Clash Royale Assistant using Dual Convolutional Neural Networks

Introduction to Clash Royale

Feel free to skip this section if you are already familiar with the popular mobile game, Clash Royale. This brief introduction is here to make sure everyone understands the basic rules of the game.

What is Clash Royale? Clash Royale is a one-versus-one card, strategy game. Currently, there is a total of 86 cards in the game. Each player can craft a deck that consists of 8 cards. The goal of the game is to place these cards in order to destroy the opponent's towers. There is a time limit of 3-minutes, and a 1-minute overtime if players are tied (both players have destroyed the same amount of towers after 3 minutes.)

During a match, each player can hold 4 cards in hand. The 4 cards that the players start off with are ***not*** the first 4 cards in the deck. They are 4 random cards selected from the deck. One a card is played, that specific card will be placed at the back of the deck. The last thing to note about this game, is that in order to "play" a card, you must have the card's required *elixir*. *Elixir* is basically a resource in the game that players accumulate over time.

(Figure 1) – A Player's card collection and the 8 cards in his/her current deck.

(Figure 2) – A Versus Match. Note that the player is only able to hold 4 cards at a time. This player currently has 7 elixer.

The problem that I solve

The first thing players learn in Clash Royale would be figuring out, "which cards are good against *x*, *y*, and *z*?" For example, after playing a few matches of Clash Royale, a player would learn that the "Arrows" card is an excellent counter for the "Minions" card, or the "Minion Horde" card, or even against the "Skeleton Army Card". However, this isn't the only skill players will have to learn.

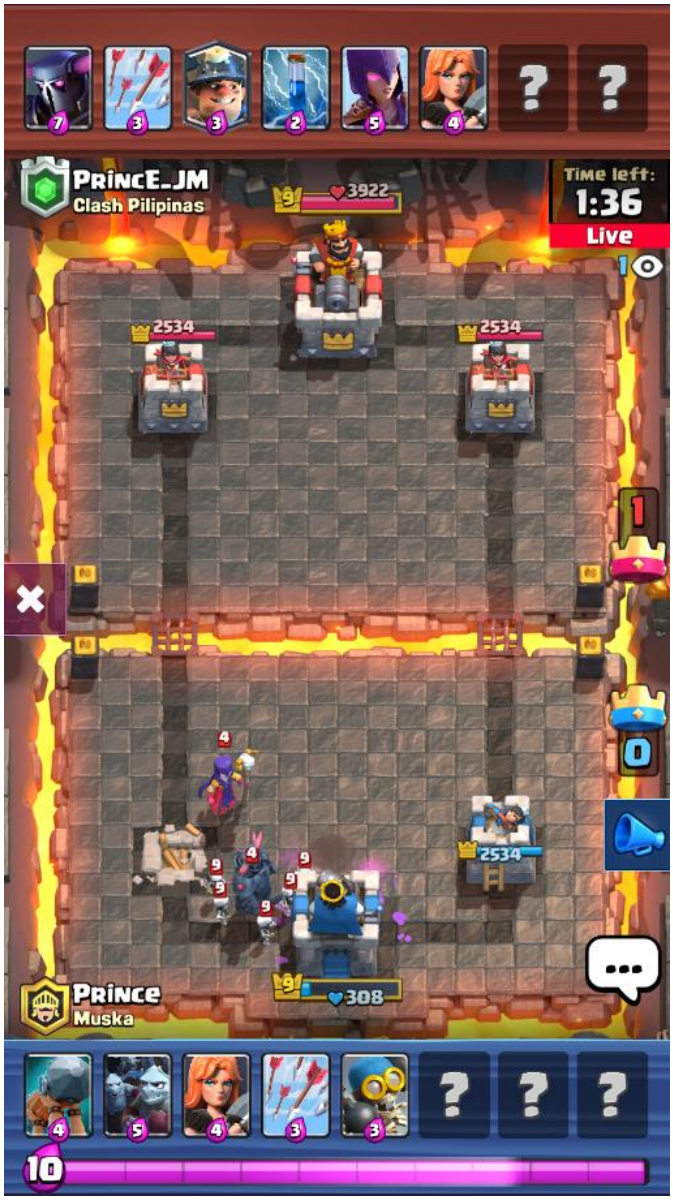
The key to winning in Clash Royale is being able to track which cards are in your opponent's hand, as well as how much elixir they have. For example, if my opponent has "Arrows" in his deck, I should know when "Arrows" is in my opponent's hand. This will allow me to place cards that are bad against "Arrows", such as the "Minions" card. I would place the "Minions" card when I'm certain my opponent doesn't have "Arrows" in his/her hand. Also, counting my opponent's elixir is *very* important as well. If I know that my opponent has "Arrows" in his hand, but doesn't have enough elixir to use the card, then I'll be able to use my "Minions" card optimally.

