Jon Rippe
CSCE A415 – Machine Learning
Project Write-up
28 April 2021

LEARNING A VIDEO GAME

Supervised Machine Learning

PRELIMINARY

GOAL

Teach a machine learning model how to play a video game.

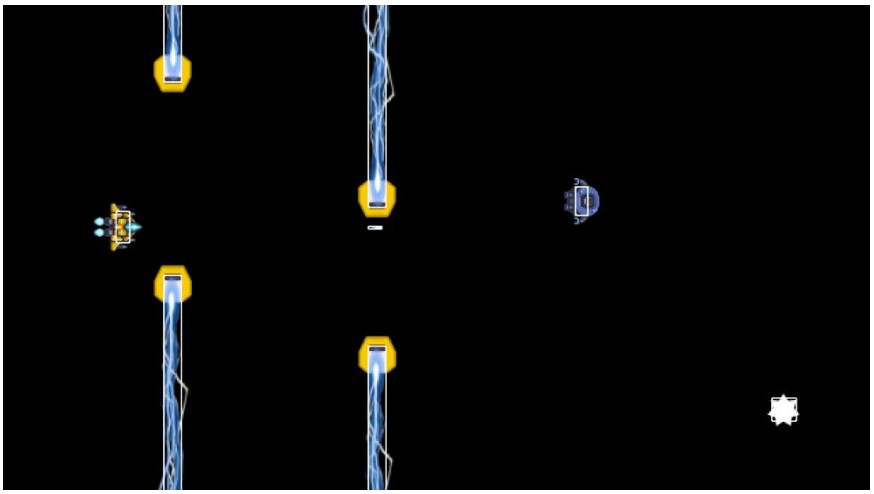
METHOD

1. Collect information about object positions and keystrokes during gameplay (i.e., data that shows proper gameplay).
2. Feed collected data into a machine learning model.
3. Allow an AI to determine keystrokes during gameplay based on current object positions that are fed into the machine learning model

GAMEPLAY

The game is a side scroller. A player controls a ship (located at the left of the screen) that can move and fire lasers. Allowed movements are Up, Down, and Fire. Only one laser may be fired at a time. The goal is to avoid walls, collect coins, and shoot/avoid enemy ships.

See Right for game screenshot. White rectangles show object hitboxes.



COLLECTED DATA

- Above Wall: Boolean – player y location is above the nearest lower wall's top.
- Below Wall: Boolean – player y position is below the nearest upper wall's bottom.
- Wall Distance: Integer – distance between the player and the nearest wall (or screen width if the nearest wall is off screen).
- Coin: Boolean – a coin is currently on the screen.
- Coin Y Offset: Integer – difference between the player's y position and the nearest coin's y position.
- Coin Distance: Integer – difference between the player's x position and the nearest coin's x position.
- Enemy: Boolean – an enemy is currently on the screen.
- Enemy Y Offset: Integer – difference between the player's y position and the nearest enemy's y position.
- Enemy Distance: Integer – difference between the player's x position and the nearest enemy's x position.
- Enemy Velocity: Boolean (modified) – vertical direction the enemy is moving (-1 for up, 1 for down)
- Shot: Boolean – a player laser shot is currently on the screen (player can only shoot one at a time).
- K_up: Boolean – the UP key is pressed (actual w key).
- K_down: Boolean – the DOWN key is pressed (actual s key).
- K_space: Boolean – the SPACEBAR is pressed.

Possible key combinations are Up, Down, Space, UpSpace, DownSpace, None.
These will be used as classes.

Data Cleaning/Prepping

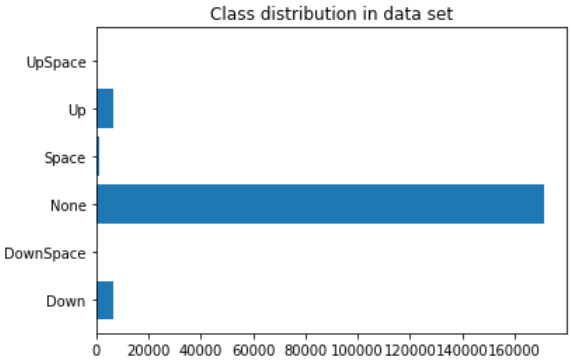
Removing Junk

There is some residual data left over from early ideas on collection and modeling and some junk data from dataset conversion, saving, loading, etc. The following attributes from the actual dataset are dropped:

- Timestamp
- SCORE
- Unnamed: 0

Normalization

Not all the models visited require data normalization, but at least one does. It shouldn't hurt the one's that don't need it. All attributes will be normalized. Booleans should remain the same, the Enemy Velocity Boolean should change to 0/1, and all other values will normalize from 0-1 based on their min/max values.



