Deep Learning Project

Imtinan Azhar i140078

BOT

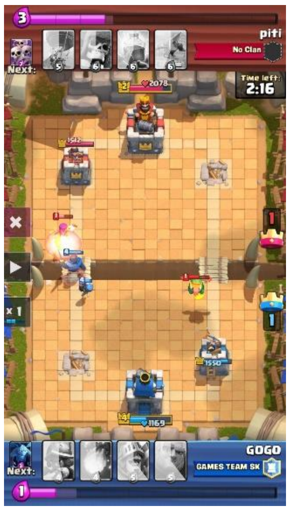
CLASH ROYALE BOT

Abstract:

Clash Royale by the creators Supercell is an online multiplayer strategy game on the Android platform and has been subject to a large community. Our aim will be to create a deep learning based bot that learns the playing style of some of the best plays across the community and tries to emulate these styles so that the bot can play the game via Ai

Introduction:

The aim of this project was to create a bot that works via deep learning methods and successfully learns how to play the game, making both offensive and defensive moves, and simultaneously learns how to beat an opponent.



So let's see what the game is about. The game is a two player game and runs for a timer of 3 minutes, if at 3 minutes the game is still won by neither party then another extra minutes is added to the timer to help decide the winner. The objective of the game is to capture the maximum number of the opponent's towers. Each player has at maximum 3 towers, if a player loses a tower to the opponent, the opponent gets a crown, the first one to three crowns wins, if the game end timer is reached then the player that holds the most number of crowns is declared the winner. To capture towers the player must use cards to his disposal, as of now there are 80 cards in total in the game. Before a match a player gets to decide on 8 cards to add to his deck, this deck is then used to play the game, initially 4 random cards are present to a player to play from, every time a card is played it is replaced by a random card, each card requires a certain amount of elixir to be played, a player can hold a maximum amount of 10 elixir, elixir regenerates slowly. Initially a player can place a card anywhere on his side of the board, later the board size increases as towers are captured. When cards are deployed they display a NPC that attacks based on some pre-set functions. Some cards are offensive and some defensive. We aim to build a bot that can initially beat the Ai put forth by the creators in the training camp section.

Clash Royale has this section known as RoyaleTV, it holds some of the best day to day plays that occur in matches amongst the community. This will be our main source of gameplay data form which we ill expect the bot to learn, but this will still require hefty pre-processing before our bot can be trained.

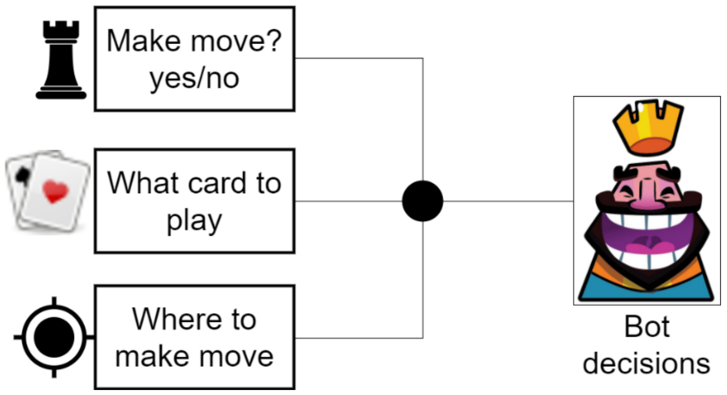
Our aim was to initially create a bot that could beat the training camp bots available in the game. Such a desirable result was achieved. Sadly the training camp feature as removed from the game and was only available to beginners, which after one time clearance was inaccessible again. So in order to test the bot we had to create a dummy account on an emulator to run the game. Our bot was able to beat the bot that was present in the game, and the next step would be playing with a real life opponent.

