Champion Annotator

CSSE 463 Dr Boutell
Timothy McDaniel Thomas Dil Benjamin Kulbago Daniel Schnipke
18 February 2017

Abstract

Without a proper tutorial or manual, learning how to play the video game *League of Legends* can be time-consuming and frustrating. Using image recognition, it is possible to make this learning process easier, locating champions in an image or video and annotating them with their name. The recognition process can be completed with relative success, 78.7% accuracy, using a basic neural net. The only limiting factor in the accuracy of recognition is time, which we suggest should be used to run a hyperspectral image classifier on a more robust system than the Rose-Hulman student laptops to more accurately identify the different classes.

Github Repository: https://github.com/schnip/champAnn/

Introduction/Problem Statement

League of Legends, a video game produced by Riot Games, is a hugely popular MOBA title with over 100 million monthly players, according to an interview with developers last fall [1]. Naturally, the game's playerbase has created a plethora of online content, especially on YouTube. There is no shortage of comedic, informative, or analytical content to be found in the form of YouTube videos. Because of this, the highest barrier to entry for a new player is not a lack of information on the game, but a lack of being able to comprehend it. Visually, *League of Legends* is a noisy game, with gaudy character design, a vividly cartoonish art style, and an array of flashy special effects that, all bundled together, can be hard to look at. A person who does not play the game or is new to it will have difficulty understanding what is on their screen.

What we have developed for this project is an annotator for the *League of Legends* game. To help someone understand the game, our system will draw a box around each champion on the screen and label them with their full name for every frame of an in-game video. With a bright red box around each character and their name labeled above their head, the game is much easier to understand and follow.

Fig 1. Cropped screenshot from a *League of Legends* World Championship 2016 game, featuring 9 individual champions. A viewer unfamiliar with or new to the game would have a difficult time identifying which champions are currently shown.

Unfortunately, while creating this project, we were unable to find a controlled, in-game data set of each *League of Legends* champion that we could use as training data, so we have created our own for this project. We needed a complete visual description of each champion, preferably with as little noise or few special effects as practical.

Literature Review

SVM Multiclass [2]: This paper covered a method of quickly using an SVM machine to distinguish between multiple classes. Because we have over 100 classes to classify between, this paper could have showed us how to classify champions quickly with the limited processing power available to us. However, we decided to use a neural net for our classifier instead of an SVM, so the information from this paper was not used.

Neural Net Multiclass [3]: This paper discusses a method to speed up the training of a neural net. The paper proposes adjusting the different weights on the net's nodes to allow the easier classes to converge faster. Being able to retrain a neural net would have benefited our project to give us faster results as we experimented with different types of feature extraction, but we did not have time to implement this feature. Instead, this paper gave us ideas of how to go about implementing a simpler version a multi-class neural net.

Hyperspectral Image Classification [4]: This paper shows a way of combining multiple kernels for feature extraction. This would allow a system to use both color and edgel information to train a classifier. Because of how complex the shapes and color schemes that make up a *League of Legends* champion are, it is difficult to completely describe an image of them using just one type

of feature. This would have made working in-game cosmetics into our system somewhat more practical because many cosmetic skins are simple color shifts from the champion's base model and carry the same edge information as the base skin. We cover this particular piece of literature in our Future Work section. We would have liked to add it, but we ran out of time.

Multiple Kernel Learning [5]: Another way of using multiple kernels to train a classifier, this method specializes in distinguishing between two similar classes that may only be separated by a single feature threshold. Working out how to best use the computations that this method uses would have taken some time, but its use would have benefited our system's accuracy.

Process

Data Gathering

Without a pre-made data set available to use for training, we went about creating our own. The game has 136 champions in its roster, allied map turrets, and any additional sections of the map that our health bar finder thought were players. This meant that we had 138 different classes to gather data for. This was accomplished manually by launching *League of Legends* through Steam, loading into a custom game, and using the Steam screenshot feature (default F12) to take images. This process was repeated for every champion available to our team. We took images of each champion standing still and in motion and facing relative North, Northeast, East, Southeast, South, Southwest, West, and Northwest in-game. This led us to take at minimum sixteen screenshots for each champion, but we took more screenshots if we felt that more orientations were needed or that a champion's walk cycle animation needed more information to fully capture.

