Predicting Block Breaker Game Wins

Zachary Christensen
University of Missouri-St. Louis

May 6, 2022

Contents

Abstract

A block breaker game in the style of the classic game Breakout was simulated using an automated player. An example of the game in action can be found here. Because the game is simulated to represent level difficulty to a human player, the auto player is tuned to a given accuracy, generally between 30% and 40%. Using the data generated from thousands of complete simulations, models are created to predict whether a game was won or lost. Models of different types and hyperparameter configurations are trained and analyzed using various performance metrics. Of these models, a set of hyperparameters is found to create the model that best generalizes the problem. Once these hyperparameters are found, then the input features are tested to find which are the most important and what combination to use for the best model. The tests in this report found that autoplayer accuracy was somewhat predictive of a win or loss. This is relevant because player engagement increases when maintaining skilled gameplay leads to a win.

Discovering a best fit model is useful because once an accurate model is found, it can be used to evaluate simulations of new levels. If a new level is not predictable like the levels used to create the model, then it can be concluded that the new level changed the original game flow.

This game is fairly simple with only three dynamic pieces, the bricks, the ball, and the player. The ball always reacts to gravity and receives a velocity boost on collision with other objects. The player moves left and right to reflect the ball at the given accuracy, and the bricks reflect the ball and sometimes are destroyed by this. The simplicity of the environment lead to results of high accuracy when training the models.

1 Introduction

What are the rules for the game?

The game environment is surrounded by a top, bottom, left, and right wall and includes a paddle, a ball, and blocks . The player controls the horizontal axis of the paddle. Not all blocks are able to be destroyed. The game starts when the player launches the ball into motion. The ball reflects from the paddle, the blocks, and the walls. When a block reflects the ball, it takes damage and may be destroyed. Once all the blocks able to be destroyed are destroyed, the game is considered won. A player loses the game if they fail to reflect the ball with the paddle.

Why is this problem interesting?

Video games can represent very complex environments. Even a few elements interacting through the physics engine can create situations that are extremely difficult to predict and control. In games that depend on dynamic elements, such as the physics engine, there is no formula to predict exactly what will happen. An example would be striking a pool ball and trying to predict all the exact reflections and end positions for every other ball. If the game can be broken down into some basic measurements and basic goals, then tests can be run to produce data. That data is something tangible that can be used to start understanding the behaviour of the

game, so it can be predicted and controlled.

Motivation

Characteristics of a good game are drama and depth. If a game has drama, then the player will have a fun, memorable experience. If the game has depth, the player will be able to get better at the game in numerous ways. These characteristics are only good when implemented in a balanced way that gives the player the most fun, memorable experience. For example, drama should engage the player but not be punishing, and depth should exist at different skill levels so it is not only engaging to a select group. To achieve this through manual playtesting is tedious if not ineffective. The motivation of this project is to learn how use statistics from gameplays to understand the elements that control the likeliness a player is be able to win a level.