These skills take some practice learning, since players would have to learn how to keep track of the opponent's card cycle and elixir usage. (Cycle is a term for the order of played cards.) After enough practice, some of the best players have no problem counting cards and elixir.

I decided to create a Clash Royale Assistant that uses deep learning/machine learning to constantly monitor my opponent's cards in hand (or card cycle), as well as my opponent's elixir.

Application Design

Clash Royale allows players to spectate live matches, which gives players a "spectator" view of the match. Note that the spectator can see the cards in the decks of both players, only if the card has been revealed, as in the card has been played at least once. ("?" Cards means that the card hasn't been revealed yet). Figure 3 shows what a "spectator" view looks like.



(Figure 3) – Spectating a Live Match.

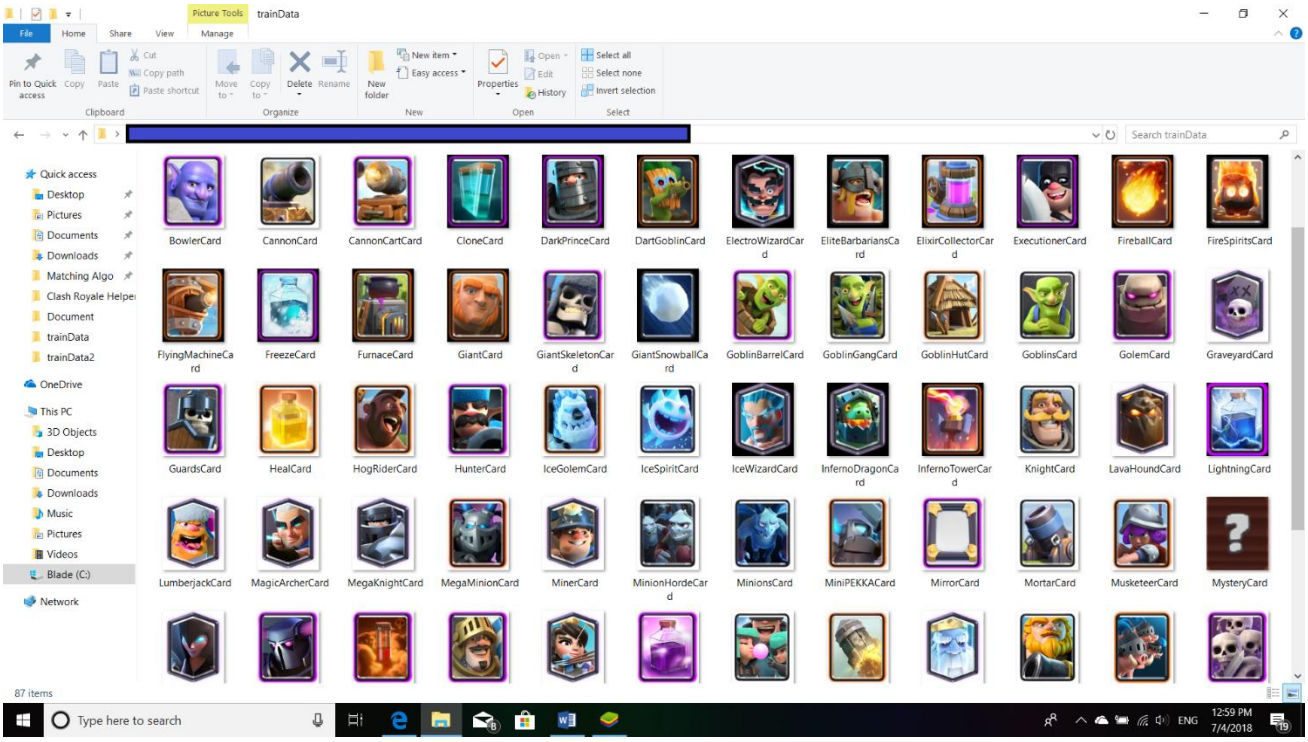
I'm able to use image recognition on this "spectator" view to gain lots of information about my opponent's card cycle and elixir usage. If I'm able to know what cards my opponent has, as well as which cards are being placed down, then I can count my opponent's cards and elixir.