The training data was simple. Each champion was shown more or less alone, with no terrain, players, or AI controlled minions nearby, and we avoided having additional special effects in the picture. If a champion had multiple states, we captured each fully, as we would another champion. For example, when she is out of combat, Evelynn becomes stealthed and to the player and her allies, the appearance changes. Udyr changes between 4 different states at will. For data gathering purposes, we treated visible and invisible Evelynn as 2 different champions to take screenshots of and Udyr as 4 different champions.

Fig 2. Udyr, The Spirit Walker, shown in Tiger, Turtle, Bear, and Phoenix stances. For data gathering, we treated each of these as a distinct champion.

Once we ingested the data, every state of each state was treated as the same champion, but we took full sets of screenshots for each champion state. If a champion had an ability passive that produced special effects on the champion and the champion would have that passive special effect on them for the entire game, then we would level up that ability and take our screenshots with it. For example, Irelia's W ability, Hiten Style, produces yellow particle effects at the center of the blades she carries from the moment she gains access to it until the game ends, so her data was taken with that ability leveled. Cosmetic skins that change the appearance of the champion were ignored for this data set. All told, we used 2,117 screenshots of data.

Fig 3. Irelia, The Will of The Blades, with Hiten Style active. This ability adds a constant green-yellow effect which persists throughout the game.

Data Ingestion

Manual data gathering also meant manual data ingestion. Our data gathering resulted in an adequate amount of images for each champion, but in order to effectively train our SVM, we needed to sort the images into individual champion folders to keep track of. Given that our champion detection was still a work in progress, we put together a function to partially automate the process. The function displayed two windows, as shown in Figure 4.

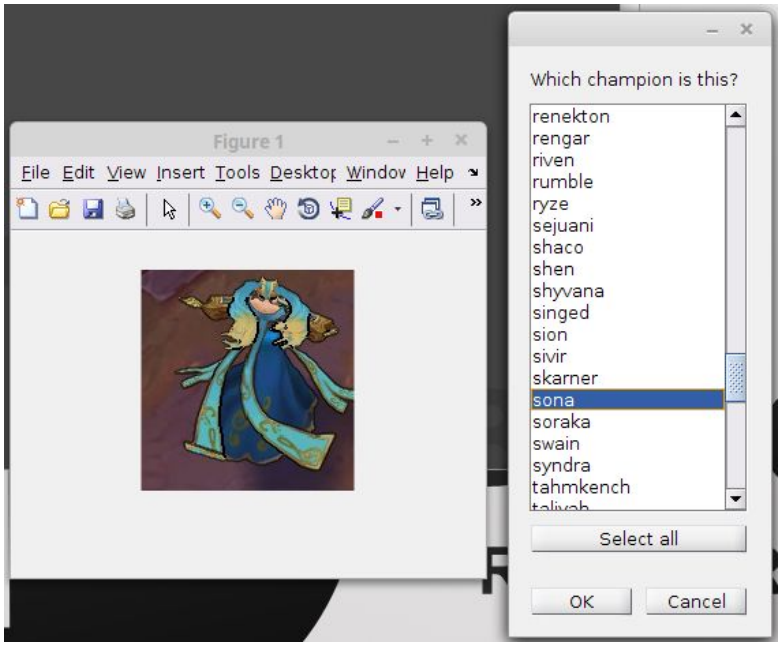

Fig 4. Sona, Maven of the Strings, shown in the bounding box created below her health bar and the list of champions used for data ingestion.

The content of the window on the left is determined by processing the raw data screenshots. Keeping in mind the ultimate goal of wanting to extract features only for the champions in each image, we designed our function to find regions of the image which fit within saturation and value thresholds determined by inspecting a few sample images with imtool().

We also added restrictions to the height and width of each region. The end result was the isolation of regions denoting the location of champion health bars such as the one in F. The functions used this location as a reference point to draw a box sized 156 by 165 pixels directly beneath it. We chose this size as it would reliably contain a majority, if not all of the champion in question. The function takes in a single screenshot from the ingestfolder folder, copies the pixels from the box bounded above, and moves the screenshot to the postgestfolder folder.

