for final implementation.

Once the CNN was selected, a larger set of hyperparameters were tested to maximize performance. An example of this process, the tuning of a scale factor, α, for MSE loss, is shown below in **Figure H**.

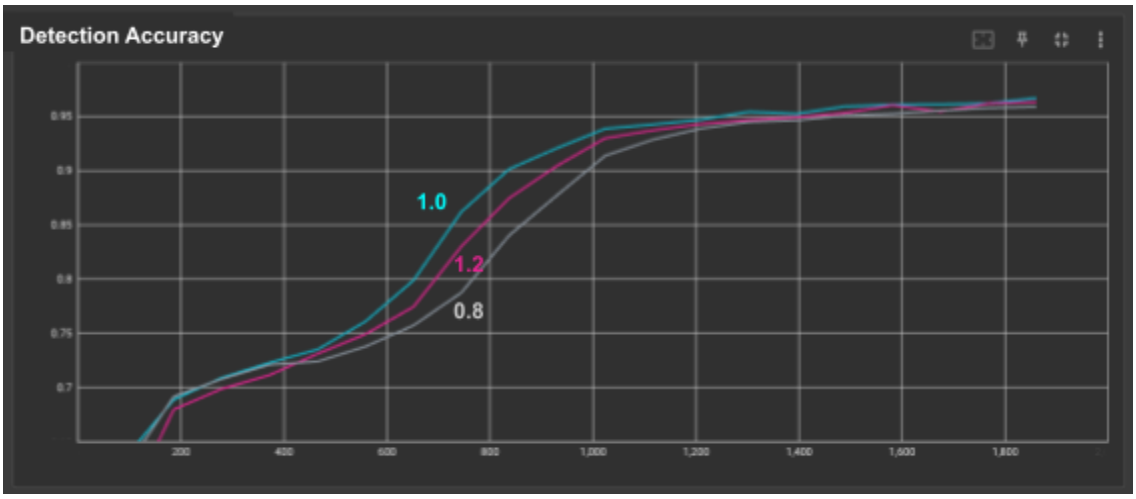

Figure H. Performance of puck detection during training using three different weights for MSELoss..

Ultimately, an equal weighting between BCEWithLogitsLoss and MSELoss was the optimal setting for maximum accuracy of the CNN. Though the improvement was marginal, every additional frame in which our agent knows the puck location provides a greater chance of scoring goals.

Model Validation

After selecting the CNN model for implementation, final validation of the generalizability of this model was performed by running another 200 frame AI vs. AI game. The puck detection accuracy of this validation set was 95% with an average relative puck distance error of 0.052 in the true positive class. The confusion matrix of this test set, along with images showing various examples are presented in **Table 2** and **Figure I**, respectively.