Methodology:

After careful examination of the gameplay features and methodology it was realize that our bot would need to make three decisions:

1. Whether to make a move or not
2. Which card to play
3. Where to play the card

Since we needed to make three predictions, we settled on creating three different models for this purpose, the model can be seen below:



We have thus named our models as follows:

- movePredictor
- cardPredictor
- positionPredictor

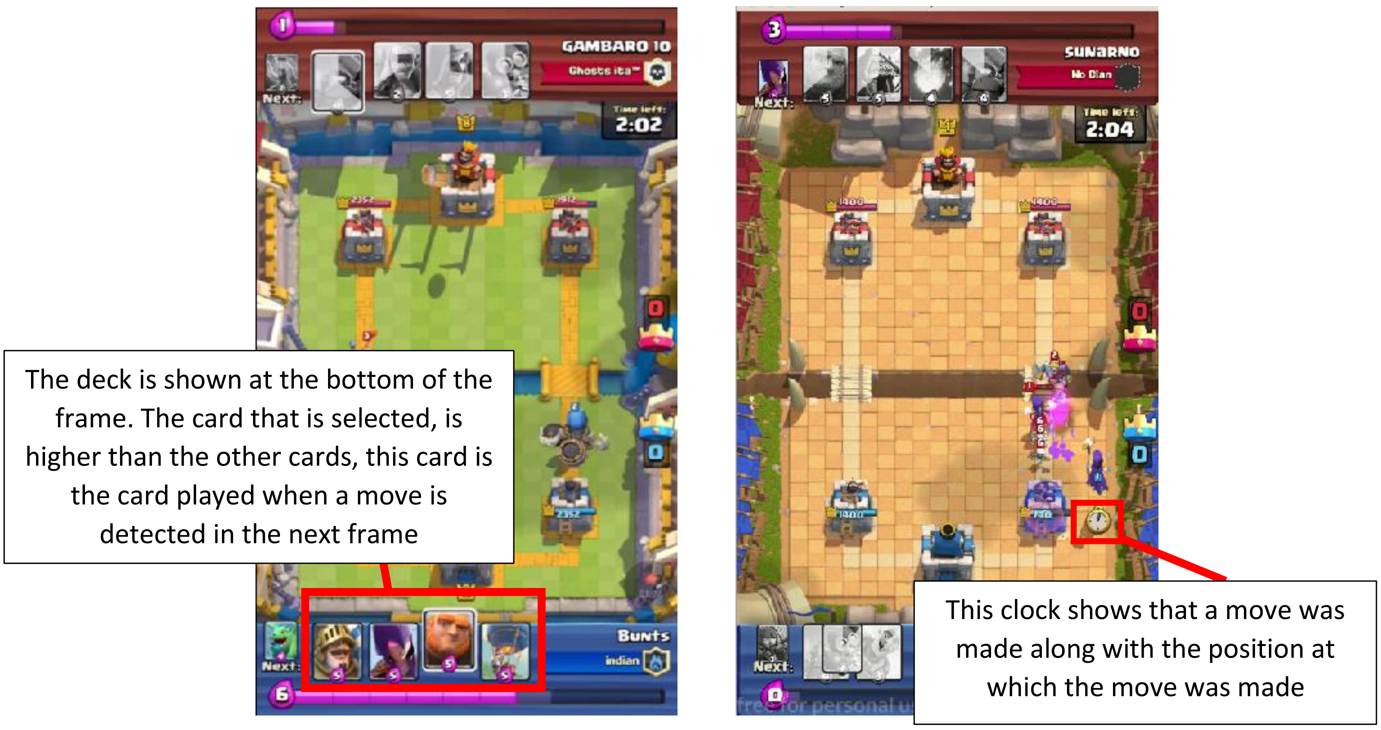
Methodology used to create each of these models will be explained later, but first let’s have a look at the methodology that was applied on the data for pre-processing purposes.

Data pre-processing

Before we began to process the data, we broke down all the info that we needed, it was as follows:

- When a move was made – movePredictor
- What card was played - cardPredictor
- Where was the card played – positionPredictor

Let’s first have a look at the board and analyse how this information could be obtained. The following us an analysis of the board.



The deck is shown at the bottom of the frame. The card that is selected, is higher than the other cards, this card is the card played when a move is detected in the next frame

This clock shows that a move was made along with the position at which the move was made

