OSRSNet: Real-time Object Recognition in 3D MMORPGS

Matt Rickard
Stanford Graduate School of Business
Stanford University
March 16, 2021

Abstract

Virtual worlds are becoming increasingly important as individuals spend more time online behind virtual avatars. These environments have been training grounds for deep learning applications like self-driving algorithms, but there have not been many attempts to apply techniques from the real world inside dynamic mass multiplayer virtual worlds. We utilize the state-of-the-art online object recognition deep learning architecture yolov5 to classify over 800 objects from the mass multiplayer online game RuneScape. We achieved a mAP (0.5) score of 0.78 after training on 25,000 640x640 in-game screenshots.

1 Introduction

Object recognition models have been developed for all sorts of online tasks – for example, they are used in applications from self-driving vehicles. Virtual worlds are becoming increasingly interesting – in terms of breadth (number of users, world size) and depth (possibility of actions and impressiveness). They can be training grounds for understanding and modelling human behavior in a reproducible and scalable manner. Additionally, games like RuneScape provide a medium fidelity experience of our world – vaguely human-like avatars, monsters, and items, but coupled with a tiling grid system and limited movement and actions in the virtual world.

We propose building a real-time object recognition model for the mass multiplayer online role-playing game (MMORPG), RuneScape.

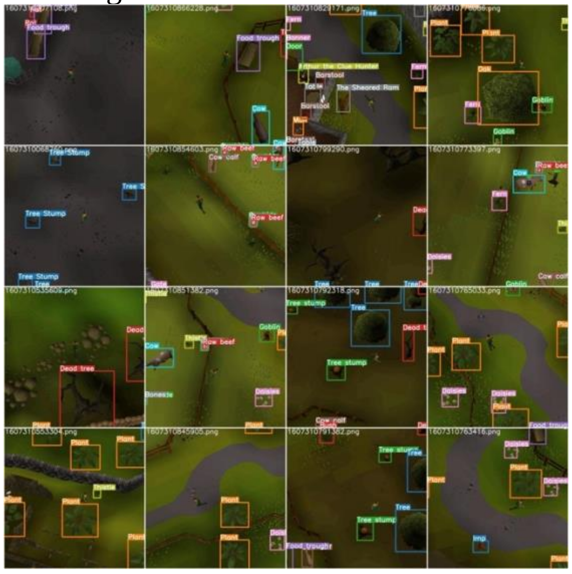
2 Related Work

The closest related work is Kim, et al., who applied the YOLO v5 algorithm to identify 2D sprites in retro video games. They achieved a mAP 0.5 of 0.9635 with a combination of 60,000 images with both real and synthetic data. However, this problem is constrained to much lower pixel densities and 2d objects, rather than the 3d virtual world we plan to train our model in. Additionally, our world contains numerous player characters, which may look similar, but are different, from in-game NPCs and objects. Finally, we are looking at a large number of classes, training on 800+ distinct classes.

Deep learning has been applied to building non-player character (NPC) AI behaviour in MMORPGs. Pfau, et. al found that models of individual game player behavior could be generated with high accuracy, especially in creating strategies. Other deep learning applications include bot detection using LTSM and time windows. Tsikerdekis found that LTSMs were effective in detecting automated player behavior in MMORPGs. Fujita found that real world traders, a complicated problem for in-game MMORPG economies, could be detected with deep learning. For game developers, Ling found that CNNs could be used to detect rendering issues in video games in real time.

3 Dataset

We built an automated bot to collect over 25,000 in-game images. We used a free-to-play (f2p) account, mapping out a walking path that covered nearly all accessible areas to new players. Since the game is a 3d virtual environment, we augmented the dataset at collection time by taking multiple images from different camera pitches and yaws. Since the data was collected in a live environment, NPCs and game objects are constantly in motion, with additional noise from unlabelled player characters. Since the game is java-applet-based, we were able to generate accurate bounding box labels for over 800 classes of NPCs and game objects through decompiling the game client and using Java reflection.