Table 2. Confusion Matrix for Validation Game

	Actual = False	**Actual = True**
Predict = False	416	28
Predict = True	13	343

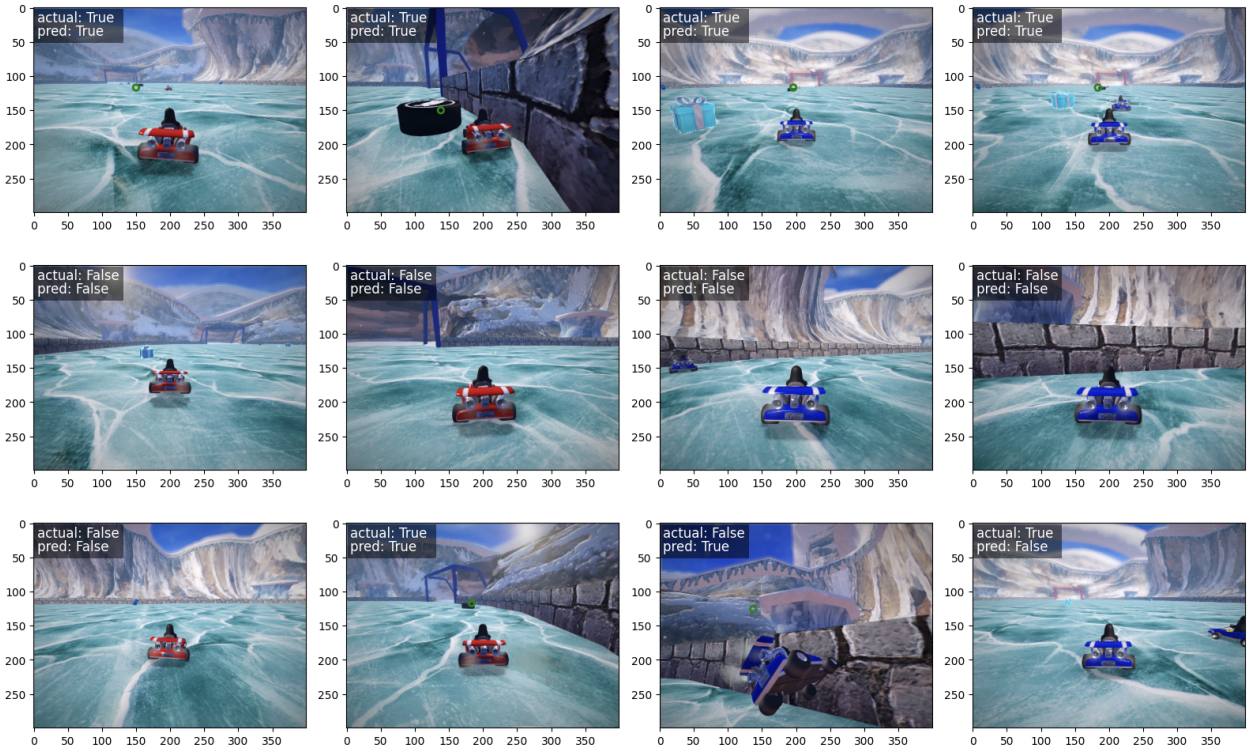

Figure I. Samples from validation set containing actual and predicted puck detections.

This set contains correctly labeled images in both the positive and negative cases. Instances of incorrect model predictions can be seen in the right two images of the last row with the first showcasing a false positive condition. Based on this image, this seems to occur due to the model activating on a dark region on the mountain in the foreground when the kart is knocked into the air. As this is a trivial misclassification (the kart cannot act while in the air), it was not of significant concern. The bottom, rightmost image indicates a false negative case. In this case, the puck is indeed on the screen, but completely hidden by a gift box in between the kart and

puck. In such instances, the negative assignment is actually correct from a visual perspective. These cases occur because the automatic labeling logic did not consider object obstructions, but because this appeared to occur at an insignificant rate no further enhancements were made.

Controller

Controller Methodology

The final step in developing a fully autonomous agent was to use the output of the puck detection model to develop in-game actions. The key considerations consisted of moving towards the puck if it is visible on screen, adjusting the angle of kart-puck impact to direct the puck towards the opponent's goal, changing field of view direction if the puck was not visible on screen, and avoiding regions where karts are likely to become stuck (primarily along the edges of the arena, as well as inside the goals). This was an iterative process in which complexity was added to the controller, a test game was run, a video of the match was inspected to assess the success, and possible improvements were identified.

Controller Design

The foundation for action state assignment was whether or not a puck was detected by the convolutional neural network. Apart from the kart image, the following information was used as additional logic was added in order to optimize overall gameplay ability:

- Kart Velocity- a three-dimensional representation of the kart's velocity
- Kart Front- a vector pointing to the front of the kart
- Kart Location- a three-dimensional location of the kart in the arena (using global coordinates)

This kart information enabled calculation of many other valuable game-state variables. In particular, the Kart Direction and the Kart Angle were calculated using the following formulas:

$$Kart\ Direction\ =\ \frac{kart\ front - kart\ center}{||\ kart\ front - kart\ center\ ||}$$

$$Kart\ Angle\ =\ arctan\left(y_{kart\ dir}\ ,\ x_{kart\ dir}\right)$$

These values assisted in determining where the kart was pointing in any given frame and were stored within a custom class created to maintain all of the necessary features. These features were then converted into in-game action states that dictated the movements a kart would make. In particular, the following actions states were developed:

- Let It Rip- This start-of-game state was focused solely on driving straight ahead as fast as possible. This was utilized to prioritize getting to the puck ahead of the opponent once a puck drop occurs.