The following is how the information was collected for each of the models

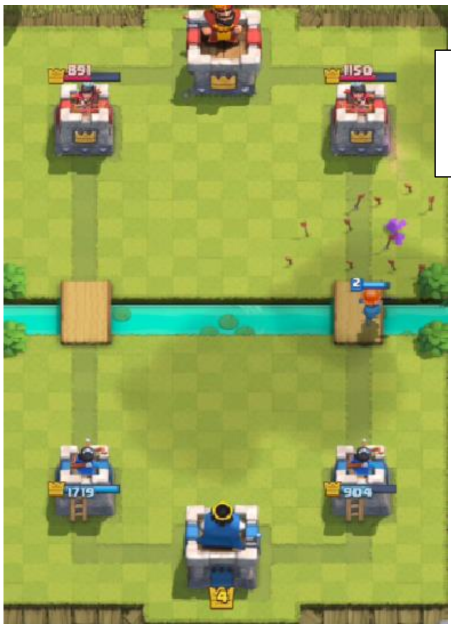
- When a move was made – movePredictor

For this we needed to detect a clock in a frame, if the clock is detected then a move was made. If the move was made then we save the previous frame along with the label “1”, this signifies that a move was made in this frame, if the move was not made then we save the previous frame along with the label “0”. We do not save the entire frame, since the entire frame will not be available at test time, instead we just save the game board image.

Moreover the reason for saving the previous frame is that if we were to save the current frame, then the model could mistake the presence of a clock for when a move as mode

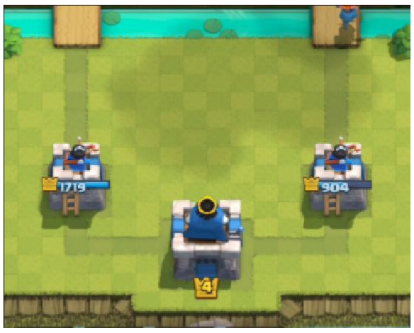
We use template matching as a method for clock detection, the following image was used for template matching. Moreover we do not scan the entire board for this clock, only the board section of the bottom player, in order to not mistake the opponent’s clock for a move being made

This is some sample data:



The frame area on the left shows that is saved when moves are detected / not detected

The frame area below shows the area where the clock was to be detected for



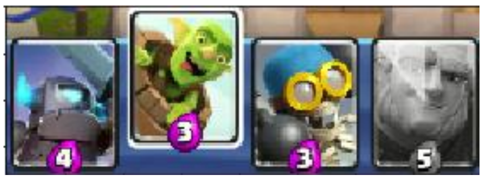
- What card was played – cardPredictor

To predict what card is to be played we need two sets of data in order to train the image. We plan on creating a CNN which accepts some additional data.

The data extracted is as follows:

- Inputs:
  - The frame at which the card was played
  - The deck of cards that was available
- Labels:
  - The card that was played

We cropped the deck area of the frame which can be pictured on the side and used that to gather an array of data representing the set of available cards. We used the template matching technique used to detect clock locations before to detect card. We sampled the deck crops with specially processed card areas. These areas were extracted, cropped, processed and saved beforehand.



These sample frames are shown on the side

We use these areas specifically to minute the differences between the card images available to us and the images on the deck

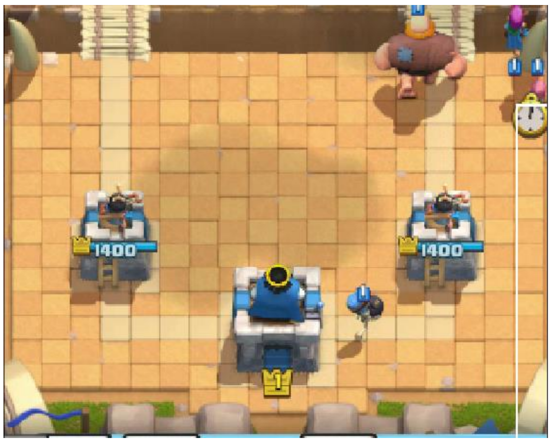
After the deck is extracted we get a one hot encoded array that is a representation of the available deck, the array is as follows:

```
[0. 0. 0. 0.  0.  1.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0
0.  0.  0.  0.  0.]
```

For the label we detect the card that has the smallest y coordinate, this indicates the card that has been selected. This selected card shows us the card that as played, this information is represented with another such one hot encoded array but with only one index as selected

- Where was the card played – positionPredictor

This data collection process was somewhat similar to the movePredictor data collection process, for this part we extracted the positon of the detected clock, this showed us where a move was made. We divided our section of the game board into 270 tiles, the game board is represented via an array of size 18*15 and one of these positions is highlighted showing the position of the detected clock.