The new cropped image is shown in the window on the left. The content of the window on the right is set by a text file containing the names of all 136 champions as well as the “turret” and “other” tag. Together the two windows serve as the UI for manual ingestion. After identifying the champion in the image, we selected the champion’s name on the list and pressed the “ok” button. The function then saves the screenshot as a new image in the folder matching the name we selected previously, under the trainingimg directory. The function then iterates to the next photo, continuing the process until no more images are left in ingestfolder.

Feature Extraction

We attempted to use three different methods of feature extraction throughout the course of our project. The first method employed the feature extraction method from the sunset project earlier in this quarter, but on the RGB channels instead of HSV. This method turned each image into a 7 pixel by 7 pixel picture with each pixel representing the average of the values in its corresponding 1/49th of the original image. We also looked at using k-means on our image set, but we quickly discovered that the background dirt, grass, and stone behind and below the champions influenced the final means result too much. This was dealt with by manually creating thresholds based on the background pixels and setting anything within that threshold to black. After the background was scrubbed away, the k-means was able to successfully run; however, it was prohibitively slow, so we were not able to run even 2% of our images through it in an hour. In order to use our time more wisely, this approach was ultimately abandoned. Our final method integrated the background remover from the k-means attempt with our first method. The background was changed to black, then we applied the 7 by 7 grid of means to the image. While we had high hopes for this method, in practice it made bright noise (such as particles, spells, and projectiles) have a larger impact on the results, making even small changes between frames in a video more likely to change classification. This resulted in the final output video having bounding boxes which quickly and constantly flicked between different classes.

Training and Classification

For training, we used MATLAB's built in implementation of neural nets. For inputs, we used all the information from the feature detection, which was 147 for the feature extraction that we used most of the time. For the hidden layers, we used three layers of 100 neurons each. While we considered using larger and more layers, we ran into the issue that anything larger took an extremely long time to train; therefore, we did not go larger. While training, the output was a vector with length of the number of classes. In training, the correct class was set to one, all others were set to zero. For classification, we ran the same feature detection on the image to be classified. We then ran these features through the neural net, and the took the highest value returned to be the one that we called as the class that was identified.

Video Output

As the primary output of our project, we have applied our classifier to a video downloaded from YouTube. Using the same technique we used on our training data, we located champions by finding their health bars in every frame of video and ran the space below through our classifier. Once we had a name for the champion, we added a bounding box around it and used MATLAB's insertText() tool to add the champion's name above them.

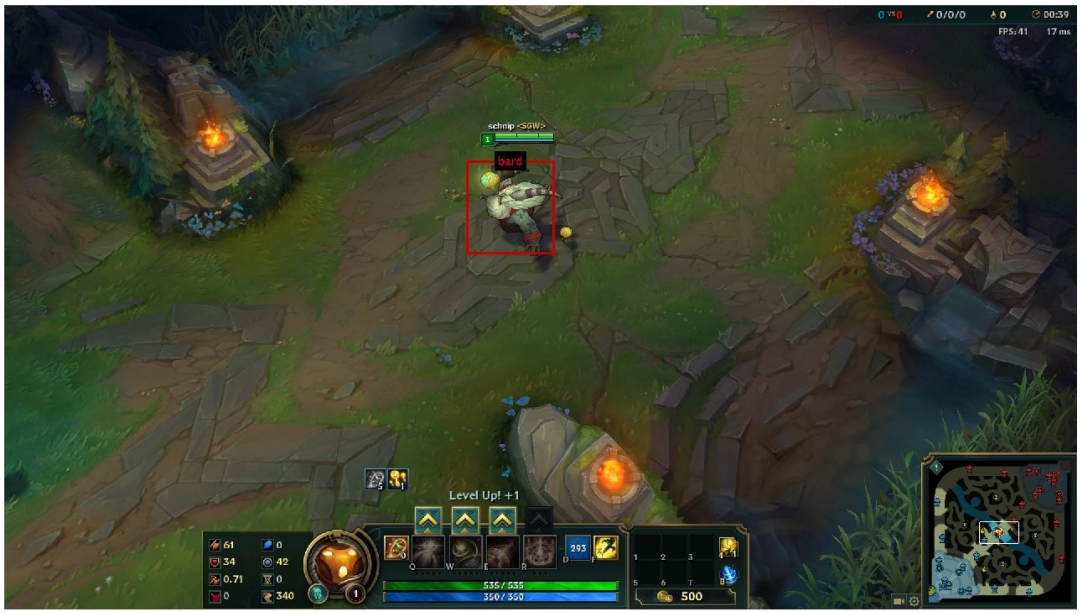

Fig 5. Bard, The Wandering Caretaker, shown annotated after classification.