I trained 2 Convolutional Neural Network models (CNN models). CNN's are very popular for image classification/recognition (Detecting certain objects in a picture). In the next section, I'll explain what each CNN was used for. For more technical details about how the CNN's were built and trained, please see the technical report.

The First Convolutional Neural Network

The first step in making this Clash Royale Assistant would be to make an AI that is able to classify/recognize cards correctly. To do this, I decided to use a CNN model. Like all machine learning models, CNN models must be trained with some sort of training data.

The training data used for this CNN would simply be all the card images. All 86 card images were used for training, as well as an extra image for the mystery card (The "?" card). When training a CNN model, it's common to also manipulate the training data, such as shifting the image in a certain direction, zooming in or out, rotating, or even changing the color tone a little bit. This is called Image Augmentation, and it helps the CNN model learn the data better, because it can account for certain image errors (such as misalignment), when we feed in our cropped images for classification.



(Figure 4) – CNN 1's Training Data

After training the CNN model, we can now use it for classification. Let's look at exactly what we want our model to classify (The testing data). Going back to the "spectator" view of the game. If we look at the top of the screen, we can see our opponent's deck (revealed and unrevealed cards). What we want to do is isolate each of these cards to their own individual image. This way we can perform image recognition on each card individually, rather than having

our CNN model trying to recognize all 8 cards from one image. This would require a more complex CNN model that can classify multiple objects at once, also known as a CNN multiclass classifier.

(Figure 5) – CNN 1's Testing Data, what we want to classify/recognize

The figure above shows the image pre-processing phase. This is basically where I prepare the data to get fed into our CNN model for classification. After slicing this image into 8 individual images, I then feed them into our CNN model. The output of feeding in an image to our CNN model, would be a list of 87 separate percentages. We have 87 separate percentages because our model is telling us how confident it is for *each* card. In other words, we have a confidence level for each card.

For example, let's look at the first card, the "Goblins" card. Our model will tell us how confident it is that this image is card 1, card 2, card 3, card 4, card 5, card 6, all the way down to card 87. If we look at the 33rd percentage that it gave us, it is 96.85%. This means that our CNN model is 96.85% sure it is card 33. Card 33 is the "Goblins" card, so our CNN model is 96.85% sure that this image is a "Goblins" card.