The frame area on the left shows the tiles on the game board, allowing for 270 different positions to be made

The frame area on the right shows one tile cropped from the game board itself

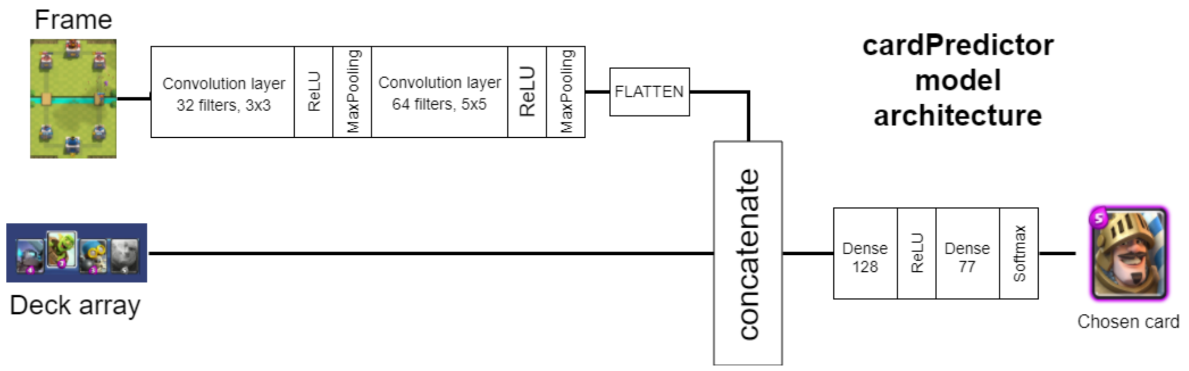
We needed some additional information for this section as well, the position of a card is also dependent on the card that is being placed, so we used some information from the previous section, along with our frames we also saved the card that as played as input to our model and the label associated with it was a flattened array representing the position of the clock on the 18*15 board.

Model Architectures

We used CNN architectures for our model. We used two unique CNN architectures that accept additional information along with images as its input. The third CNN architecture is a very simple binary image classifier, hence a complex architecture wasn't necessary.

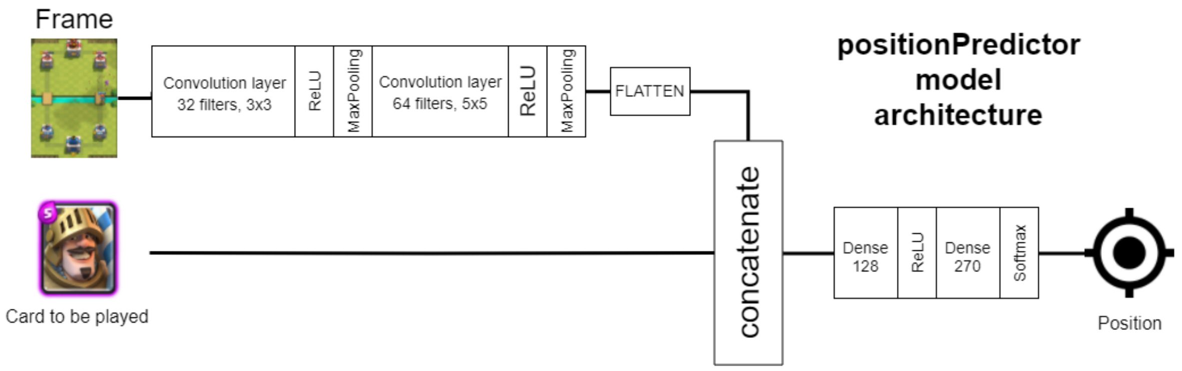
The model architectures can be seen below

Card predictor model



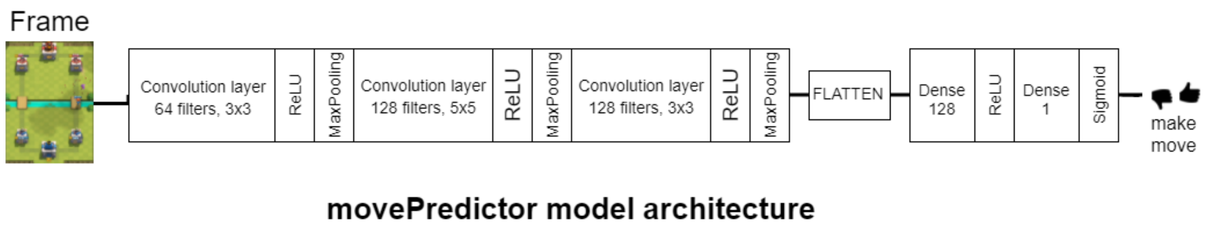
This model takes input a frame image along with the array representation of a deck, the input frame goes through a 3x3 convolution layer of size 32, then batch normalization is applied along with a ReLU activation function, after which max pooling was applied, this process is repeated again but with 5x5 convolutional filters of size 64 and then the filters are flattened. These concatenate with the deck array and are passed through two dense layers of sizes 128, and 77 respectively, with the latter having a Softmax activation function. The output is a 77x1 output array, indicating the card to be played.

