Machine Learning and Object Detection in Gaming

Team members' names

Ryan McDonald

Alexander Frederick

Problem Statement

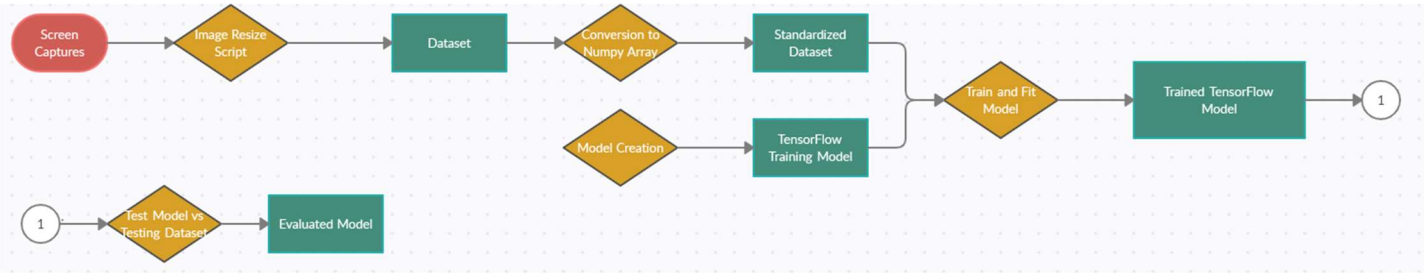
In the MMORPG Old School RuneScape, players complete repetitive tasks in order to level their skills within the game. We are attempting to automate these tasks using machine learning to recognize objects within the game without interacting with any of the game's files that are running on the user's machine. This is to minimize the chances of being detected by any anti-cheating software. The task we were looking to automate was leveling the player's mining skill by mining an ore and dropping it repeatedly in order to maximize the player's experience within the game.

Approach

In order to solve the issue of detecting the ore within the game, we first begin by creating a dataset of images from within the game that showcase the ore from various angles. We use this dataset with the TensorFlow framework in order to create a model that can recognize the ore within an image.

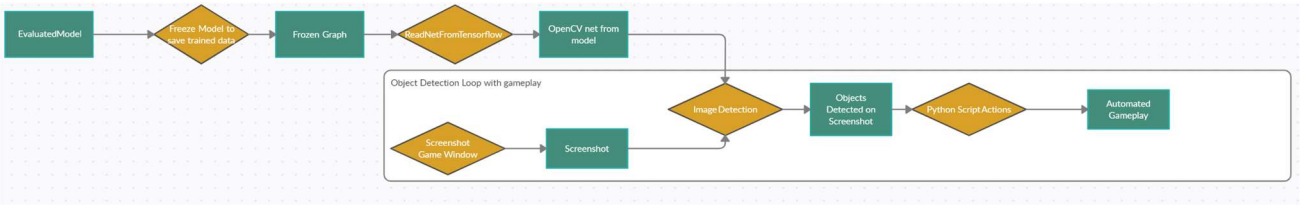
We want to now use this model with the OpenCV library in order to recognize multiple instances of the object from within a single image. We intended to capture images from within the game window using a screen capture using the PyAutoGUI library and OpenCV library together. These screenshots would be performed within a loop that repeats 100 times per second to create a video like input stream for OpenCV to recognize the ore within the game in real time. From here we would just use scripting within Python to move the cursor and click on the screen in order to perform actions such as mining the ore and dropping the ore.