Noteworthy Observations

- Some data is correlated (e.g., Enemy & Enemy Distance).
- The data set is very imbalanced. This may be the most critical issue.
- Train/Test splits need to be stratified due to the imbalance.

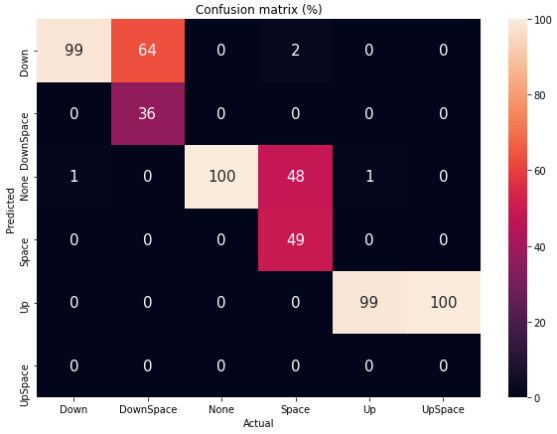
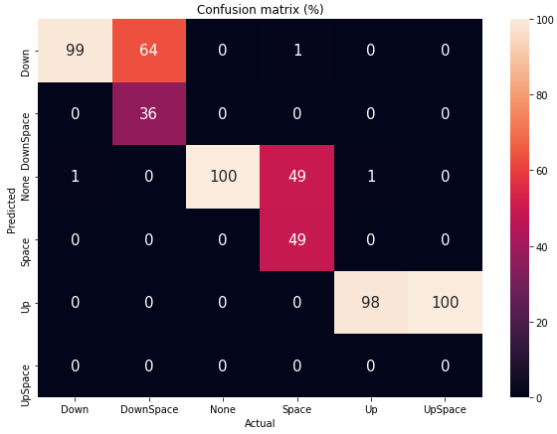
Models

Model Selection

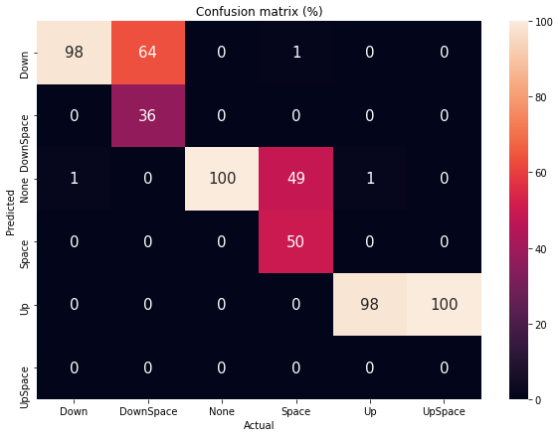
Selected models will ideally make quick predictions so the game can run, and the AI can make predictions, in real time.

Random Forest Classifier

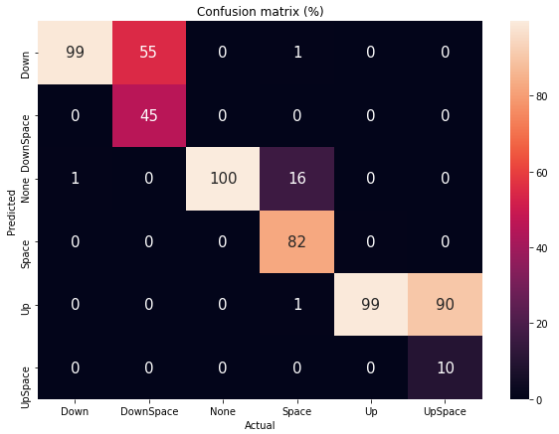
A decision tree is nice, but a forest is better. The value for *n* will be important. The more trees, the longer predictions will take.