Position predictor model



This model is very similar to the card predictor model as they make similar predictions off similar inputs, it takes input a frame image along with the array representation of a card being played, the input frame goes through a 3x3 convolution layer of size 32, then batch normalization is applied along with a ReLU activation function, after which max pooling was applied, this process is repeated again but with 5x5 convolutional filters of size 64 and then the filters are flattened. These concatenate with the deck array and are passed through two dense layers of sizes 128, and 270 respectively, with the latter having a Softmax activation function. The output is a 270x1 output array, indicating the position to be played at.

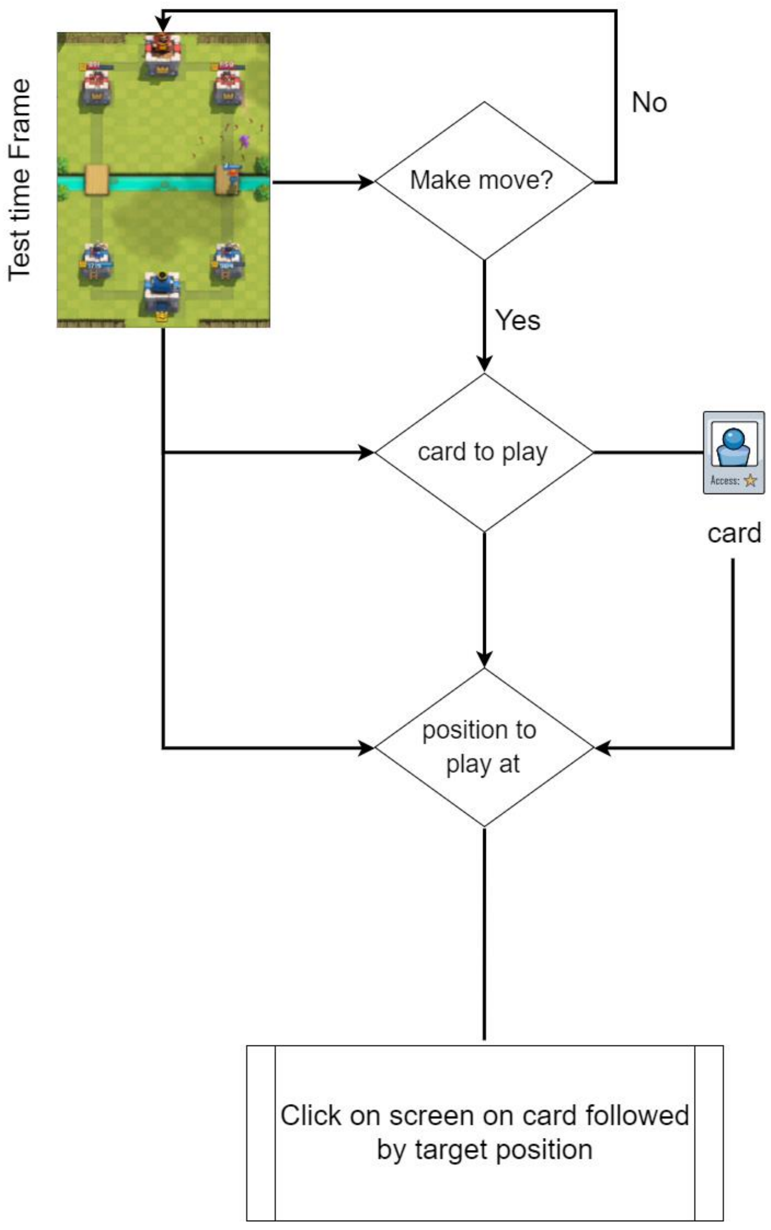
Move predictor model


Frame
Convolution layer 64 filters, 3x3
ReLU
MaxPooling
Convolution layer 128 filters, 5x5
ReLU
MaxPooling
Convolution layer 128 filters, 3x3
ReLU
MaxPooling
FLATTEN
Dense 128
ReLU
Dense 1
Sigmoid
make move
movePredictor model architecture