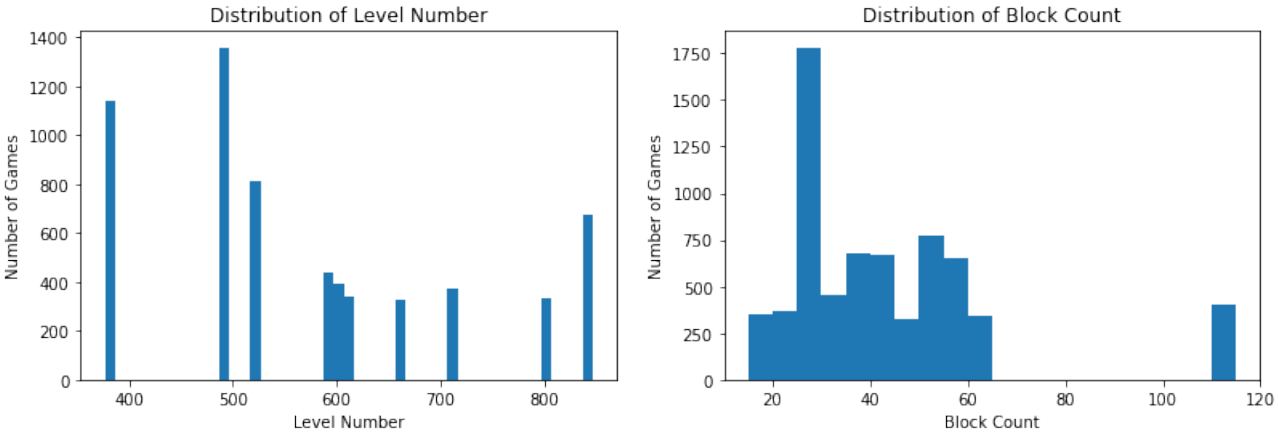
2 Dataset

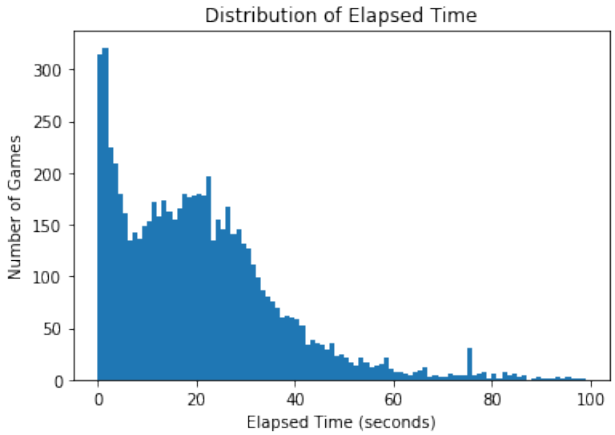
The "Arcade Game Stats" dataset was found amonst the Kaggle datasets and can be found here [1]. The data was obtained from thousands of gameplay simulations using a robot for player input. The data contains 3266 games where the robot won and 3548 games where the robot lost. To account for a human player, the robot was assigned an accuracy that sets how well the robot performs. The output of the data is 0 for a loss and 1 for a win. The input features are as follows:

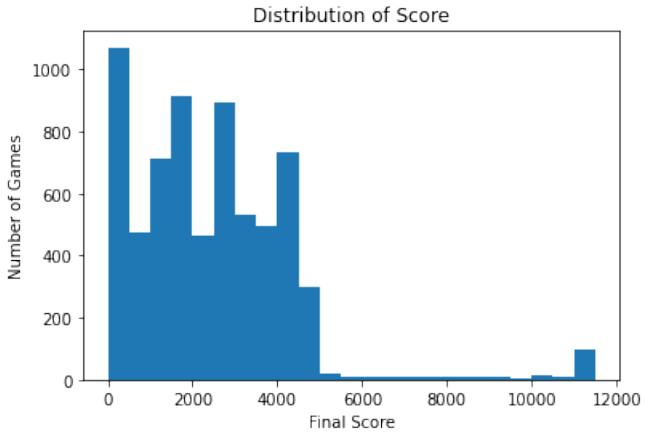
1. Level Number
2. Number of Blocks
3. Elapsed Time
4. Score
5. Accuracy

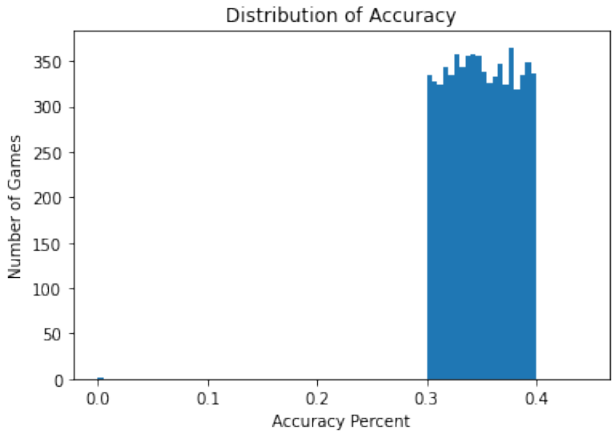
2.1 Data Visualization

Level number appears to be grouped into several clusters between about 400 and 800. Number of blocks looks like it is normally distributed. Elapsed time is skewed to the right. Score looks like it is normally distributed. Accuracy is evenly distributed within its range.