A typical batch of labelled data

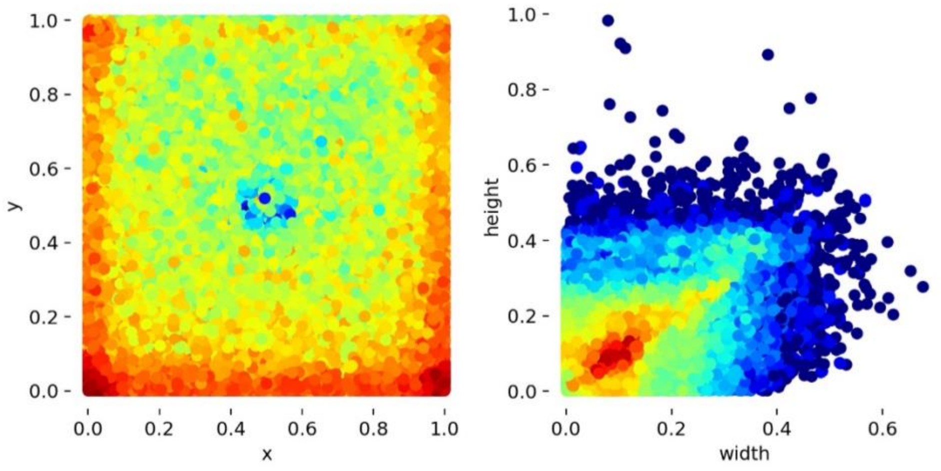


Image label dimensions across all classes

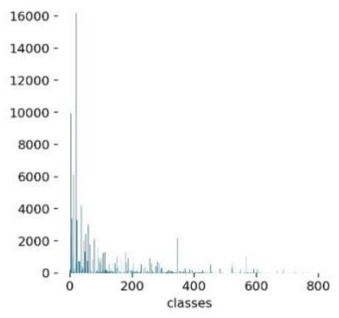


Image counts per class

4 Methods

We used a YOLO v5 algorithm, which is a recent rewrite of the *You Only Look Once* algorithm in pure PyTorch. From the original paper, the architecture of the network:

5 Experiments/Results/Discussion

5.1 Base Model

We used the yolov5 architecture as an initial baseline model to train upon. We modified hyperparameters and made simple optimizations – leaving the overall architecture in place.

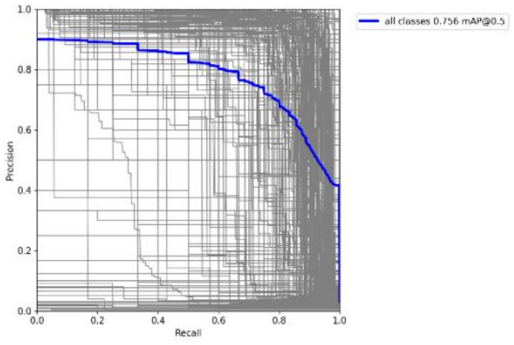
We trained 3 separate models to get a baseline – first, with a small subset of classes, second, with a specific subset of classes, and finally, with all 800+

classes. We experimented with different images sizes and will continue to tune the other hyperparameters as we go forward.

