Phillip Boettcher

CS 455

Final Project

12/7/24

Abstract:

This project focuses on developing a Python bot capable of autonomously navigating Mario Kart Wii using real-time machine vision techniques. By using the YOLOv8 instance segmentation model, the bot identifies and classifies objects from a continuously updated video feed. The dataset used for training was created with Roboflow, featuring manually annotated gameplay screenshots, achieving a model precision of 98.1%.

To provide the bot with a real-time visual input pipeline, the Windows API was utilized for efficient screenshot capture, allowing for a higher frame rate compared to alternative methods. The use of OpenCV allowed for visualization of YOLO's outputs. This algorithm, while simple, used positional data to calculate steering adjustments and maintain the bot's trajectory on the road.

The system achieved an average processing speed of 10 frames per second on a CPU, demonstrating functional object detection and control while being limited by computational constraints. While the project was tested in a simplified environment, it successfully showcased the feasibility of combining object detection and algorithm-based decision-making to automate gameplay.

The results provide a strong foundation for future improvements, including expanded datasets for multiple courses, and the incorporation of reinforcement learning for more adaptive and strategic bot behavior. This project highlights the potential of machine learning and computer vision in developing systems for interactive and visually dynamic environments.

Introduction:

This project aims to create a bot capable of playing Mario Kart Wii autonomously. Mario Kart Wii is a racing game that provides a suitable environment to exercise and challenge one's machine vision and machine learning techniques. The main objective for this bot is to perceive its surroundings to control the player, emulating decisions and actions that a human player may make.

The machine learning model used for this project comes from the collection of YOLO (You Only Look Once) models. These models are used for real-time object detection and can classify objects using only a single pass of an image, hence the name. YOLO can provide users with where and what an object of interest is. Using a neural network, YOLO can detect multiple items simultaneously. These are some of the fastest and most accurate models for real-time applications.

The YOLO algorithm uses a single Convolutional Neural Network (CNN) that divides an image into a grid. Rather than using a sliding window or alternative methods, each cell predicts bounding boxes, confidence scores, and class probabilities. YOLO processes the entire image through one pass of the CNN. This allows for faster computing time in comparison to traditional approaches that may use multiple passes. YOLO then uses Non-Max Suppression (NMS) to reduce the number of bounding boxes by eliminating the boxes that have a high overlap with a box that has the highest confidence score.

The OpenCV Python module is used to give a proper visualization of what the YOLO model is accomplishing. Using OpenCV to draw the results of the YOLO model allows the user to truly understand what the algorithm is detecting. This is very helpful during debugging.

To bridge the gap between the YOLO algorithm and the gameplay execution, this project implements a relatively simple algorithm to detect positional data provided by the model and execute inputs accordingly. By using instance segmentation, the bot achieves accurate object detection for curved objects, like the road.

Methods:

The Roboflow website provides a relatively simple way to collect an effective dataset for training object detection models. Uploading images of gameplay creates the basis of the dataset. From there, objects of interest were manually annotated by drawing polygon segmentations around the objects of interest. The three classes that define these objects are the road, offroad, and player. These annotations are just visual representations of JSON data that describes where the objects of interest are within the image. This dataset now lives within Roboflow's servers and can be used to train several different models. Roboflow also provides resources, such as extensive documentation and Jupyter Notebook files, to train a created dataset. Using their python module, the specific dataset that was created can be imported for training a model.

The specific YOLO that was used for this project is YOLOv8's instance segmentation model. Instance segmentation is generally more resource-heavy than using a standard bounding box. This is because the information for a bounding box can be described with four coordinates, while a segmentation is described by multiple vertices coordinates. Instance segmentation was necessary due to some of our objects of interest having curvature to them, specifically the road and offroad objects. A standard bounding box encompassing a concave or convex object will also encompass a significant amount of area that is not our object of interest. Using instance segmentation provides a better description of the space these curved objects truly hold.