- Move to Puck- This is the default game state when the model detects the puck in the image. The kart steers towards the puck and accelerates if below threshold velocity.
- Move to Goal- If the puck is within a close proximity of the kart, then this game state is activated. When activated, the kart begins to turn towards the goal and maintains constant velocity with the goal of hitting the puck at an angle that propels it towards the enemy goal.
- Move in Reverse- When activated, the kart will move in reverse. This state is used when there is no puck detected.
- Change Direction- An action state that is used in conjunction with Move in Reverse. So as to change view direction and to prevent a kart from getting stuck in a loop, the Change Direction state adjusts the steering angle of the kart.
- Move to Arena Center- An action state that is used to help center the kart in the middle of the arena. This state is used when the kart is already in a Move in Reverse state but not changing location. This was to prevent the kart from continuously backing into the walls.

The action state selection criteria is detailed in **Figure J** below. In summary, the controller agent directed a kart towards the puck if one was identified on screen at a distance, directed towards the enemy goal once contact with the puck was made, and attempted to reposition the kart if the puck was not on screen.

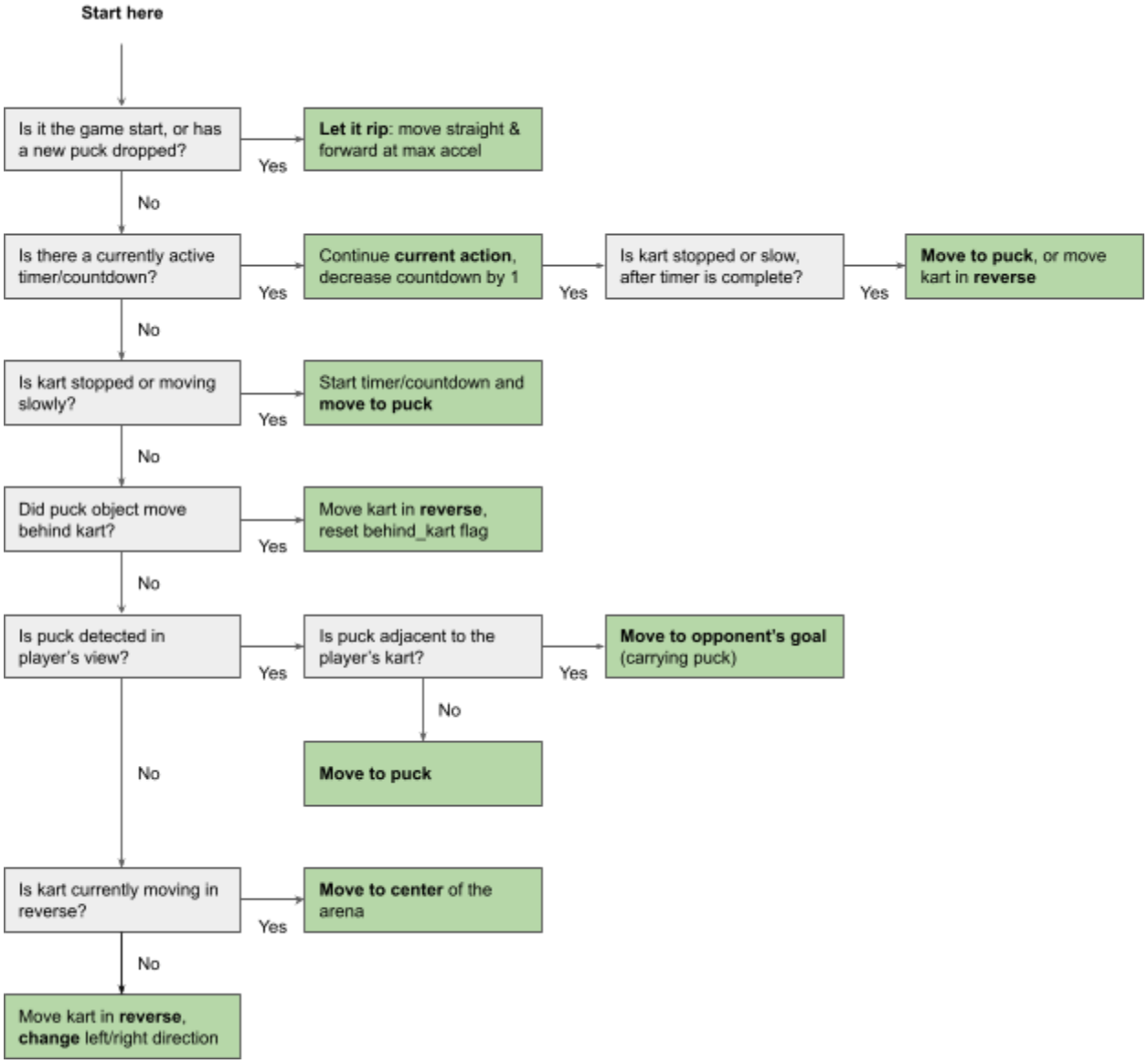

Figure J. Flowchart of our controller algorithm

In order to prevent a kart from entering an unproductive decision loop, the action state of a kart was frozen for a number of frames during certain action loops. This ensured that the kart completed its maneuver to situate itself in a more advantageous position with increased field of view. These actions were intended to increase the likelihood that the kart would be able to bring the puck back into vision and then enter the puck tracking states, Move to Puck and Move to Goal.

While these six action states dictate the final controller logic, many other avenues were explored. These ideas included attempting to convert a puck detection from either kart's relative view coordinates into global coordinates for both karts to leverage. The motivation for this was to enable both karts to navigate towards the puck while only requiring a visual of the puck from

one kart, increasing the percentage of frames in which puck navigation occurs. Unfortunately, we were unable to accurately convert the puck's relative location into global coordinates in all situations.