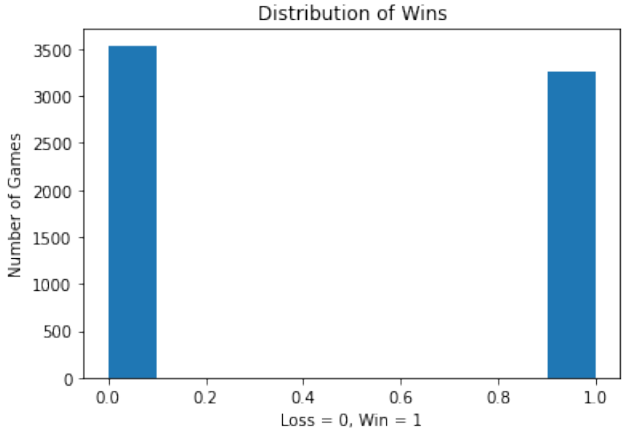




	Level	NumBlocks	ElapsedTime	Score	Accuracy	IsWin
count	6799.000000	6799.000000	6.799000e+03	6799.000000	6799.000000	6799.000000
mean	597.859979	42.430946	4.684771e+07	2386.115605	0.349926	0.479776
std	160.211059	20.815046	1.727022e+09	1918.080282	0.029051	0.499628
min	378.000000	18.000000	3.350642e-01	0.000000	0.000000	0.000000
25%	492.000000	28.000000	8.118599e+00	1150.000000	0.325401	0.000000
50%	596.000000	39.000000	1.865289e+01	2100.000000	0.349511	0.000000
75%	715.000000	50.000000	2.910713e+01	3550.000000	0.375015	1.000000
max	849.000000	112.000000	6.370349e+10	11900.000000	0.399999	1.000000

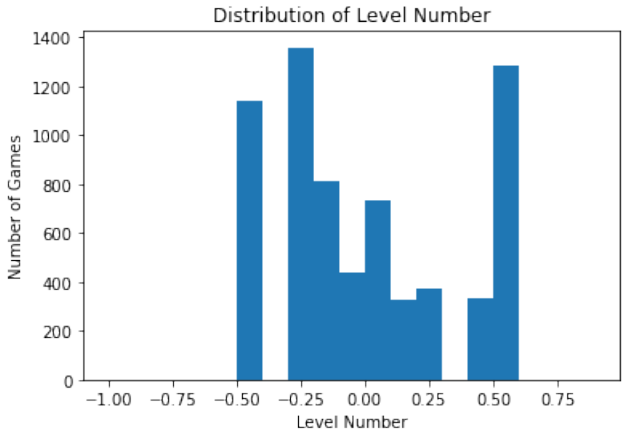
2.2 Distribution of Output

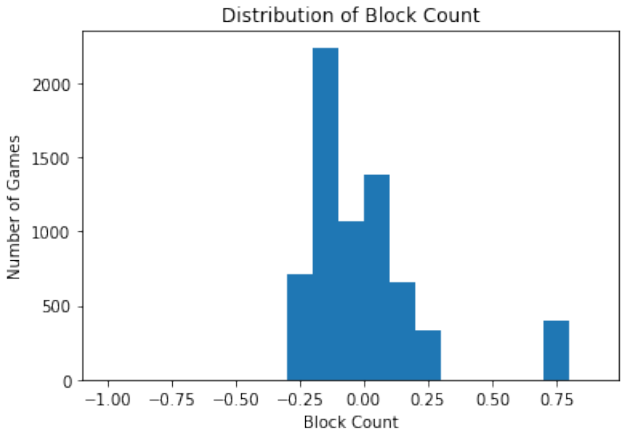
The number of wins and losses are almost perfectly balanced. There is an appropriate sample size for each sample.