In our presentation, we showed a video from a montage of Lee “Faker” Sang-hyeok, a player widely considered to be the best in the world. We selected one of his videos because he is known for not using cosmetic reskins of the champions he plays. That particular video was used because many of his teammates and enemies were not using skins and because it showcased him playing multiple champions. These two factors gave us a video of in-game activity to use as an appropriate test given our starting assumptions.

Experimental Setup and Results

In addition to the YouTube video, we also sought to quantify our accuracy. We pulled 474 screenshots, 4 from each champion folder, into a new directory. We then ran these images through our SVM in order to verify that our training was capable of finding the champions that we were training for. This test resulted in a **78.7%** accuracy, correctly denoting 373 of the 474 images with the expected champion name. It is worth noting that these test images represented an ideal case, without the noise, chaos, and overlapping models that are present in an actual game of *League of Legends*. However, because these 474 images were not used to train the data, we can treat this accuracy as a result of test data.

Table 1. Final results after running a test set of images through the classifier

Total Images	Classified Correctly	Classified Incorrectly
474	373	101

Video showing the results of our final feature extraction and classifier:
https://www.youtube.com/watch?v=SN-z4tDE8YM&feature=youtu.be

Key Challenges

The largest challenge presented to us was the lack of readily available data in our chosen topic. Over the last three weeks, we have spent much of our project work time clicking our way through the screenshot process for every champion available to us or manually ingesting the data. Given more time, we would still prefer to have more data per champion, but we feel that we would have had better results with more time to gather data.

Once we began work on ingestion, we needed a way to locate a champion in screenshot. Luckily, big or small, evil or strange, each champion has one feature in common: a health bar. We used value thresholding to create a binary image mask of each of the three colors frequently used for health in *League of Legends* and [0, 0, 0] black to represent missing health. Blue represents allied champions, red indicates enemies, and green signifies the player's champion. Initially, we thought that this would be a simple task because unlike images of the real world, we could use raw pixel data to identify the exact colors that the game uses to represent health or lack of health, but *League of Legends* uses a gradient, seen in Fig 6 so that the top of the health bar is brighter than the bottom. We solved this by searching for pixels that were black or in a range of red, green, and blue values, but this frequently picked up pieces of the HUD or terrain. Each image would have hundreds of objects that could potentially be health bars. Initially, we checked every one of these for width and height to see if they matched the dimensions that we were looking for. This ran slowly, and could take up to six seconds to find a new champion during data ingestion. To speed up the computing process, our team applied a min and max size threshold after using bwlabel to filter out all pieces of green or red terrain. All pieces that passed the color thresholding but were smaller than 700 pixels or larger than 1600 pixels were ignored. As a result, our system found health bars in an image almost instantly and the slowest part of the ingestion process was the amount of time that it took a team member to click on a champion's name.

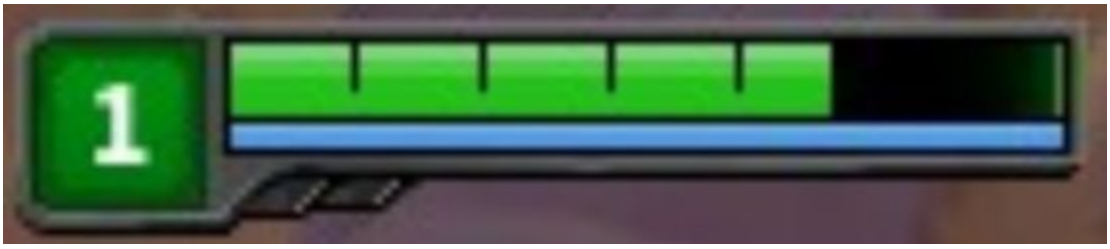

Fig 6. The gradient in a champion's health bar can be clearly seen up close.