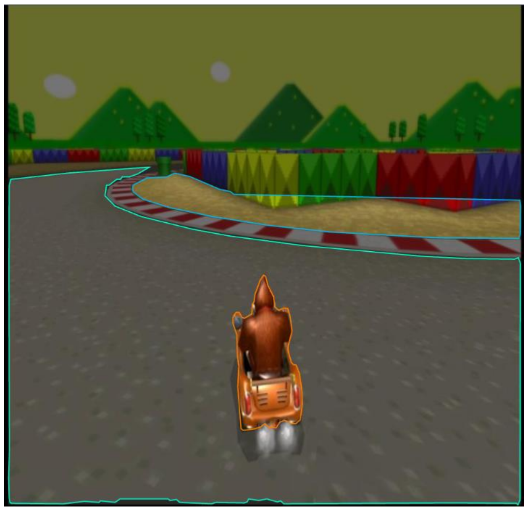
Figure 1: Instance Segmentation from Mario Kart Wii using Roboflow

Providing a video feed to the YOLO model is accomplished by taking rapid screenshots of the game window. The common Python module PyAutoGUI can take a screenshot of a window, and then the screenshot can be displayed using OpenCV. An inifinite while loop is used to take a screenshot of the desired window and display said screenshot. This will essentially provide a video feed where the frame rate is dependent on how fast the machine is able to iterate through the loop. A high frame rate is valuable when trying to achieve real-time object detection. PyAutoGUI uses the Windows API for its screenshot functionality. This means that the Windows API can be used directly to take screenshots instead of using PyAutoGUI as an intermediator. This change can yield a significant increase in the speed of loop iterations.

All the machine vision related processes are handled together within a class. Ultralytics provides a YOLO module for Python. By using this module, an image can be passed through the YOLOv8 algorithm to have the objects of interest be classified using weights generated when the relevant dataset was created. By using the same loop used to take rapid screenshots, each screenshot is also passed through the YOLO algorithm. The algorithm then will classify each object it can determine within a specified confidence range. This can be visualized by pulling the segmentation coordinates from the results of the image and the model, and using OpenCV to draw the contours of the segmentation masks. The YOLO model and OpenCV now accumulate to a workflow where the YOLO model can provide relevant positional data of the detected objects, and OpenCV provide a visualization for debugging purposes.

The positional data collected through the YOLO model can be used to determine the actions the bot will take. For the scope of this project, a simple reactionary algorithm is implemented. Using OpenCV, two points are created on the road. One point in placed on the geometric centroid of the road, and another dot is placed at the northmost end of the road centered in the x-axis. Finding the slope between these two points gives an estimate of how the road curves further down the path. Comparing the position of the geometric centroid of the road with the geometric centroid of the player also gives an idea of how far the player is from the center of the road. Pynput is used to produce keyboard inputs depending on all the positional data previously mentioned. This allows the bot to react based on the visual information present.

Results:

Manually using the Windows API to take screenshots of the game window rather than using Pynput gave an increase in roughly 20 frames per second on my machine, making it a worthwhile strategy to implement. The main bottleneck on the speed YOLO model's inference time. Without using my GPU, I was able to achieve approximately 10 frames per second after passing each screenshot through the model and using OpenCV to draw what the model detected.

The YOLOv8 instance segmentation model, trained using a dataset created with Roboflow, achieved an impressive precision of 98.1%. The decision to focus on a single Mario Kart course and limit the scope to time trial mode allowed for faster training and an easier testing environment. The use of instance segmentation provided accurate representations of curved objects such as roads and offroad regions, enhancing the ease of creating an algorithm for spatial-based decision making.

OpenCV was used to display segmentation masks on the video feed. This was able to give a very clear visualization of the YOLO model's predictions. OpenCV also allowed a way to see the points of interest regarding the bot's decision-making algorithm. Without being able to see what was going on, the trial and error it took to make the algorithm would have been much more extensive and time-consuming. These visual cues provided by OpenCV were invaluable.