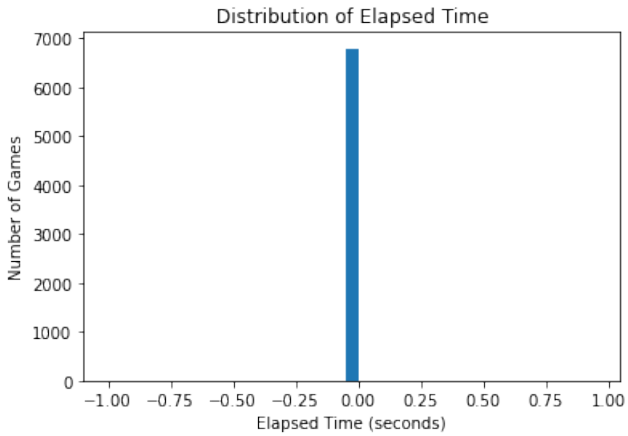


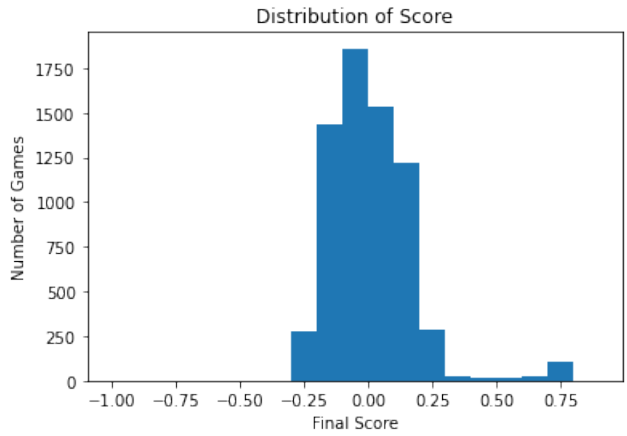
3 Data Normalization

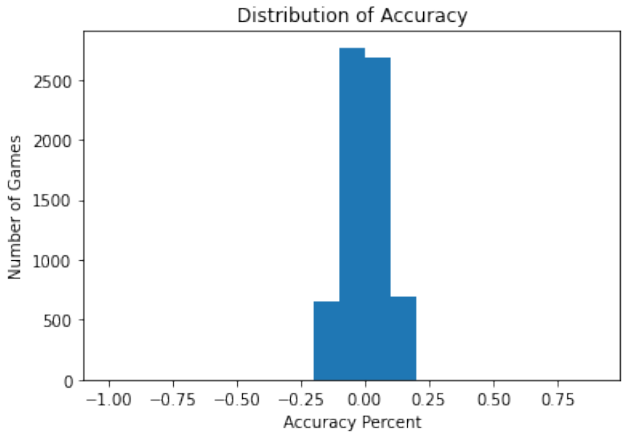
The normalization technique I chose was mean normalization. The only column I removed from the original data was the date column.











4 Data Modelling

I built a single layer model, several models of different neuron and layer count, an overfitting model, and several models where the last layer was changed to relu. The single layer model was not significant. The graphs have been removed for all but the best fit model and the overfitting model from this section. Overfitting was difficult to achieve due to most model performing well at high numbers of epochs.

4.1 Multi Layer Models

I correlated 2 and 3 layers with 32 and 128 neurons. This produced 4 different models. I ran the models for 128 epochs.

4.1.1 Model 1

This model uses 2 layers and 32 neurons.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.862524	0.316271	0.804444	0.936853	0.865615	0.925784

4.1.2 Model 2

This model uses 2 layers and 128 neurons.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.896771	0.272822	0.848673	0.95996	0.900892	0.947816

4.1.3 Model 3

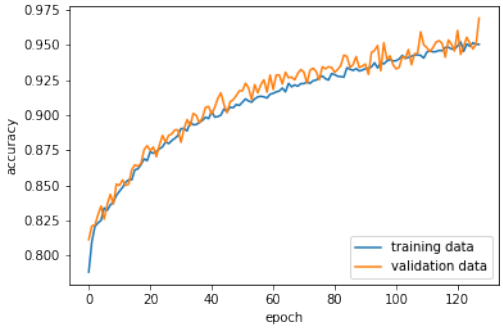
This model uses 3 layers and 32 neurons. For the second layer, I used 25% of 32, or 8 neurons. This is also true for the next model using 3 layers.