Feature extraction also gave us quite a bit of trouble in this project. We used the averaging 7 by 7 grid from the Sunset Detector project at first because we already had it written and implementing it was simple, but a large amount of our training data's area shows the background terrain of the map. The ground changes frequently in *League of Legends*. Many areas of Summoner's Rift, the main map in *League of Legends*, have grassy areas with patches of dirt, but champions will frequently pass through the river or fight on the stone paved platforms

of their base. We wanted to keep our feature extraction separate from the location of the screenshot. Turns out that the background map of Summoner's Rift contains only muted colors and with some thresholding, we were able to remove most of it from our training data. From here, K-means seemed like the most logical method to use, but was too slow to run on the twenty-three thousand images. We found that applying the 7 by 7 grid to each piece of training data after thresholding out the background gave us an optimal balance of speed and quality.

Discussion and Future Work

In order to improve the success rate of our champion annotator, we have a number of changes we would suggest to implement for future iterations of this project. Although we had an average of 173 training images for each champion, other research has used 3 times that number of images [6]. As a result, we would first seek to augment our training data set with additional unique images. These images would be taken from multiple locations around the map to assist with the training process and reduce the effect of background textures in the features. With enough data, we could switch to using a convolutional neural net which would improve our accuracy and speed.

Potential short term improvements in accuracy could also be drawn from simply increasing the number of neurons and hidden layers in our neural net. A long term solution, however, would involve the implementation of hyperspectral image classification (HIC). HIC allows the use of more than one kernel in the training of a neural net and could be supplanted within a deep neural net. On top of both of these, we could add motion tracking to our video processing code, programming the net to remember what a champion was last classified as and increasing the weight of the selection of that champion in the following frames.

From a functionality point of view, we would recommend as a final touch to the project no longer excluding the cosmetic skins of each champion. Although some players, such as the aforementioned "Faker", regularly play without these additional graphics, it is very uncommon that a game is played in which none of the players involved are not using one. The dramatic nature to which some of these skins affect the champion features warranted us excluding them for basic classification, but would be necessary for a finalized Champion Annotator. Additionally, we would look to further refine our 156 by 165 input images by thresholding out standard special effects produced by abilities or items used in game which are standard across all champions.

Ultimately, the main factor limiting us from implementing our suggestions is time. Each of our suggestions greatly increases the amount of time which would be spent either gathering data or allowing our computers to run extensive operations in MATLAB.

References

[1] Phil Kollar, In Off-topic by James Elliott, In Off-topic by Cobaltios, In Xbox One by Donald Michan, In Off-topic by LittleChatelain, and In Pokémon Sun/Pokémon Moon by KajunBowser. "The past, present and future of League of Legends studio Riot Games." *Polygon*. N.p., 13 Sept. 2016. Web. 18 Feb. 2017.

[2] Wang Z., Xue, X. Multi-Class Support Vector Machine. Y. Ma and G. Guo (eds.), Support Vector Machines Applications, Springer International Publishing Switzerland 2014.

[3] Anand, Rangachari; Mehrotra, Kishan; Mohan, Chilukuri K.; and Ranka, Sanjay, "An Efficient Neural Algorithm for the Multiclass Problem" (1991). Electrical Engineering and Computer Science Technical Reports. Paper 136.

[4] Xin Jun Li, Paolo Gamba, Jose Bioucas-Dias, Liangpei Zhang, Jon Benediktsson, Antonio Plaza. Multiple Feature Learning for Hyperspectral Image Classification. IEE Transactions on Geoscience and Remote Sensing, 53(3), p. 1592-1606, March 2015.

[5] T. Joutou and K. Yanai, "A FOOD IMAGE RECOGNITION SYSTEM WITH MULTIPLE KERNEL LEARNING," IEEE, 2009 16th IEEE International Conference on Image Processing (ICIP), Nov. 10, 2009. [Online]. Available: http://ai2-s2-pdfs.s3.amazonaws.com/a10c/bb12a63f39447e766863ad158f0fcf162f1c.pdf.

[6] Matthew Boutell, Jiebo Luo, and Robert T. Gray. Sunset scene classification using simulated image recomposition. IEEE International Conference on Multimedia and Expo, Baltimore, MD, July 2003.Matthew Boutell, Jiebo Luo, and Robert T. Gray. Sunset scene classification using simulated image recomposition. IEEE International Conference on Multimedia and Expo, Baltimore, MD, July 2003.