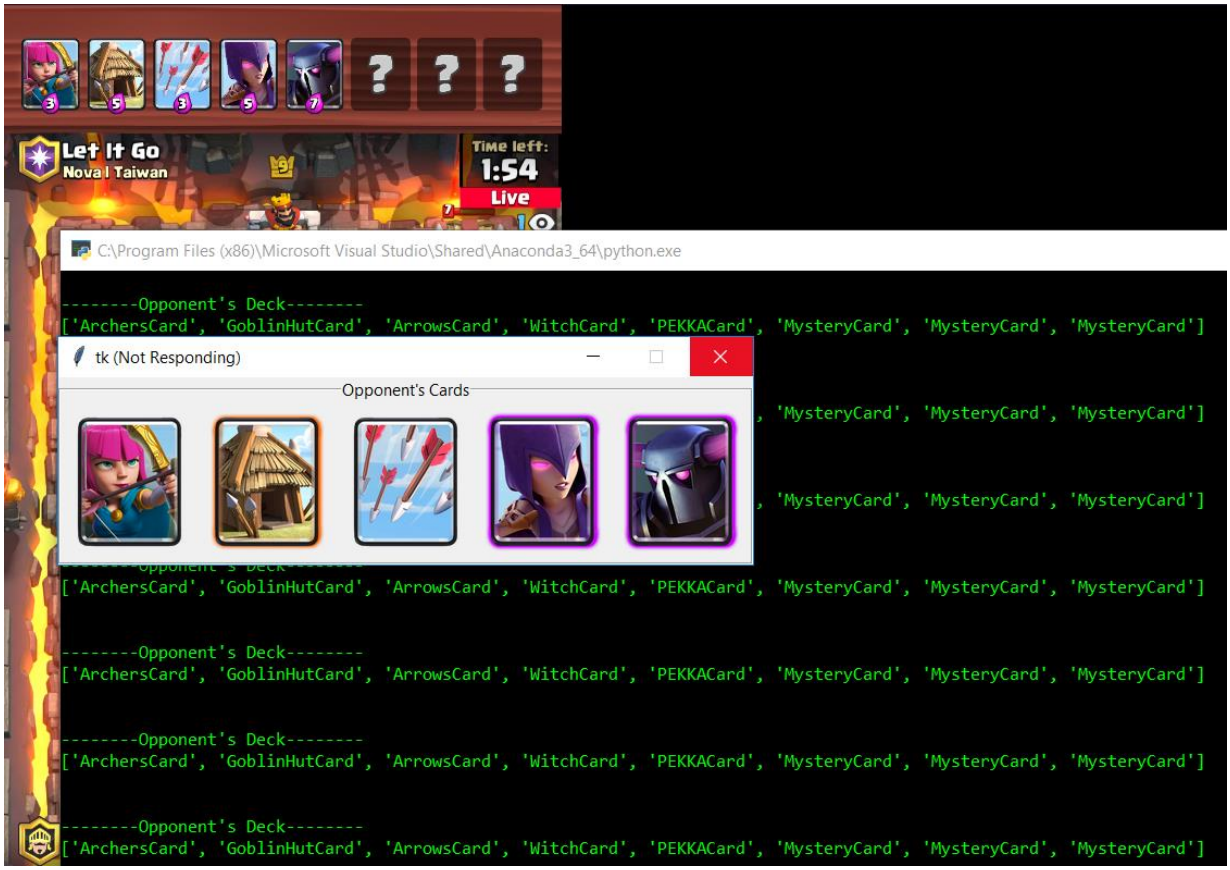
If we look at the first percentage that our CNN model gave us, it is 0.00%. This means that our CNN model is 0.00% sure it is card 1. Card 1 is the "Archers" card, so our CNN model is 0.00% sure that this image is a "Archers" card. Since the confidence is so low (0%), the model is

trying to say that this image is *not* the “Archers” card. To see which card our CNN model is classifying our image, we just go through all the percentages and find the one with the most confidence (highest percent).

(Figure 6) – CNN 1’s Classification on Testing Data.

As shown above, we can see that the CNN model is able to classify these images correctly, even with the bad crops. (This is because our CNN model trained on the original card images as well as the manipulated images from Image Augmentation).

With this CNN model, I am able to watch a live match and have my program keep track of what cards my opponent is using in his/her deck. See Figure below.



(Figure 7) – CNN 1's Classifying during a Live Match

The Second Convolutional Neural Network

Now that I'm able to know which cards my opponent is using. The next step would be to figure out *which* card is placed. If I'm able to know which cards my opponent is playing, then I'll be able to constantly keep track of my opponent's card cycle and elixir.