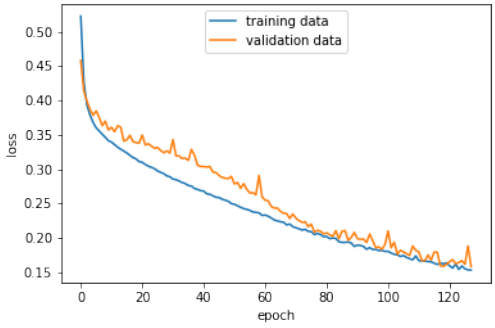
Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.908513	0.262939	0.855335	0.972251	0.910053	0.953967

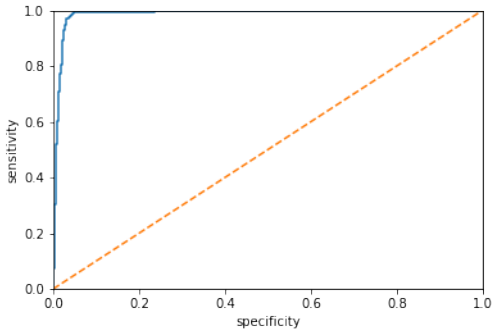
4.1.4 Model 4

This model uses 3 layers and 128 neurons. For the second layer, I used 32 neurons.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.969178	0.158653	0.94702	0.993056	0.969492	0.987966



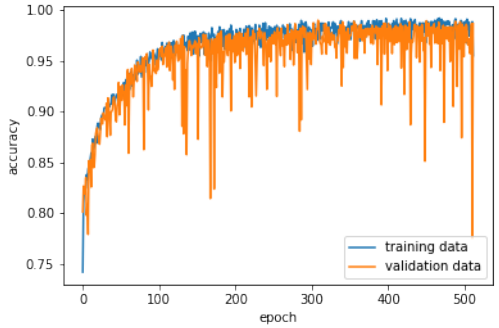


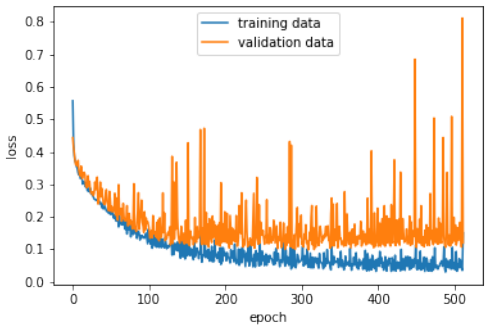


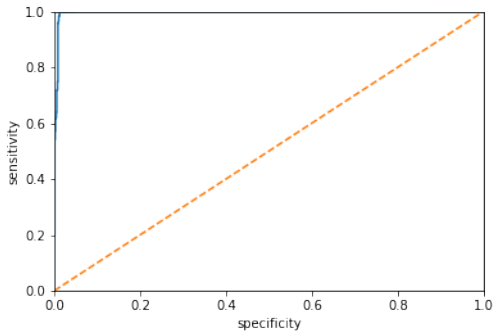
4.2 Overfit Model

The models I have tested on this data all seem to perform well. In order to create an overfitting model, I used a 5 layer model. The model used 512 epochs, and each descending layer contained 25% of the neurons of the previous. It was difficult to find a model that showed an overfit because most models seem to be able to generalize the data. In this model, the validation loss curve was above the training loss curve.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.986301	0.065453	0.97549	0.996994	0.986125	0.995604







4.3 Last Layer Changed to Relu

Because the problem is binary classification, the last layer should be sigmoid; however, these next models used relu instead. I trained these models for 64 epochs.