Additionally, a seventh action state was tested that attempted to position the kart such that it had a straight line alignment from the kart to the puck to the enemy goal. The intent of this action state was to use the global kart location, relative puck location, and known enemy goal location to create a multi-frame navigation alignment that set up an easy shot-on-goal. This state was activated if the puck was detected a significant distance away, measured by the absolute value of the predicted y-coordinate. However, this sequence actually yielded worse results as the position of the puck would change between the alignment phase of the sequence and actually hitting the puck.

Lastly, analysis on the kart characters to use was performed, as the weight, velocity, and acceleration features of the karts are different. We tested several karts:

- Tux (default kart)
- Sara the Racer
- Sara the Wizard
- Beastie
- Konqi
- Wilber

Upon further review, karts did have an impact on acceleration, speed, and turning radius. While specific nuances of the game were more successful (for example, Wilber was very successful at reaching the puck first when a new game was initiated), there was not consistent improvement in scoring when changing characters. As a result, the default 'tux' karts were used for both teammates in the final implementation.

Results

The image-based puck detection logic with a six action state logic controller was tested across 16 matches against 4 different AI opponents. The results were optimistic, as the agent was able to score an average of 1.2 goals per game. The success of this implementation is a product of the highly accurate image-based puck detection model developed, which had over 95% detection accuracy in training, validation, and spot testing throughout development with minimal location error. Additionally, the custom controller prioritizing either kart movement towards the puck or repositioning the kart to maximize the potential of the puck being in view in the next frame was able to convert the model outputs into in-game success.

Albeit a promising result as a whole, the agent still had varying performance in certain games and against different agents. For example, the image agent was unable to score a single goal in 5 of the 16 matches, while scoring 2 or more goals in 6 of the matches. Additionally, only 3 goals were scored across the four matches against Yann while 8 goals were scored in total against Yoshua. This inconsistency game-to-game and opponent-to-opponent indicates that there are

certain configurations in which the agent still does not perform well. Video review of matches during development seemed to indicate that the movement of our AI karts was often in the correct direction, but occasionally lacked the fine grained precision required to consistently put the puck in the net at a high rate. As there were very limited instances of incorrect model predictions, this result can likely be entirely attributed to the controller action decision phase of this implementation. Ideas to decrease the performance variability by adding complexity to the controller are discussed in the next section.