To do this, I created a 2nd CNN model. While the first model is responsible for learning what cards the opponent has, this second model will be responsible for knowing *which* card is being played, (card 1, card 2, card 3, card 4, card 5, card 6, card 7, card 8).

The key feature to note here is that while a player is in the "spectator" view, cards that are played have a small animation to them. See below for some examples.

(Figure 8) – Card Placement Animation

My original idea for training the second CNN model would be to just look at the entire image, and just have it train on which card flashes. This wasn't a good idea for the following reasons:

1) Cards only flash for a small amount of time. This would require me to feed the model lots of images per second to make sure it doesn't miss the card placement.
2) What if they play 2 cards at once? This would mean 2 cards would flash, which means I'll have to design a complex CNN multiclass classifier (to detect multiple objects in one image).
3) Each card has a unique design, this would mean I would have to train the model on *lots* of training data. I would have to make sure that the model learns how each card looks when it flashes.

My 2nd idea for training this new CNN model would be to crop the image into 8 individual images, exactly like my first CNN model. Then look at each card to see if it's flashing. Here's how this idea compares to my original one:

1) Doesn't solve the issue of how the card flashes for such a small amount of time.
2) This idea solves the issue of playing 2 cards at once, since we look at each card separately.
3) Doesn't solve the issue of unique design and flashes.

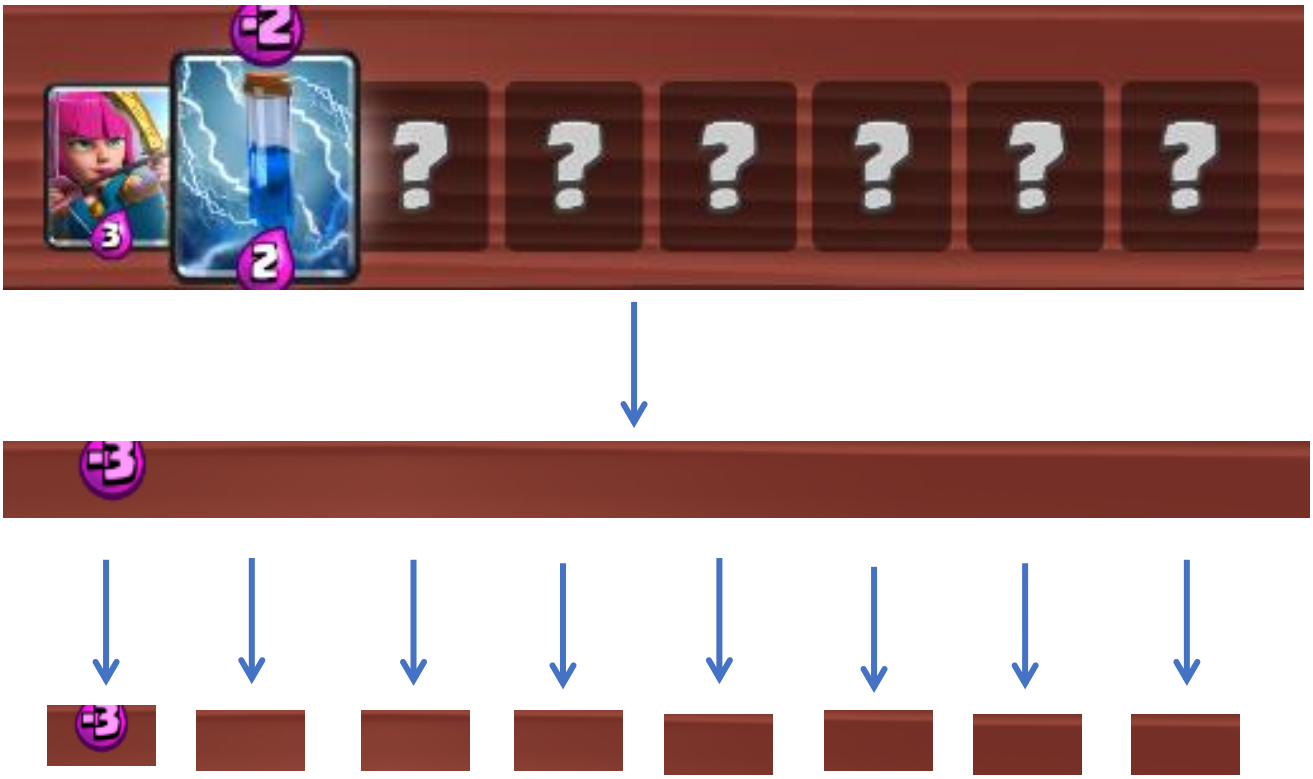
This idea only solved one of my problems. However, I found another solution. I noticed that when a card is played, the purple elixir icon shows up and flies upwards. This icon lasts for a while too. My 3rd idea for training this new CNN model would be to crop the image *above* the 8