Description of the software



The first component of the software used was an image resizing script created in Python using Pillow and Python's os module in order to standardize our dataset so that it could be loaded into a TensorFlow

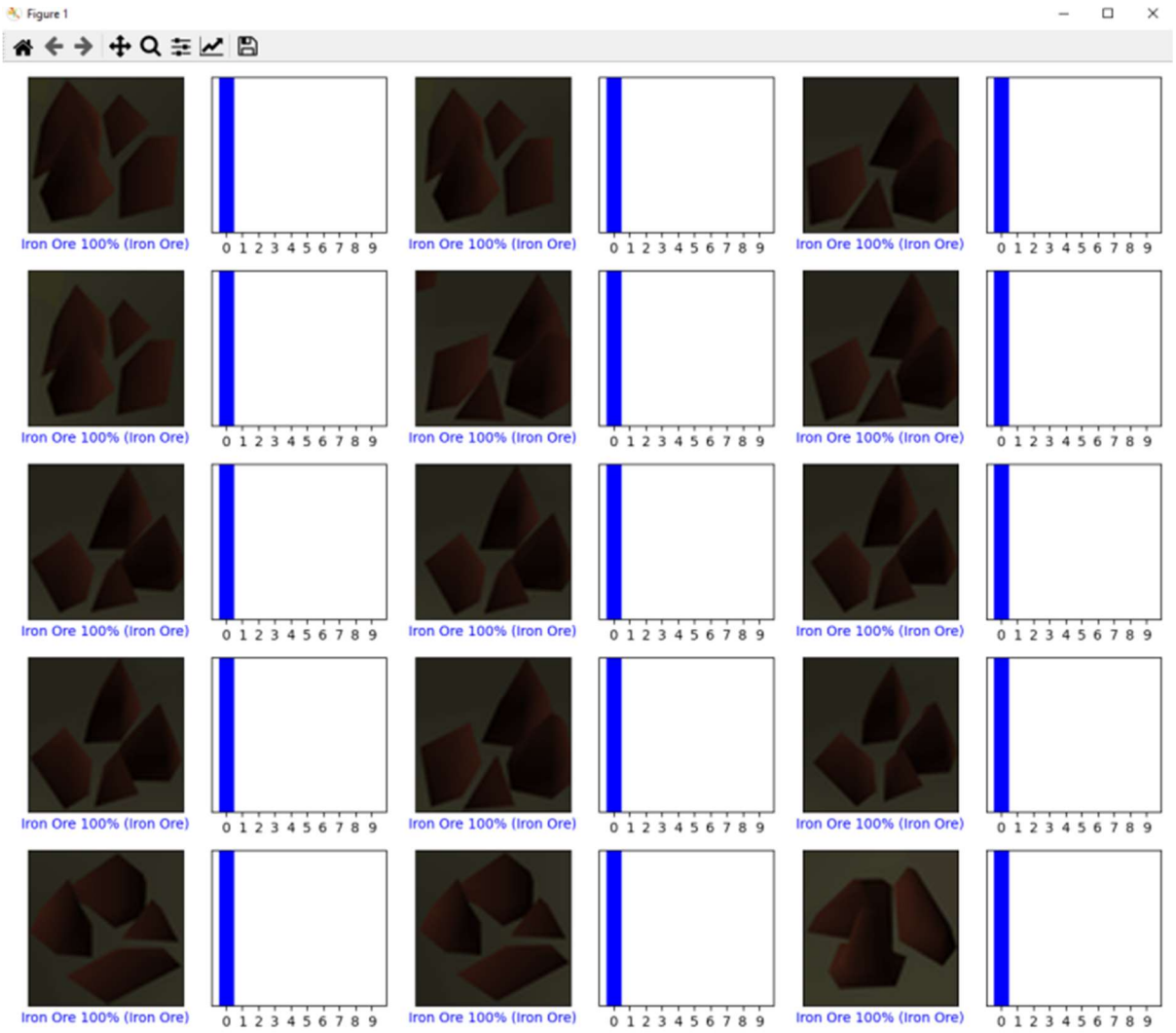
model. Once the dataset has been created, we convert it from Python's default list into a NumPy Array so that it can loaded into our model. After standardizing our dataset and preparing it for the model we create a TensorFlow Sequential Keras model. The Sequential model allows each layer to have only one input and output tensor from the previous layer and to the next layer respectively. After the layers have been added the model and the model is compiled, it is ready to be trained with our dataset. Once the model has been trained it is ready to recognize objects within an image using the dataset that it has been trained with. We then evaluate the model using a set of testing images in that contain the object that it has been trained for in order to see if it is able to recognize it.