Future Improvements

The controller component of the agent could be improved in a few areas that would require further development and tuning. Therefore, the proposed future improvements focus solely on areas to enhance the action decision logic.

Six gameplay actions were implemented in this solution, but a more intricate and expanded action set could score more goals. One major avenue for this is tuning the angle at which the puck is hit by the kart to direct it towards the goal. Video review of our agent demonstrated a consistent ability to identify and navigate towards the puck, but occasionally struggled in producing an optimal output trajectory. Therefore, while the CNN model was used in this case for single object detection, the training set was developed with multi-class labels for the enemy goal posts and enemy karts shown in **Figure K**, **Figure L**, and **Figure M**.

Target Goal Post Labeling

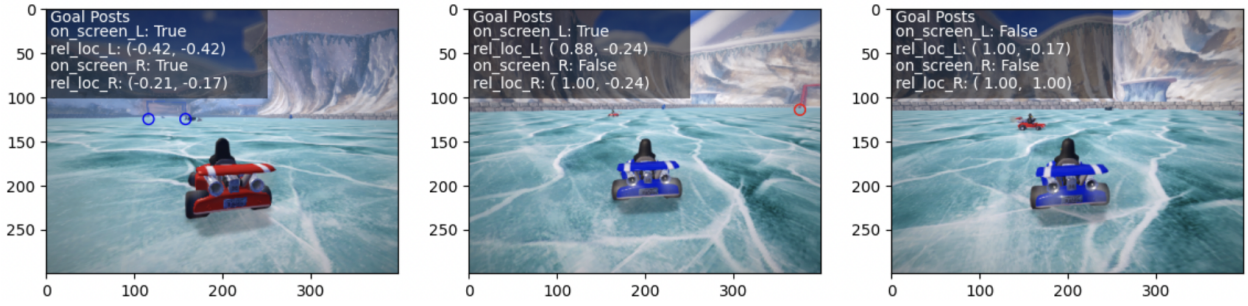

Figure K. Three examples of the auto-labeling of the indicator for left and right target goal posts on screen with relative x,y coordinates. Red and Blue circles highlight the labeled location of the posts, provided they are visible.

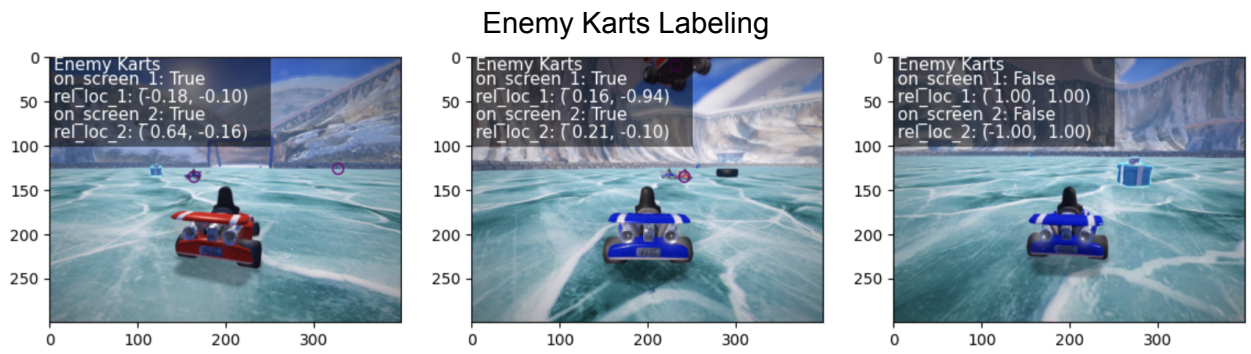

Figure L. Three examples of the auto-labeling of the indicator for enemy karts on screen with relative x,y coordinates. Purple circles highlight the labeled location of the kart(s), provided they are visible.