cards, and then divide that empty area into 8 more images. This is a great solution to these 3 problems because:

1) Instead of looking at the quick flash, we can look at the purple elixir symbol, which lasts longer.
2) I'll have an image of the area above *each* card. This idea solves the issue of playing 2 cards at once, because I'll consider each one individually.
3) I don't have to worry about the design of the card and its flash, because I am just looking for the purple symbol now!

The next step would be to gather my training data. My 1st CNN model simply used all the default card images as training data. For my 2nd CNN model, I have to gather the training data myself. To do this, I just played some matches, and every second I took a screenshot. I would then go back through the screenshots and delete the ones that didn't have any meaning (no cards were being played). I made a small program to help with gathering and classifying the training data.

After collecting the training data, it was time to train my 2nd CNN model. It followed a similar process as the first one.

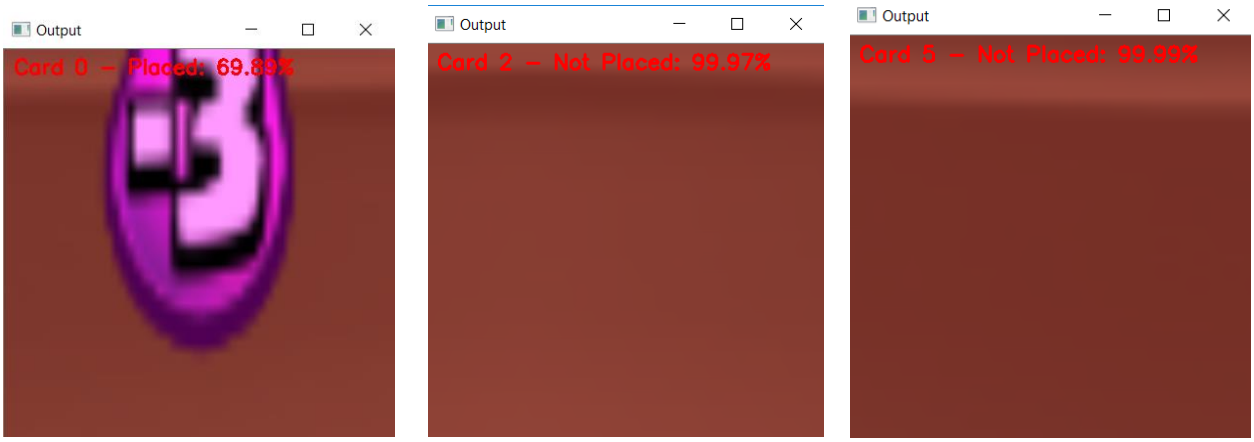


(Figure 9) – CNN 2's Training/Testing Data, what we want to classify/recognize

After training the 2nd CNN model, we can go ahead and see if it performs well. The input to this 2nd CNN model is the same as the input to the 1st CNN model, which would be the image. The output for the 2nd CNN model is a bit different. Instead of having 87 different percentages, we'll have 2.