Base Test: n=100 As expected, the model has difficulty with the least represented classes:	Increased Estimators: n=1000 Increasing the estimators doesn't affect results:
	

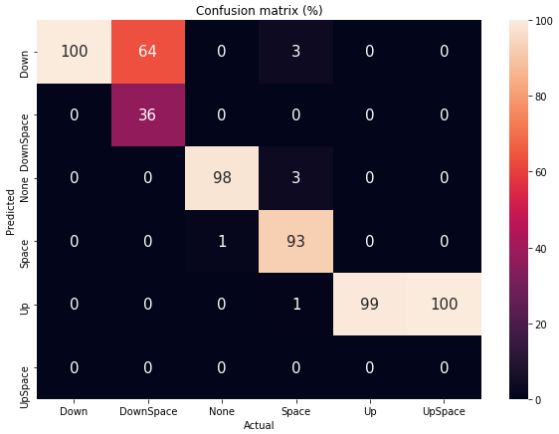
Balanced Weights:
n=100
Surprisingly, balancing the class weights to account for the data imbalance doesn't have a significant effect on predicting the least represented classes:



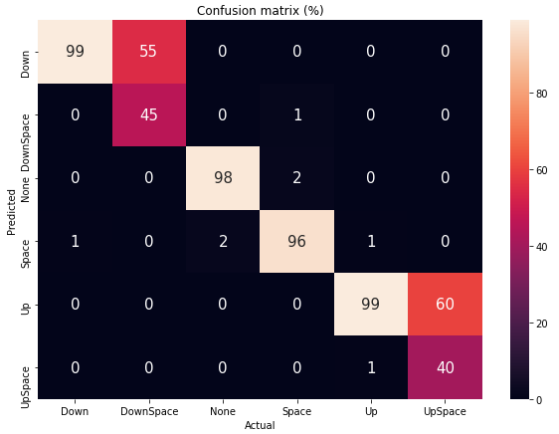
Oversampling (SMOTE):
n=100
SMOTE Oversampling slightly improves classification of least represented classes:



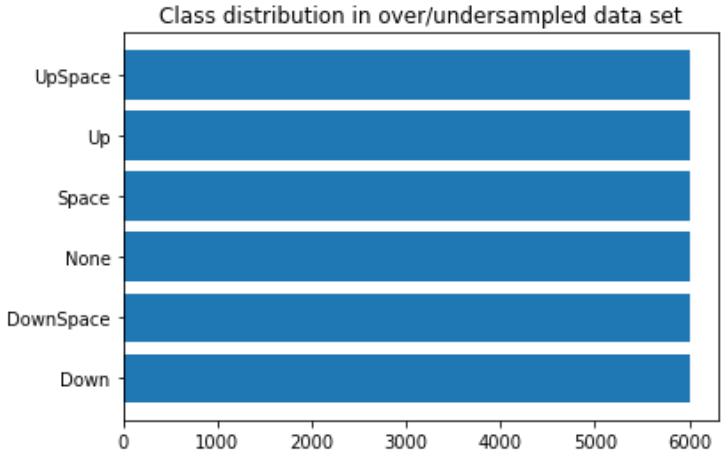
Undersampling (Random):
n=100
Undersampling has no significant effect.

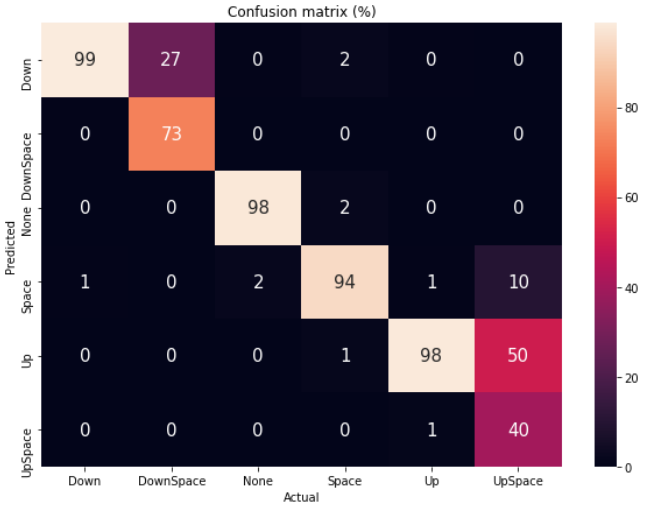


Undersampling+Oversampling:
n=100
Undersampling the majority class closer to the median classes, then oversampling all other classes shows progress in successfully predicting the least represented classes.



Final Model: Random undersampling + SMOTE oversampling
n=10
This forest may perform "well enough" while keeping *n* count low. Training data was first undersampled on the majority class to bring it closer to median classes, then the entire training set was oversampled to bring all classes to the same representation.