Once the model has been seen to accurately detect the ore we are looking for, we freeze the graph so that it saves the trained information and makes it so the model's variables cannot be modified and no additional training can be done. We would then load our graph into OpenCV and set the input to a screenshot of the game window. OpenCV would detect our objects within the image using the trained model and draw rectangles around the object's location on the screen. From here we would automate the moving of the player's mouse, clicking on the ore and dropping the ore from the player's inventory. This automation of player action and the input image being set would be done on a loop to create a real-time automation and detection system to play the game.

Evaluation

Because of our inability to load our TensorFlow frozen graph into OpenCV we were unable to complete the full automation of gameplay as we had originally intended. Due to a low-level issue we had to adjust our evaluation parameters in order to test our model within TensorFlow. While we had originally intended to evaluate the project based on the experience rates it was earning within the game, we are now evaluating it based on its ability to detect objects within the TensorFlow framework. At this point we are taking our trained model and feeding it a set of testing images to see if it is able to correctly identify them as the ore we are looking for. When creating a prediction model with our trained model and testing set we can see that our model is accurately detecting and labeling the testing images as seen below.



Conclusion and Future Work

The main lesson we learned from this project is that there is a plethora of errors that can occur when attempting to combine and utilize multiple open source libraries in using to perform different tasks within a project. Although these libraries may be written to work in unison there is still a lot of issues that can arise that are not always well documented.

When looking into using these two tools it seemed as though the majority of the tutorials were using either a pretrained model or a precompiled data set in order to train the model. This led to us having to experiment with making our own custom model with a unique data set in order to fit our problem. We learned about the process for creating a deep neural network and machine learning model through TensorFlow. When learning about the model we had to resolve the issue of loading and compiling our own set of data from custom images in order to accurately train and test the model.

For the future we plan to continue to attempt to resolve the errors that are occurring when loading the frozen graph from TensorFlow into OpenCV for object detection within an image. Once this

error is resolved we plan to move forward with the project as originally intended. Once we have it working for just the Iron Ore, we plan to add more classes to be able to detect all of the different ore within the game.

After that is done one thing we wanted to look into for further improvement on the project is potentially using machine learning with mouse movement in order to create human like mouse movement for the in game cursor in order to even further minimize the risk of being detected by an anti-cheat software and appear more human like.

References

TensorFlow 2 References Used

- TensorFlow2 Model & Predictions: https://www.tensorflow.org/tutorials/keras/classification
- Freezing TensorFlow2 Model: https://medium.com/@sebastingarcaacosta/how-to-export-a-tensorflow-2-x-keras-model-to-a-frozen-and-optimized-graph-39740846d9eb

TensorFlow 1 References Used

- Saved Model TF1: https://cv-tricks.com/tensorflow-tutorial/save-restore-tensorflow-models-quick-complete-tutorial/
- Attempt to solve getConstBlob error: https://github.com/opencv/opencv/issues/11542
- Getting output node names: https://stackoverflow.com/questions/40028175/how-do-you-get-the-name-of-the-tensorflow-output-nodes-in-a-keras-model
- Freeze Graph using session: https://github.com/mrharicot/monodepth/issues/181
- TF1 Model References: https://faroit.com/keras-docs/2.0.2/getting-started/sequential-model-guide/

OpenCV References Used

- Attempt to solve getConstBlob error: https://github.com/opencv/opencv/issues/11542

Anaconda Environment Versions and Libraries

TensorFlow 1 Attempt:

MatPlotLib 3.3.2

NumPy 1.19.2

TensorFlow 1.15.0

TensorFlow 2 Attempt:

TensorFlow 2.1

NumPy 1.19.2

MatPlotLib 3.3.2

OpenCV 3.4.2