The first percent will be the confidence for if a card hasn't been placed, and the second percent will be the confidence for if a card has been placed. For example, let's say the CNN model gave us [30%, 70%]. This would represent that our CNN model is 30% sure that a card wasn't placed, and 70% sure that a card was placed. The one that has the biggest percentage/confidence level will the model's classification of the image. In this example, the 70% is bigger, so the model has classified that image as "Card Placed"

Here are some results from feeding the above image into the 2nd CNN model.



(Figure 10) – CNN 2's Classification on Testing Data.

As shown above, we can see that the CNN model is able to classify these images correctly. With this CNN model, I can watch a live match and have my program keep track of *which* cards my opponent is playing. (This is done by feeding in screenshots every second or so) See below.

(Figure 11) – CNN 2's Classifying during a Live Match

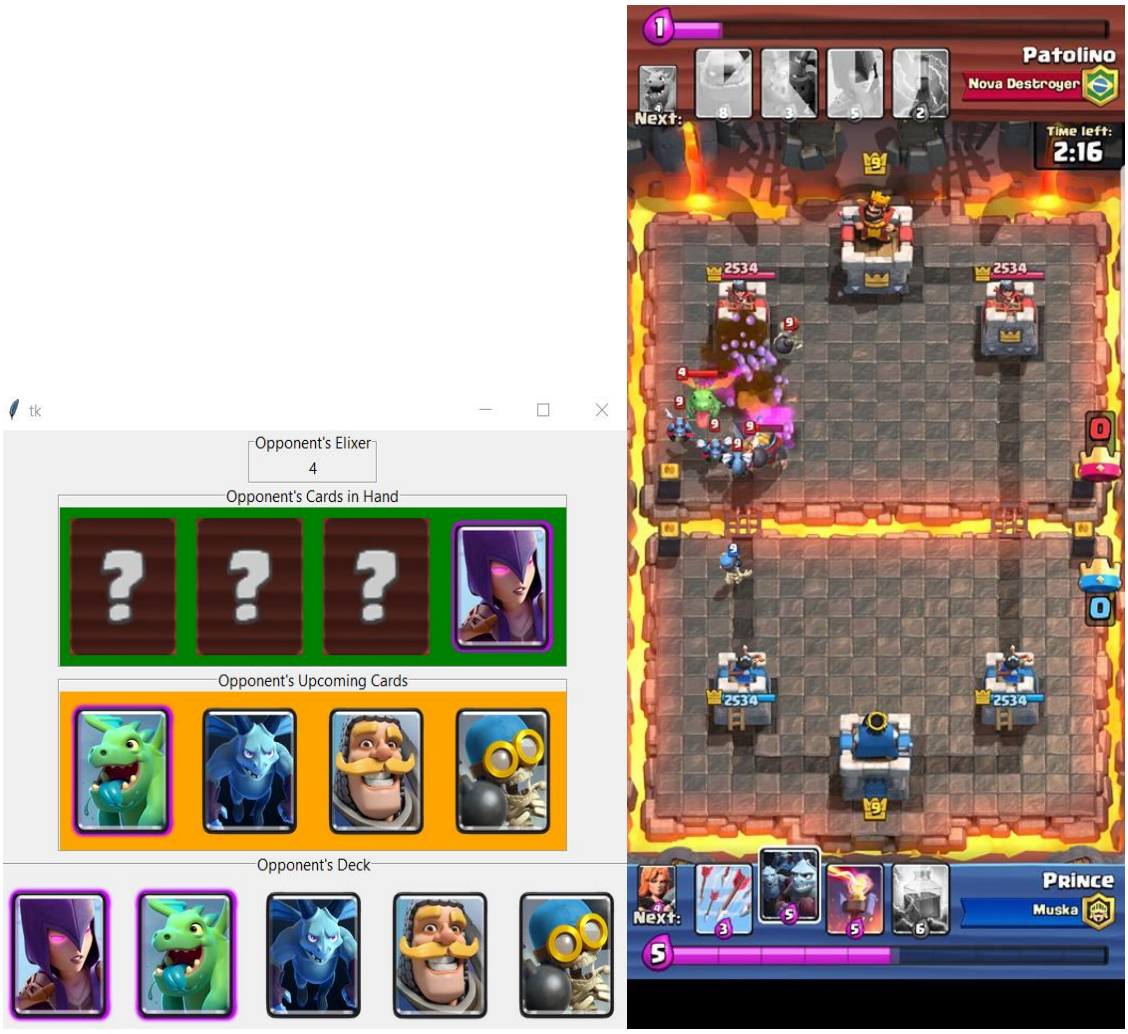
Putting Everything Together

Once both CNN's were complete and trained, I was able to combine both of them to make a Clash Royale Assistant. Using both of these CNN's, I'm able to:

1) Show opponent's cards in current hand
2) Show opponent's upcoming cards (to the hand)
3) Show opponent's elixir

I'm able to use the 2nd CNN to know which card was placed, and then I'm able to use the 1st CNN to know what card was placed. I have a virtual deck that keeps track of the opponent's order of cards. The top 4 cards of this virtual deck are the cards in the opponent's hand, and the bottom 4 cards are the one's that are going to soon be in the opponent's hand. As for counting elixir, I have a dictionary of cards. If I see that the "Archers" card was played, then I'll look in the dictionary for "Archers" and see that it's 3 elixir. I add elixir overtime and subtract elixir when cards are placed.

Here's some images of the Clash Royale Assistant in action:



(Figure 12) – Clash Royale Assistant and screenshot of match replay.

As seen in the above figures, the Clash Royale Assistant correctly predicted what cards the opponent has in hand. (A “Witch” + 3 cards that weren’t showed yet in the game). Match replays show both player’s cards in hands, so we can go back to the match replay to verify that our Clash Royale Assistant is working. The opponent’s cards can be seen at the top of the match replay image.