This is a very simple yes no classifier, we pass it through 3 convolution layers, the first having 64 3x3 filters, followed by another one with 128 filters of size 5x5 and finally ending the convolution layers with 128 filters of size 3x3, they all go through ReLU activation and Max pooling, the filters are then passed through a dense layer of size 128x1 which outputs a prediction on whether to make a move or not.

The first two models were reminiscent of architectures used in a chess game problem, since we have a similar problem where depending on a board state we have to pick a card, for our third model we used a very small network that is used for binary classification.

Test time model usage



At test time the model captures the screen and crops out the game board followed by the deck area.

The frame is passed onto the movePredictor model, the model from here decides whether to make a move or not.

If the model predicts that a move is not to be made then we discard the frame and move on to the next one, if the model decides to make a move then we pass the frame onto the cardPredictor model.

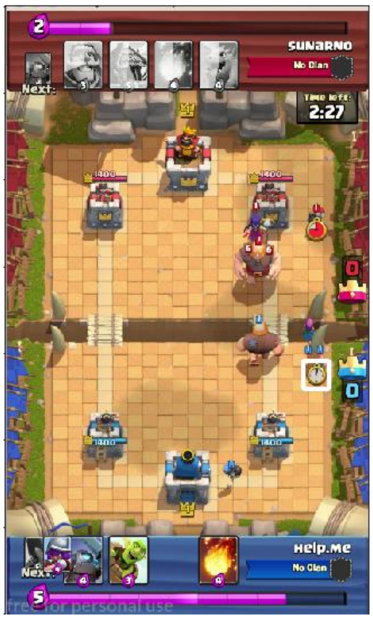
Deck information is extracted from the deck crop and is passed as supplementary information to the cardPredictor model. This model then decided which card is to be played.

The card to be played and the frame are then passed onto the positionPrediction model. This model predicts where to make the move.

A screen click is then simulated on screen by clicking on the card followed by a click on the position to play at

Experiments:

There were quite a few number of experiments that had to be carried out. Initially there was a problem in finding the clock position and extracting the deck form the images. Many methods were used for this and can be seen on the project Github page in the experiments section, where multiple ipython notebooks with the conducted experiments.



Finally after settling for the template matching methodology some hyperparameter tuning as necessary, some threshold value was to be decided for finding the clock.

With a threshold value of 0.7 the result on the left as achieved

Thresholding values smaller would give erronous results and larger would not detect he clock, hence this threshold values was decided.

Other expeiments were conducted with the architectures of the model, iniitially for the card and position prediction mdoels a hefty and large architecure chosen, but due to its large size the porgram was failing to load the model on to the ram, since the training was done on a virtual machine, so we reworked the architecture, decreasing it in size.

Other experiments include tweaking the number of filters and filter sizes in the convolution layers and the sizes of the dense layers.

Coclusion

A partially working model was obtained but couldn’t completely be tested. The problem as that only one model could be loaded at one time. Each module provided somewhat reliable results and could move towards a human like player. The issue with the models that there was no way to meaure the accuracy of the bot, this is becasce the bot learns the moves of actualy players who tend to make mistakes, the bot thsu learns the msiatkes as well, and since we arent forcing the bot to learn the play style of one specific player the different decisions made by different players make it impossible to check the accuracy of the model. Moreover the models couldd tremendously be imporved by further training, since the current data gathered was done by manually sitting d watchig 9 mathces for 3 minutes each, hence if we were t find a reliable source of data and were to extesively train our model on the variety of cards as well as the different arenas, we could achieve a bot that would reach human level progress, the current model also achieves this but is yet to be tested against a human opponent. Moreover finally it should be mentioned that maybe a deep learning apporach was not the right way to go for this bot. A much reliable and better way of achieveing such a bot would be through reinfforcement learning, as done by OpenAi to create a Dota 2 bot. But it is safe to say that we achieveed a working model using deep learning apporaches.

The project can be found at: https://github.com/Imtinan1996/Clash-Royale-Bot-using-Deep-Learning-technologies