**Capstone 2: Starcraft-2-Unit-Recognition Milestone 1**

* What is the problem you want to solve?

Starcraft 2 is an extremely fast paced RTS game where players from around the world versus one another to improve in skill and ranking. Starcraft 2 started the era of video game streaming during 2010 to 2012 by dominating the two most popular streaming sites at the time: own3d.tv and twitch.tv formerly known as justin.tv. Twitch.tv is now the most popular US video game streaming site with little competition in it’s way and allows for streamers to interact with the millions of unique viewers that visit the site to watch video game tournaments, ladder climbing matches, speed runs, and many other video game centric content.

Using deep learning, we can create a model that recognizes Starcraft 2 units and structures as a way to create an interactive stream. If the viewer decides to hover over one of the units that has been recognized by the model, we can display the unit’s details and background information giving unfamiliar viewers a deeper understanding of the game. This model can be used to handle in-game replays, live streams, and other video content.

* Who is your client and why do they care about this problem? In other words, what will your client do or decide based on your analysis that they wouldn’t have done otherwise?

This project can benefit the viewers, content providers, and the content hosting companies by creating a more interactive environment for their selected audience. This interactive environment is something that only Twitch.tv provides at the current moment and its usage is very limited (Hearthstone is one of the few games that has stream overlays).

* What data are you using? How will you acquire the data?

The data will be curated through in-game screenshots and I will manually label the images. The first step is to collect the images to be used in game. Starcraft 2 has well over 100 different units and structures so I will isolate a couple of them to reduce the amount of work necessary to collect and label the data. The second step is to label all of the data. I will be using [labelImg](https://github.com/tzutalin/labelImg) to annotate the units I wish to deal with.The figure below is a sample of an image being used for analysis.



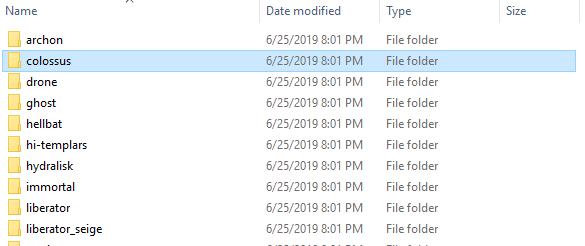
The screenshot is parsed using [labelImg](https://github.com/tzutalin/labelImg) to annotate the specific unit as shown on the figure below.

’

After the annotation, the tool will generate an xml file that saves the binding box pixel location of the unit and it’s class. We generate a comma-separated values file from all of the xml files (one for each of the screenshots) to create a more basic dataset to work with. A sample dataset is shown below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| filename | width | height | class | xmin | ymin | xmax | ymax |
| Screenshot2019-06-17 18\_21\_27.jpg | 1920 | 1080 | overlord | 525 | 655 | 651 | 806 |

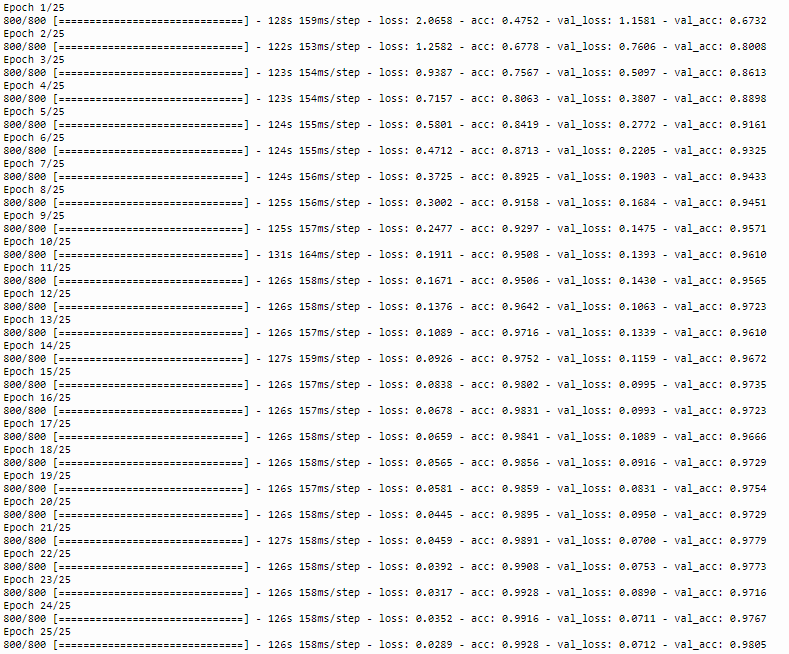
I wrote a script to parse the screenshots to feed into my MobileNet model through keras’ ImageDataGenerator object. The script will take take the .csv file and parse through it and crop the image before renaming and shelving it for the next step. Since the ImageDataGenerator takes images and categorizes them using their folder structure, the script will generate the proper folder structures for the test and training data. Image shown below is the specified folder structure:



The ImageDataGenerator object will add subtle effects to each image during the model’s training phase to improve and greatly benefit models that have very low amounts of data to work with. These effects can be seen in the image on the side where the image is flipped, shifted, blurred, or rotated.

The model that I have decided to use was [MobileNet](https://github.com/keras-team/keras-applications) because MobileNet neural networks have a good speed to accuracy ratio and since we plan on using this with streaming data such as desktop inputs or video inputs, this model is more preferable than others. Keras’ includes an implementation of MobileNet V2 which can be overwritten to append an input and output layer (to match the picture resolutions being fed in and the classes we are using).

The training for MobileNet is done through a GTX 1070 MaxQ and resulted in an impressive validation accuracy of 98%. This is extraordinary because many of the classes had less than twenty sets of images. After training the model, I saved out the weights and the model settings to be used for a later date.



Our results yield near 100% accuracy with any of the images that we test with. The image below shows a snippet of the astounding accuracy of the model.