Here’s another example:

As shown above, the Clash Royale Assistant correctly displays the cards in the opponent's hand. As well as the next card to come into the hand.

<u>Difficulties in creating Clash Royale Assistant</u>

1) Getting used to some of the API/libraries, such as OpenCV (for image preprocessing), Keras/TensorFlow (for the CNN's), and tkinter (for the GUI).
2) The GUI. I'm not very great when it comes to front-end design, but I feel like my GUI is organized well. It shows elixir, cards in hand, upcoming cards, and opponent's deck.
3) Combining both CNN's together. One problem I faced was dealing with moving cards to the back of the deck *as* they were revealed. When a card is played for the first time. The 1st CNN is responsible for classifying the card. However, if it tries to classify the card right when it's revealed, then the model will be looking at a flashing image of the card.

This flashing card image can make the model think it's a different card. To solve this, I made sure that the model classifies an image as a specific card *2 times in a row*. In other words, it must classify a card correctly in 2 different screenshots. This solves the problem of misclassifying a card due to the card animation, (since the animation won't last more than one second). Since I have to wait for a second screenshot to classify a card, that means I have to hold off on moving it to the back of the deck, since I don't know what card it is yet. To solve this issue, I just drop whatever card number, that is in the middle of being classified, into a "pending" list. This "pending" list is responsible for moving cards to the back of the deck once they are classified for the first time. I also have to consider the change in elixir while I wait for a card to be classified.

Credits and References

A great introduction to CNN's using Keras: https://www.pyimagesearch.com/2017/12/11/image-classification-with-keras-and-deep-learning/

Keras Website: https://keras.io/

Beta Testers for Clash Royale Assistant: Twinkle Mistry & Vraj Mistry

OpenCV Website: https://opencv.org/

Clash Royale Creators: http://supercell.com/en/

My GitHub: https://github.com/AmarSaini

Special Thanks for some general advice on Neural Networks: Brian Tsan