Neural Network

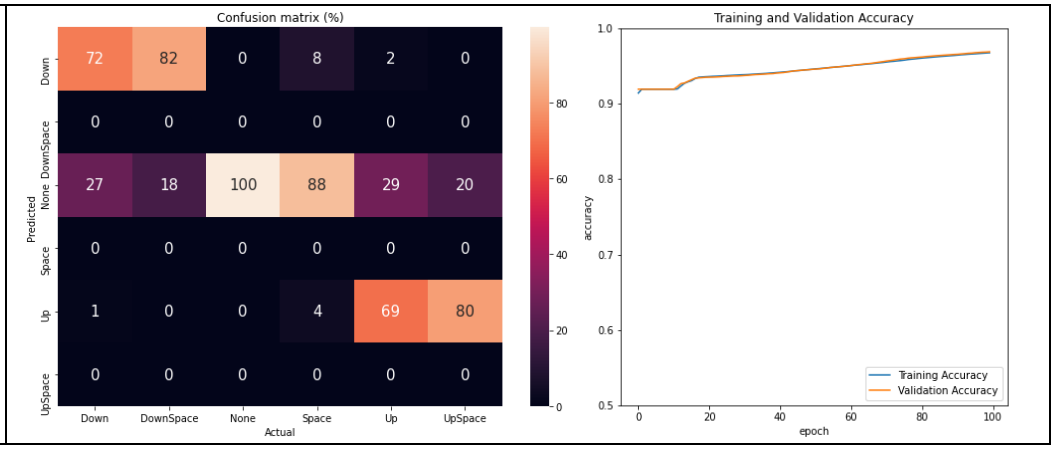
Initial Settings:

- Large batches will be necessary due to the class imbalance. Using larger batches will increase the probability of each class having at least some representation within each batch.
- Dataset has a relatively low number of features, so epoch count can be maximized.

Base Test:
batch_size=4096
epochs=100

Layers:
Dense(128, relu)
Dense(6, softmax)

While the overall accuracy is high, the model is still unable to classify the least represented classes.



Adjusting Weights:

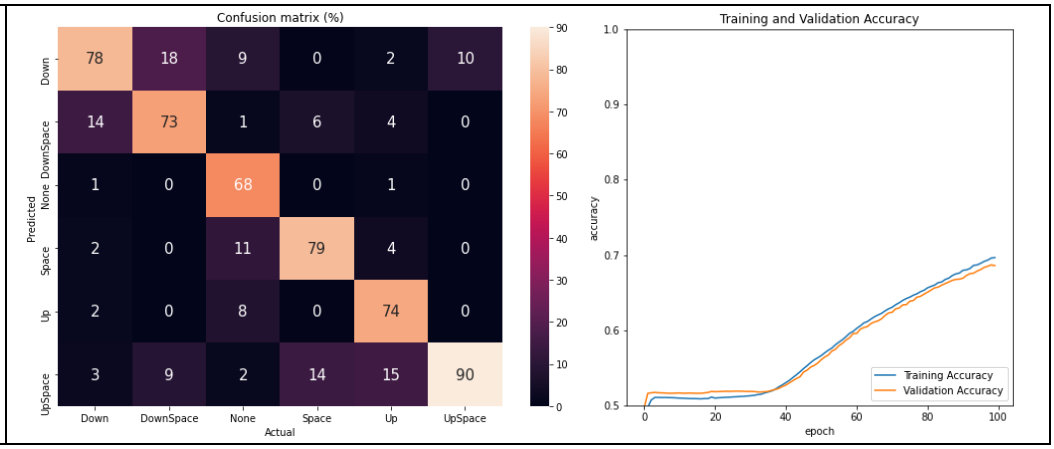
Including weight adjustments based on class representation should help the model classify the lowest represented classes. Each class will be weighted based on the ratio of its cardinality to the total number of datapoints:

$$W_y = 0.5\frac{N_{total}}{N_y}$$

Weighted:
batch_size=4096
epochs=100

Layers:
Dense(128, relu)
Dense(6, softmax)