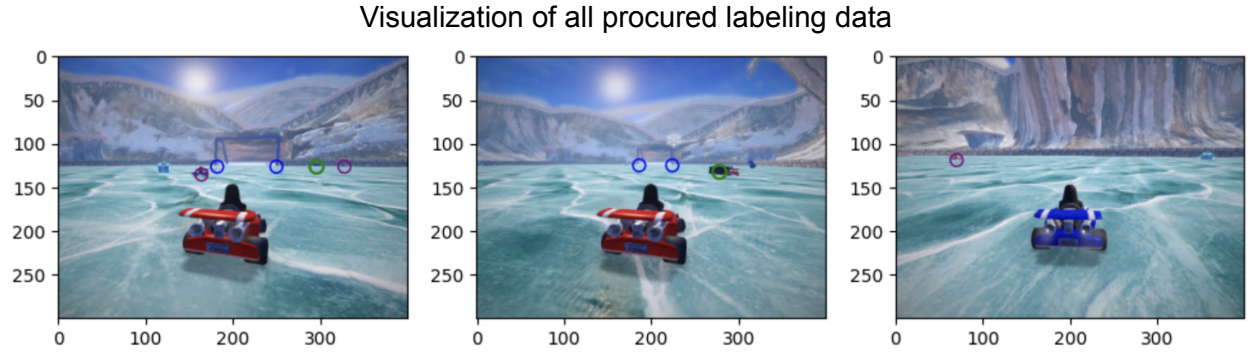

Figure M. Three examples of the auto-labeling of puck, goal posts, and enemy karts with relative x,y coordinates. Red and Blue circles highlight the labeled location of the target posts, green highlights the puck, and purple highlights enemy karts, provided each is visible.

These additional labels could be used to develop a multi-class, fully convolutional network that would provide better information to the controller. The benefit of adding enemy kart and goal post detection as outputs of a single network is the ability to create more nuanced and precise in-game actions with the additional data. The detection of goal posts, in relative coordinates, would make it easier to compute the desired trajectory of the puck. Additionally, the detection and location of enemy karts would enable easier navigation of the puck into the goal by avoiding blocking by enemy karts. However, training the multi-class FCN would require thoughtful updates to the loss function to ensure that puck detection is not negatively impacted as the model updates weights for the other object classifications.

Conclusion

We were able to successfully train a six-layer convolutional neural network (CNN) image detector that performed extremely well at puck identification and location. This network was trained on a 12,000 image data set, custom curated with automated labeling of a boolean indicator for whether the puck was within the vision field of a kart and the relative xy-coordinates of the puck. Additionally, a custom action decision controller was built that ingested the CNN outputs and converted those to in-game actions in order to navigate the puck into the enemy goal. This end-to-end AI solution was developed with the sole aim of autonomously scoring goals in a hockey match of SuperTuxKart.

The stark contrast of a grouping of black pixels (the puck) against the light colored arena and karts proved an easy task for the neural network, because only minimal hyperparameter tuning was required to achieve in-sample and out-of-sample accuracy rates above 95%. The six state action decision controller was then able to convert these detections into goals scored by prioritizing rapid movement towards the puck when visible and repositioning the kart with a larger field of view when the puck was not visible. In a final test of 16 matches across 4 different AI opponents, the solution detailed in this paper was able to score an average of 1.2 goals per game.

This work is foundational and there are avenues of exploration that could produce better outcomes, as detailed in the Future Improvements section above. Contrary to our initial hypothesis that the puck detection would be the more difficult task, developing the controller logic actually consumed a vast majority of effort. Therefore, these ideas focus on the key challenge identified in this work: the complexity required to create an effective action decision controller.

Citations

[1] “Computing the Pixel Coordinates of a 3D Point”, [Online]. Available: https://www.scratchapixel.com/lessons/3d-basic-rendering/computing-pixel-coordinates-of-3d-point/mathematics-computing-2d-coordinates-of-3d-points.html

[2] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. (2021) Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53. https://doi.org/10.1186/s40537-021-00444-8

[3] Olaf Ronneberge, Philipp Fischer, Thomas Brox (2017).U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI, arXiv:1505.04597

[4] Yang Yue and others (2023), Improving therapeutic synergy score predictions with adverse effects using multi-task heterogeneous network learning, Briefings in Bioinformatics, Volume 24, Issue 1, bbac564, https://doi.org/10.1093/bib/bbac564

[5] Diederik P. Kingma, & Jimmy Ba. (2017). Adam: A Method for Stochastic Optimization. 3rd International Conference for Learning Representations, arXiv:1412.6980