5.2 Hyper Parameters

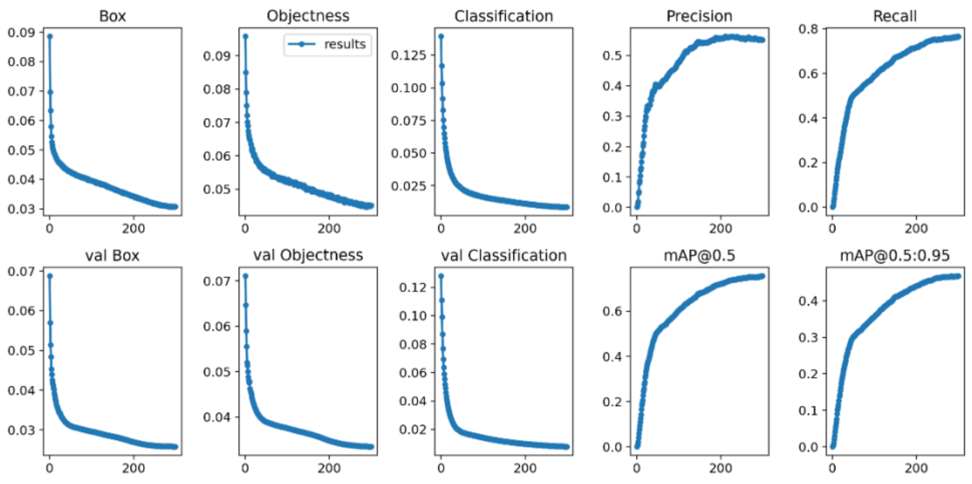
For our hyperparameters – we used a batch size of 16, an image size of 640x640, and trained for 300 epochs on our personal hardware (NVIDIA Titan X GPU, 64gb RAM). We augmented the data with translation, scale, and horizontal flips. We used default values for momentum and learning rates, but we plan to experiment with tuning these hyperparameters later.

5.3 Results



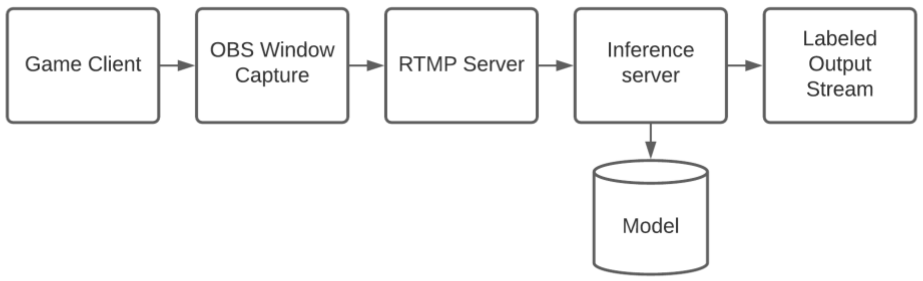
We judged our results based on the mean average precision (mAP) of the neural network. We can see that there are many classes for which precision and recall is both very high, and some classes that the model had difficulty with. Since our classes and data were generated programmatically, we might need to dig deeper into these classes to understand if there is a labelling issue, or even a miscategorization by way of overlapping classes.

Here is the output of result after training for 300 epochs.



6 Online Inference Architecture

Online inference is important for our ultimate application. The ability to do inference at 30fps would satisfy our use case. To do so, we have an architecture where we capture the game window with Open Broadcaster Software (OBS) streaming that to a nginx RTMP gateway, which then multiplexes the stream to both the python inference server as well as a separate source for debugging or streaming live to the player.



7 Conclusion

The algorithm performed well enough to serve as a foundation for higher-order actions in the virtual world. Additionally, we were able to get the online inference working at a desirable framerate, making the application useful for using in real-time.

There is considerable work to be done to refine the model and achieve better results. First, we could collect more data. We collected data with a single account over the course of a few days, but could horizontally scale the operation by running more game clients in parallel. Second, we could experiment with synthetic data combined with real data. This method was effective for Kim, et. al for 2d sprites, and the strategy might generalize to 3d models.

8 References

Fujita, Atsushi, Hiroshi Itsuki, and Hitoshi Matsubara. "Detecting real money traders in MMORPG by using trading network." *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 6. No. 1. 2011.

Kim, Chanha, Jaden Kim, and Joseph C. Osborn. "Synthesizing Retro Game Screenshot Datasets for Sprite Detection." (2020).

Ling, Carlos, Konrad Tollmar, and Linus Gisslén. "Using Deep Convolutional Neural Networks to Detect Rendered Glitches in Video Games." *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 16. No. 1. 2020.

Pfau, Johannes, Jan David Smeddinck, and Rainer Malaka. "Towards deep player behavior models in mmorpgs." *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*. 2018.

Tsikerdekis, Michail, et al. "Efficient Deep Learning Bot Detection in Games Using Time Windows and Long Short-Term Memory (LSTM)." *IEEE Access* 8 (2020): 195763-195771.