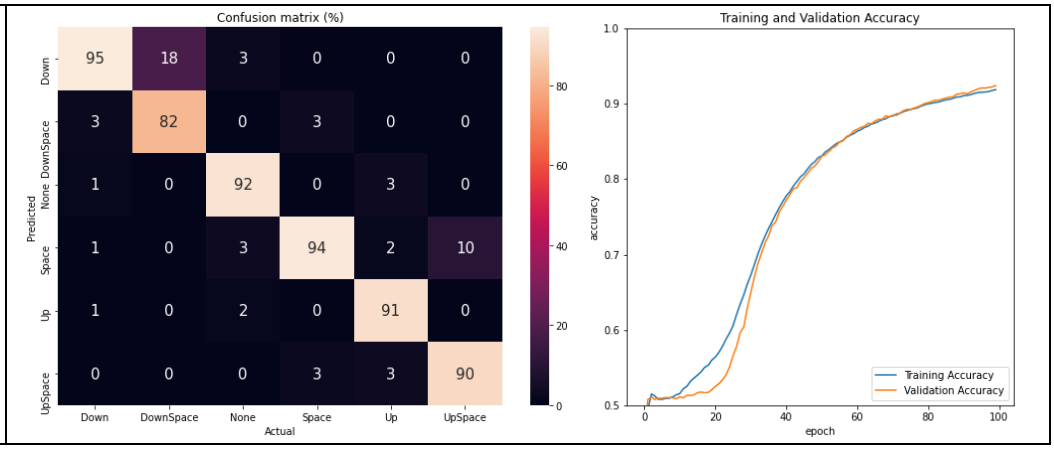
The model is better able to classify the least represented classes at the expense of the other classes.

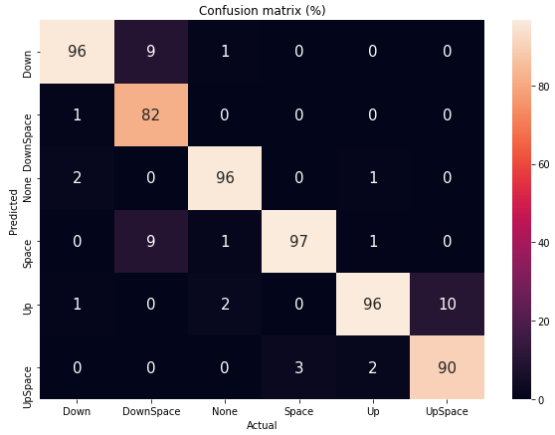
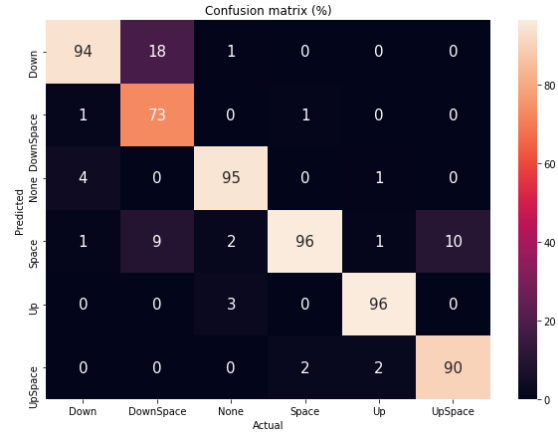
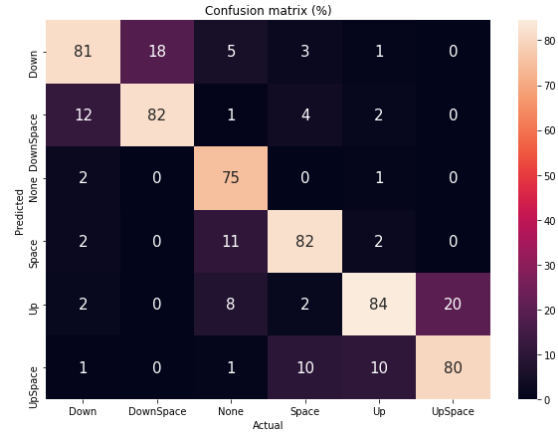
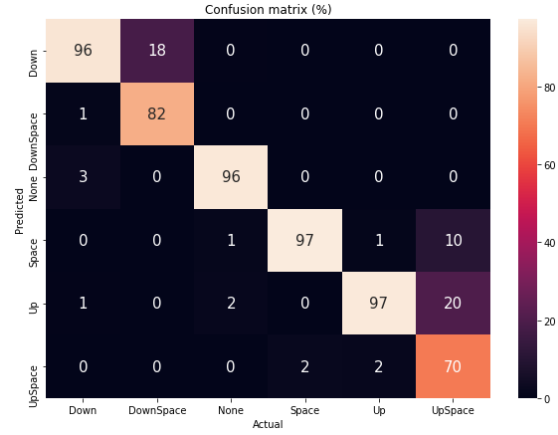
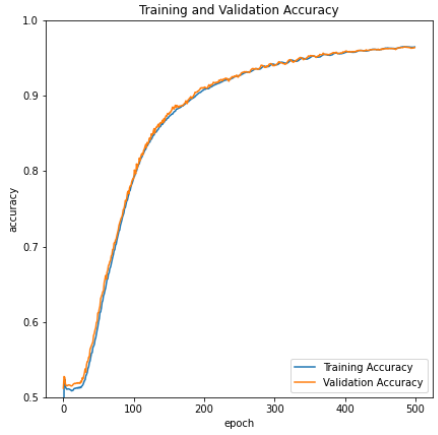