4.3.1 Model 1

This model uses 1 layer and 8 neurons.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.806751	0.486973	0.786398	0.826788	0.806087	0.880884

4.3.2 Model 2

This model uses 3 layers and 64 neurons in layer 3, 16 neurons in layer 2, and 1 neuron in layer 1.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.914873	NaN	0.853913	0.993927	0.918616	0.970585

4.4 Best Model

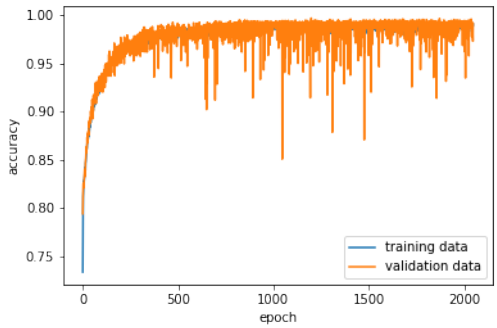
The best model found was model 4 from the tests in section 4.2. Model 4 was a 3 layer model, using 128 neurons for the 3rd layer, 32 neurons for the 2nd layer, and 1 neuron for the last layer. It was trained for 128 epochs and used a batch size of 256. The optimizer was rmsprop and the activation layers used were relu execpt for the final layer using sigmoid. The accuracy of the

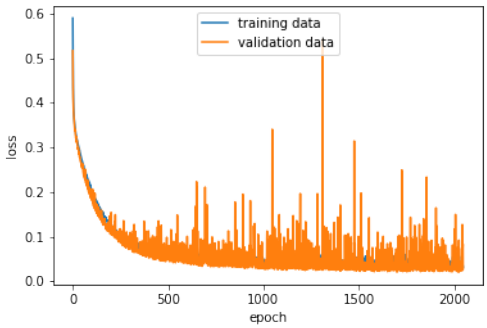
model was consistently similar for the training and validation data throughout the fitting of the model. The ROC curve produced was also very close to the top left corner.

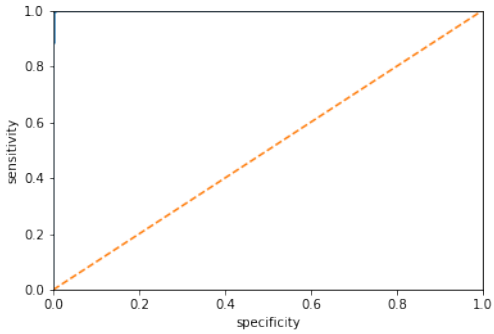
5 Early Stopping and Checkpointing

Using the values from the best model found from the tests in section 4, I created a model to used checkpointing and early stopping. I trained this model with a patience of 512 for 2048 epochs. It completed the epochs and found the best loss value at epoch 1701.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.993151	nan	0.991194	0.995088	0.993137	0.997627



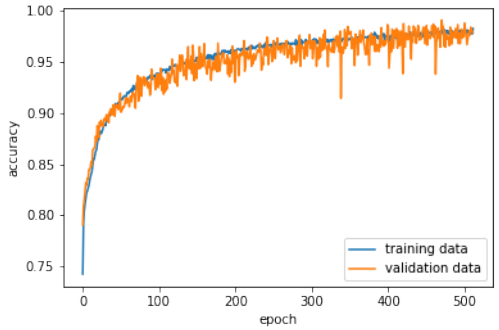


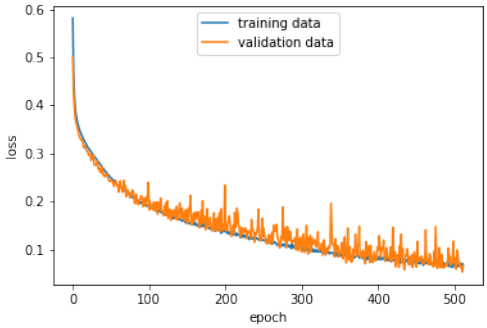


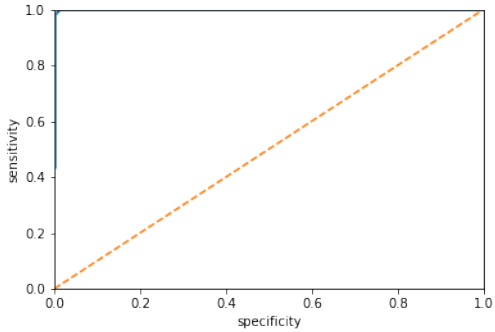
6 Best Fit Model with Highest Accuracy

Using the values from the best model found from the tests in section 4, I created the same model but trained it for more epochs. When I trained it for very large amounts of epochs, sometimes the results were unpredictable, and the model would become overfit to the training data. When I lowered the epochs to 512, or 4 times larger than the original test, then the model performed better and had a much lower validation loss score than the original test. This model did not perform as well as the model from the checkpointing test.

