Machine Learning in Minecraft: Proof of Concept for Object Detection Oriented Autonomous Bots in Minecraft

John Merkin

KENNESAW STATE UNIVERSITY
COLLEGE OF SCIENCE AND MATHEMATICS

Machine Learning in Minecraft

Proof of Concept for Object Detection Oriented Autonomous Bots in Minecraft

John Merkin

Introduction

Machine learning provides innumerable opportunities for automation and data analysis. One primary objective for machine learning is object detection; object detection seeks to identify the location of objects in an image as well as determine the most probable class of each object. An Interesting challenge is to implement various machine learning models into the context of video games and other virtual environments.

Minecraft, an open-world sandbox game, gives players the freedom to collect resources and alter the environment as they choose. In Minecraft's survival mode, objectives and resources must be collected according to a built-in hierarchy. For example, to collect stone, a player must first collect wood to craft a pic-axe. As such, this project seeks to automate the initial objective of finding and collecting wood in a Minecraft virtual environment through implementation of a machine learning model.

Neural Network Architecture

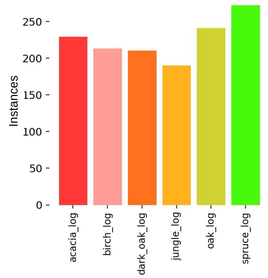
This project utilizes the Ultralytics Yolov8-medium pipeline architecture, a convolutional neural network (CNN) framework pretrained on Microsoft's COCO dataset. Yolov8 is designed to predict the location and classification of objects in images with bounding boxes.

The Yolov8 backbone consists of several interwoven convolutional and pooling layers. Between layers Yolov8 utilizes mosaic augmentation, segmenting images into parts and attaching them to other image segments. Mosaic augmentation further diversifies the dataset to better generalize and stops near the end of training to improve performance (Solowetz).

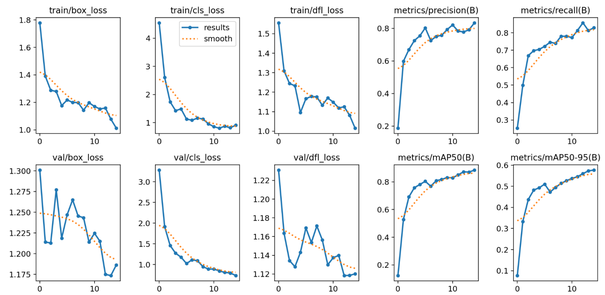
Although pre-trained models are not guaranteed to improve accuracy over initially untrained models, pre-training typically yields improved results over less iterations for smaller datasets. Pre-trained models have also been shown to have improved robustness and generalization to new data (Hendrycks et al).

Data and Methodology

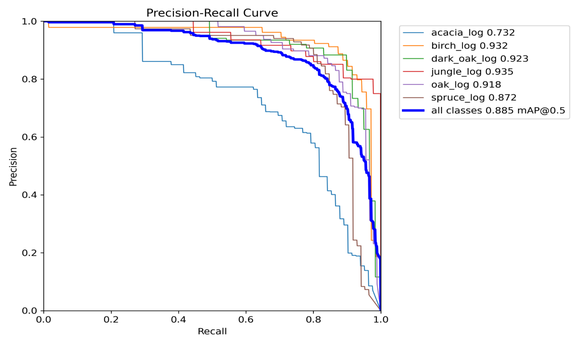
Training data was collected via sampling image frames from videos of Minecraft gameplay recorded with OBS. Images were resized into a 512x288 pixel resolution. 510 unique images were sampled and manually labeled with bounding boxes representing 6 classes of Minecraft trees. Images were then stretched into a 640x640 resolution and exported using Roboflow; 351 images were used for training, 105 for validation, and 54 for testing.



Training Metrics and Results



Running on an Intel i5 12600KF CPU, the network took 1 hour and 16 minutes to train over 15 epochs with stochastic gradient descent, reaching an overall average precision of 88.5% at a recall threshold of 50%. Specific precision measures for each class are shown below.



Application

Using *JSPyBridge* to access JavaScript functionality within a Python script, the program uses the npm package *Mineflayer* to access API for creating a Minecraft bot. A local server hosting a Minecraft world provided a virtual environment in which the bot can interact. Using a *Mineflayer* plugin, the bot's first-person view is displayed in an external window with resolution 1296x853.

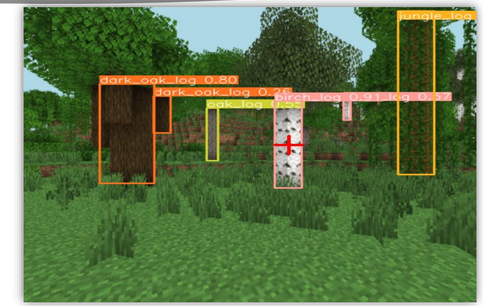
A screenshot is taken from that window using the *Selenium* library with resolution 1280x720. The *open-cv* library is used to downscale the image into a 512x288 image and then stretch it into a 640x640 image. The trained neural network returns the 640x640 image with added bounding boxes and confidence values for each predicted class instance. For each bounding box of a given class, the center of each box (x_c, y_c) is determined and used to generate horizontal (θ) and vertical (φ) reference angles as follows:

$$x_r = \frac{320 - x_c}{320}, \quad y_r = \frac{320 - y_c}{320},$$

$$\theta = x_r * \left((-22.0302)x_r^2 + 74.5912\right) * \frac{\pi}{180},$$

$$\varphi = 35y_r * \frac{\pi}{180}.$$

The 2nd degree polynomial term in θ was approximated using Lagrange Interpolation to account for first-person image distortion and is specific to the size of the external viewing window. The bot then rotates by θ and φ and logs the coordinates of the object. After locating each object in the frame, a pathfinding algorithm brings the bot to the object; it then faces the tree and collects wood. This process repeats for each desired object in the frame.



Conclusion

As shown above, the model excels at generating bounding boxes and gives mostly accurate classifications. The model occasionally struggles with classification of acacia trees outside of their original biome, as seen in the image above. Further training the model with mosaic augmentation or expanding the dataset would likely improve cross-biome classification and robustness for objects with similar textures like oak and acacia trees.

Using *Mineflayer*, implementation of neural networks offers much flexibility. Hard-coded behavior, such as the collecting, moving, building, and other behaviors of the bot can be altered to a programmer's desire. A new dataset on the exterior of caves could be created and used to train a new network to identity caves; a dataset on ores such as coal, iron, and diamonds can be used to train a bot for mining. A state-machine could be used to switch between neural networks and behaviors to carry out more complex tasks with desirable autonomy.

REFERENCES

1. Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).
2. Hendrycks, D., Lee, K. and Mazeika, M., 2019, May. Using pre-training can improve model robustness and uncertainty. In *International conference on machine learning* (pp. 2712-2721). PMLR.
3. Solawetz, J., 2023. What is Yolov8? The Ultimate Guide. *Roboflow Blog,* blog.roboflow.com/whats-new-in-yolov8/.
4. Andrew Kelley, Mineflayer, (MIT 2015), github.com/PrismarineJS/mineflayer
5. PrismarineJS, primsarine-viewer (MIT 2020), github.com/PrismarineJS/mineflayer/.
6. Yolov8, Free Software Foundation Inc., (GNU AFFERO GEN PUBLIC 2007), github.com/ultralytics/ultralytics/.

Special Thanks to Dr. Min Wang, the InspireSummer2023 Program, Casey Hampson, Nicholi Moore Caron, and Jonathan Holloway