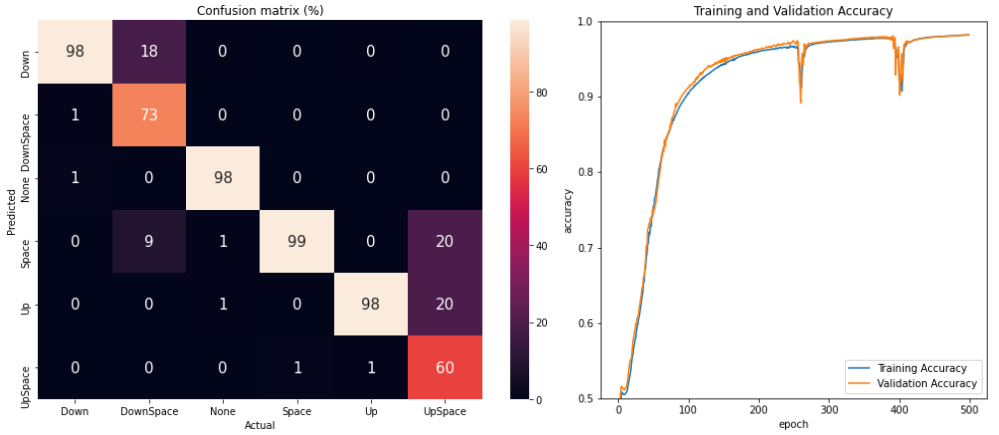
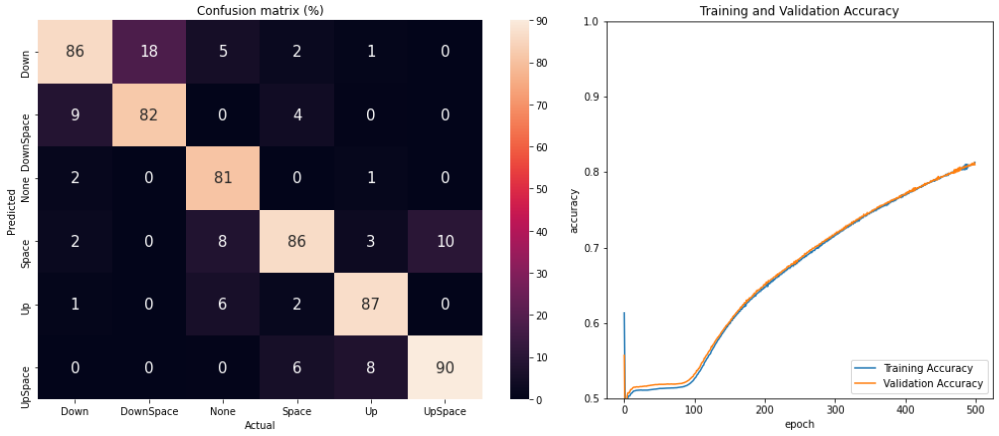
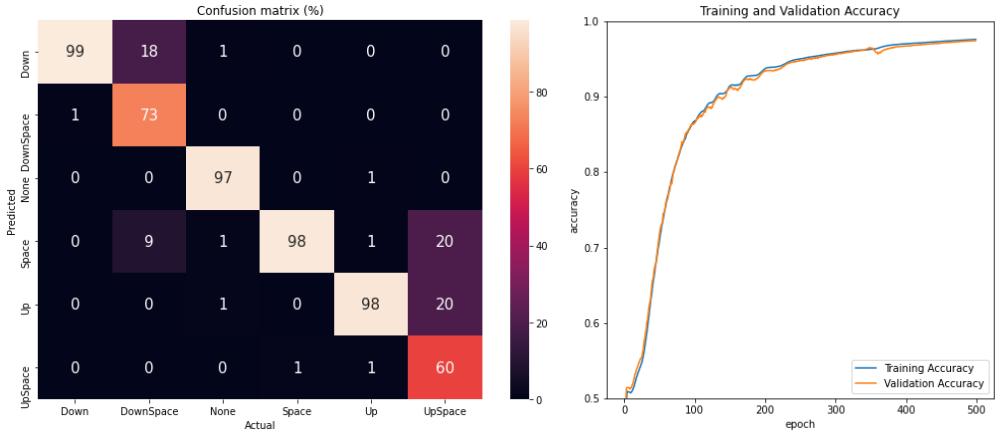
Testing Various Combinations:

batch_size=4096
epochs=100

Layers:
Dense(128, relu)
Dense(128, relu)
Dense(6, softmax)

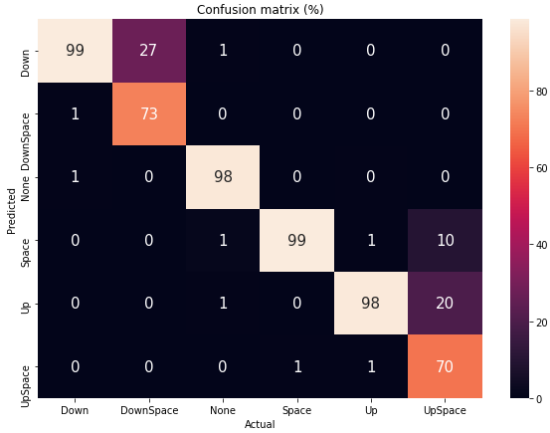


batch_size=4096 epochs=100 Layers: Dense(128, relu) Dense(128, relu) Dense(128, relu) Dense(6, softmax)	 
batch_size=4096 epochs=100 Layers: Dense(256, relu) Dense(256, relu) Dense(6, softmax)	 
batch_size=18607 epochs=100 Layers: Dense(128, relu) Dense(128, relu) Dense(6, softmax)	 
batch_size=18607 epochs=500 Layers: Dense(128, relu) Dense(128, relu) Dense(6, softmax)	 

batch_size=18607 epochs=500 Layers: Dense(128, relu) Dense(128, relu) Dense(128, relu) Dense(6, softmax)	
batch_size=18607 epochs=500 Layers: Dense(128, relu) Dense(6, softmax)	
batch_size=18607 epochs=500 Layers: Dense(256, relu) Dense(256, relu) Dense(6, softmax)	

Final Model:

Increasing batch size to 18607 (half the size of the validation set) and minimizing layer numbers greatly increased training speed and stability. The optimizer algorithm used (Adam) caused some spikes in the training charts. Increasing batch size, reducing layer count, and reducing node count helped to mitigate these spikes. These changes also sped the model up, which allowed more epochs to be run and reduced the complexity of the overall model, increasing prediction time.

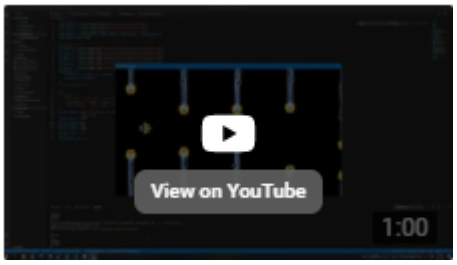
batch_size=18607 epochs=5000 Layers: Dense(128, relu) Dense(6, softmax)	

Final Results

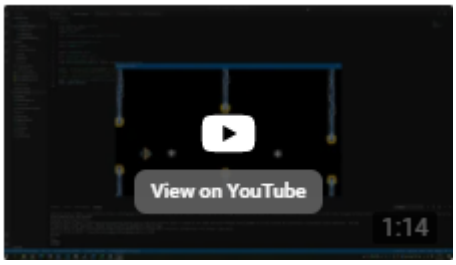
Adjusting class representation, either through over/undersampling or weight adjustment, was the most important factor in an accurate model. Both models performed well in an applied test. The neural network provided faster, more accurate predictions, which resulted in quicker, more skilled gameplay. While the models still have some trouble shooting, the prediction rate (once every 5 frames) is high enough in most cases to mitigate a small percentage of incorrect predictions.

To see the models in action, visit the following links:

Random Forest



Neural Network



Future Projects

An unsupervised neural network model.