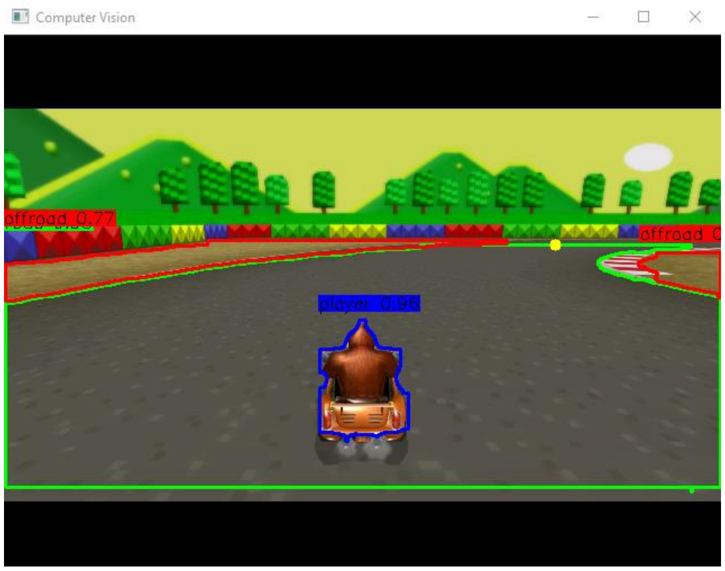

Figure 2: Drawing YOLO Instance Segmentation with OpenCV

Discussion/Conclusion:

Despite this project technically being a success, there are some challenges that arose. The system was limited to my machine's CPU. While my CPU does have respectable computing power, being able to utilize a GPU would likely yield a significant increase in performance. Mario Kart Wii runs natively at 60 frames per second, so having the speed of the bot run as close as possible to this would allow a more accurate rection time when it comes to steering and driving the bot.

This project was implemented using the simplest environment I could imagine for Mario Kart Wii. It was done on what I deemed the most visually simple track. It was also done in time-trial mode, which removes the opposing racers and items. Given more time for this project, I think it would be a straightforward process to scale it up to include multiple course environments and introduce other racers and items.

The algorithm used to control how the bot steers is too simple. This is also another aspect that could be improved. A reinforcement learning algorithm would likely yield a significant improvement in the objective performance of the racing capabilities of the bot. The current algorithm is only able to react to the positions of the objects it can see. A reinforcement learning algorithm would theoretically be able to have a simple reward system depending on things like lap completion time, speed of the bot, and whether the bot is staying on the road or veers into the offroad.

Overall, I am happy with the outcome of this project. While it may not be the best bot one could create, I believe this is a good starting point. I think this project provided a strong baseline to improve upon. Providing a larger and more varied dataset and implementing some smarter decision-making algorithms are just a couple ways that this could become more robust.

Works Cited:

LearnCodeByGaming. “How to Use Machine Learning to Train Bots to Play Video Games.” *YouTube*, 1 Feb. 2023, www.youtube.com/watch?v=WymCpVUPWQ4&ab_channel=LearnCodeByGaming.

Ultralytics. “Train Custom Data.” *GitHub*, https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data.

Roboflow. “Train YOLOv8 Instance Segmentation on Custom Dataset.” *GitHub*, github.com/roboflow/notebooks/blob/main/notebooks/train-yolov8-instance-segmentation-on-custom-dataset.ipynb.

Pysource. “Instance Segmentation YOLO v8 + OpenCV with Python Tutorial.” *Pysource*, 21 Feb. 2023, pysource.com/2023/02/21/instance-segmentation-yolo-v8-opencv-with-python-tutorial/.

OpenCV Documentation. “Adding Images Using OpenCV.” *OpenCV Documentation*, docs.opencv.org/4.x/d5/dc4/tutorial_adding_images.html.

OpenCV Documentation. “Contours in OpenCV.” *OpenCV Documentation*, docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

Roboflow. “MK Wii Objects Dataset.” *Roboflow*, app.roboflow.com/mkwii/mkwii-objects/6.