Accuracy	Loss	Precision	Recall	F1 Score	AUC
0.978474	0.065543	0.958212	0.998987	0.978175	0.997898







7 Feature Testing

To understand the importance of the individual features in the dataset, I created models of different feature sets using the hyperparameters from the best fit model. First, I created a model for each of the features to test the accuracy of each feature individually. Using the accuracy results from these tests, I ran tests where I created models by continually removing features in order of the accuracy results from low to high until two features were left.

7.1 Single Feature Models

Of the single feature models, the best performing features were Score and Accuracy. The title of the each result table is the single feature that was used. The results showed that Score and Accuracy were the features containing the most information about our output. Score was a strong predictor by itself.

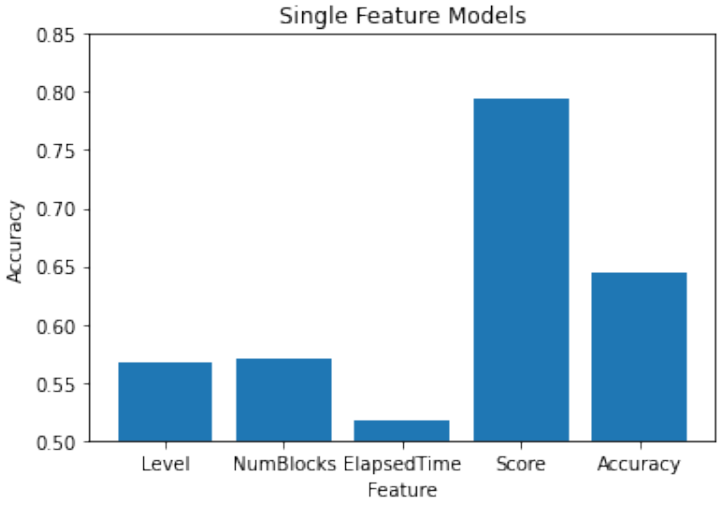
Level			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.564780	0.566536	0.683778	0.684367

NumBlocks			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.564990	0.570450	0.675491	0.674306

ElapsedTime			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.521803	0.518102	0.692198	0.692543

Score			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.803564	0.793542	0.403179	0.471723

Accuracy			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.671069	0.643836	0.618103	0.629783



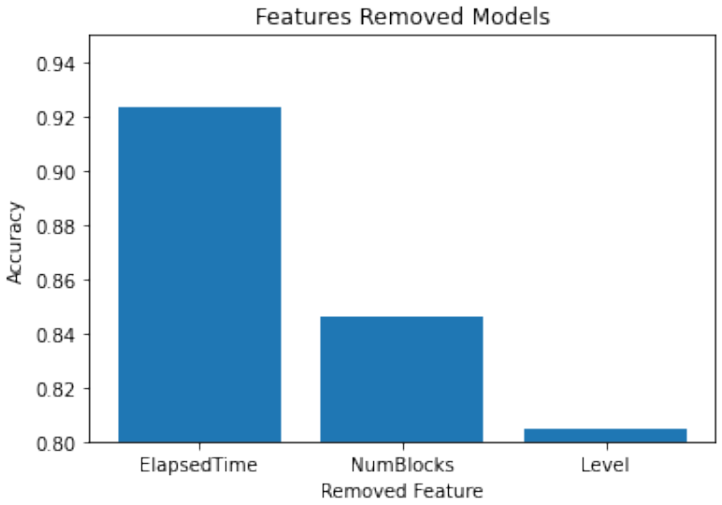
7.2 Features Removed Models

Using the accuracy results from the single feature tests, I eliminated the features in the order of the lowest accuracy score. I eliminated features until only Score and Accuracy were remaining.

No ElapsedTime			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.942558	0.923679	0.162861	0.221792

No ElapsedTime or NumBlocks			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.848008	0.846380	0.346879	0.344794

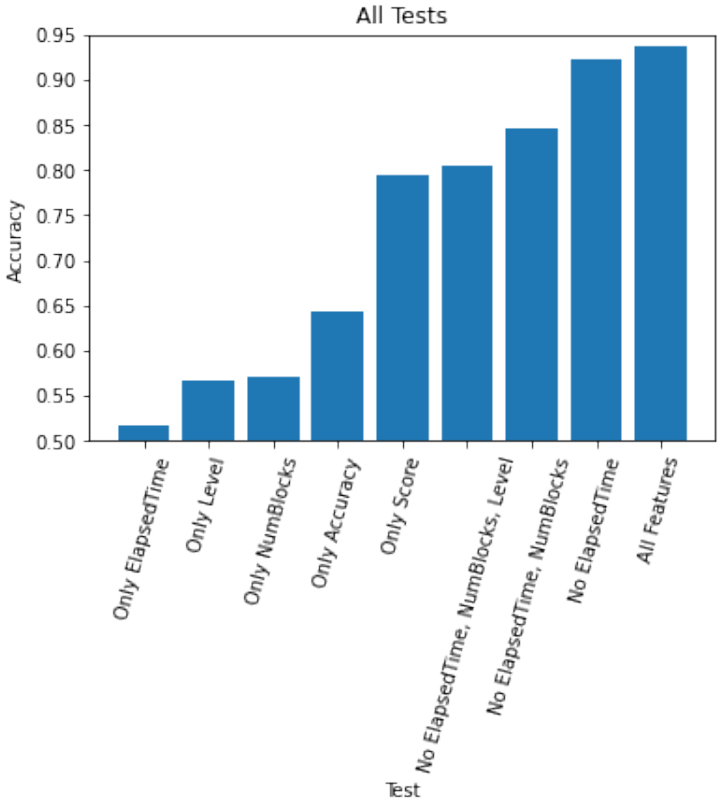
No ElapsedTime, NumBlocks, or Level			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.800839	0.804795	0.401473	0.393210



7.3 All Features Model

After testing different combinations of features, I ran a test to compare the results to a model containing every feature. I displayed the results from least to most accurate.

All Features			
Train Accuracy	Valid Accuracy	Train Loss	Valid Loss
0.946541	0.937867	0.176638	0.175824



7.4 Feature Test Conclusion

Score is the most important feature to determining whether the game was won or lost. This makes sense because a level likely always has a minimum score after being beaten. Accuracy was somewhat predictive at 65% which is good to know because the accuracy feature represents the expectations of the players input. ElapsedTime does not seem to affect the results of the training much and could be removed. The rest of the features are important to raising the accuracy of the AI.

Conclusion

A highly accurate model can certainly be trained to predict whether the statistics produced during a gameplay signify a win or a loss. Using a fairly common set of hyperparameters and also employing techniques such as checkpointing will produce a model as accurate as 99% on both training and validation data.

Predicting a win or loss in a brick breaker game is not necessarily useful by itself, however, understanding which features predict a win or loss is extremely helpful. This type of information would be very difficult to conclude without machine learning. Also, the model can be used for predictions on new data to test the difference in the new data to the old data. New levels and win/loss prediction models for them can be created and their results pitted against each other to discover the commonalities between different configurations. This can allow humans to draw conclusions about different level designs. Being able to slowly introduce confusion to a highly accurate, preexisting AI can be a useful measurement to understand the types of changes introduced. An example for this case would be lessening the importance of the score. If the developers want the win and loss score to be similar because they want to reward players for time playing, but they don't want it to feel completely unearned, then they could test their changes with similar, new models until the score feature's accuracy is lowered from its current 80% to whatever their target is. Caution should be taken when trying to measure data alteration by training the same model hyperparameters because the best fitting model may have altered as well.

References

[1] Vance, T. D. (2019, September 10). Arcade game stats. Kaggle. Retrieved February 21, 2022, from https://www.kaggle.com/depmountaineer/arcade-game-stats