Teamfight Tactics Win/Lose Predictor

Min Gyu Woo
mgwoo@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Sijie Xu
s362xu@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

We present a novel end-to-end pipeline for predicting the win-rate of battles in an autochess game called Teamfight Tactics. The absence of data can be solved by collecting big data from content creators who play the game. These videos are processed using video/image-level segmentation. The features are fed into a customized state-of-the-art model called ResNet to predict high-level features like the type of unit and the background. Finally, the subsequent features are used to predict the overall win-rate given two states. We hope this novel process can open paths in improving artificial intelligent problems for more complicated tactical games.

Supplementary Material: Code and data available at Github: Teamfight Tactics AI

Introduction

Machine learning has been proven to be great at finding patterns that are non-easily captured by traditional techniques. Because of this, machine learning and artificial intelligence have significantly been adopted by many fields such as biomedical research, finance, and manufacturing. Recently, the gaming industry is shaken by artificial intelligence. Two great examples would be AlphaGo, which beat Lee Sedol, a world-renown player of Go (Silver et al. 2016), and DeepChess (David, Netanyahu, and Wolf 2016), which reached grandmaster-level (top 0.3% of players (contributor 2021). This hints toward the idea that artificial intelligence can be more advanced in playing tactical games than humans. However, one crucial limitation to all machine learning algorithms is the lack of data. Since games such as Chess and Go have been played for many years, the amount of experience and strategies are well-known and readily available. In recent years, a new genre of strategy games emerged: Auto Chess games.

Generally, there are eight players facing off, and users have the ability to place their units on a grid (similar to a chess grid). Every round, a random pair of users are faced off against each other, and the units fight automatically. These series of mini-battles are played out until the health points of all but one individual are depleted. There are multiple factors that determine who wins the mini-battles; these include items that boost individual units, special abilities that a unit may have, placement of a unit, etc. Recently in 2019, a new auto chess game developed by Riot Games is released: Teamfight Tactics. In this paper, we deploy a sequence of neural networks: A convolutional neural network to extract information from gameplay footage such as unit type, trait, and position. Then, a fully-connected neural network will predict the probability of one board winning against another.

Contribution

- First to train deep neural network algorithm on auto-chess game based on video data.
- Collected big data on YouTube and performed video/image-level segmentation.
- Modified the implementation of image classification workflow, which saved human labor and time on ground truth labeling.
- Adopted the Word2vec algorithm to train champion embedding vectors to significantly increase the training accuracy.

Related Work

1. Attempting to use state-of-the-art models in order to classify the champions, items, and augments. A customized version of ResNet (He et al. 2016) will be used to implement the classifier.
2. Similar approach comparing different state-of-the-art models. The data here is gathered using the RiotGames API, which only produces data on the final battle of a player. (Casian and Zannato 2020)

Methodology

In this section, we will be focusing on the implementation of our model and the entire data processing workflow. The workflow overview can be found in the image below 1:

Collecting Data

The data is scraped from popular Teamfight Tactics Youtubers, recorded in the file Youtuber_list.csv. The video URLs are fetched from the YouTube video streams. Since the

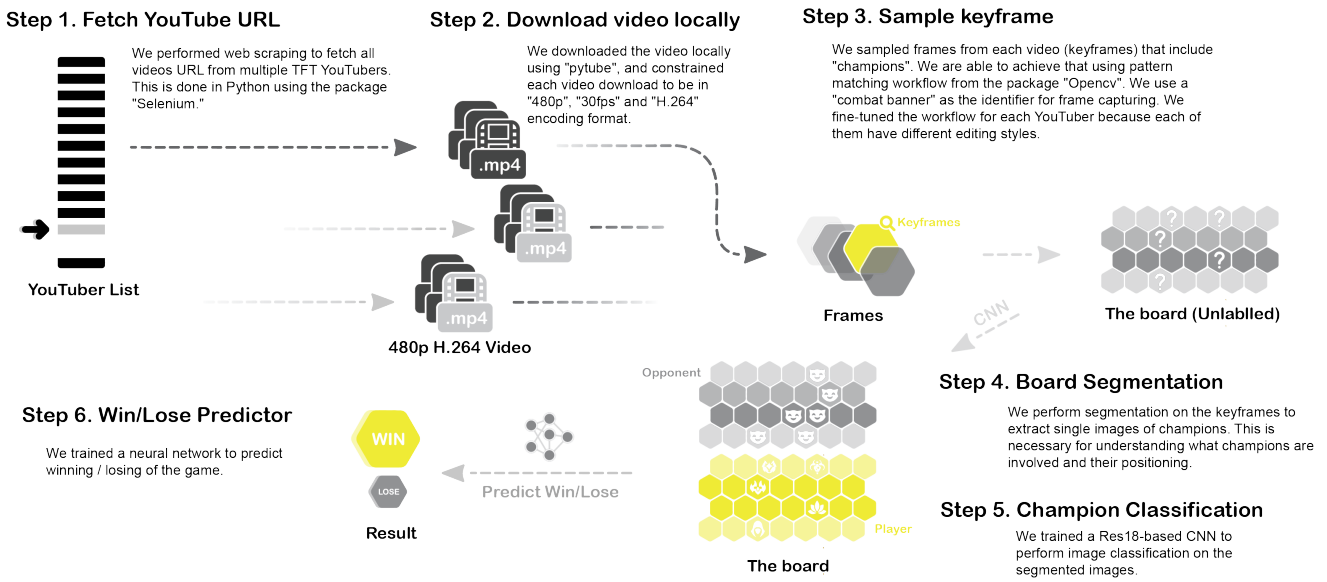

Figure 1: Overall Workflow

YouTube web page uses lazy loading, namely, the webpage does not fully load until the bottom of the page is reached, the package Selenium (Simon 2022) and Web Driver Manager (Pirogov 2022) is used to mimic page scrolling. Then, using the Pytube (Shubham 2021) package, the video information is fetched. This includes video titles and keywords. Refer to Video_Download.ipynb for detailed workflow. The video list is filtered to include only videos with relevant topics (i.e., Teamfight Tactic games). The video is then downloaded with the constraint that each video has to be in ”480p”, ”30fps,” and ”H.264” encoding format.

Video Segmentation

After the video downloading step, frames are sampled from each video (Keyframes), including Champions and Champions Layout. Using the OpenCV (Bradski 2000) package, the pattern matching workflow is followed. The ”combat banner” is used to identify the frame to capture since it always appears on the center of the screen before each ”combat” round. Due to different YouTubers having different editing styles, the workflow is fine-tuned to be applicable in general cases. The ”Combat” pattern is shown below. 2:

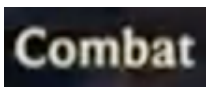

Figure 2: ”Combat” flag

For example, this image 3 shows the time right before two players face off. The “Combat” in the top middle of the image denotes the beginning of matches. Hence, we can use the pattern “Combat” as our key pattern. After capturing the keyframe, around 4.5 seconds must pass for all the units to be properly visible on the board. The screen-capture will be considered the keyframes.

Figure 3: TFT Gameplay

The pattern matching algorithm chosen is the ”Normalized Correlation Coefficient Method” described on the OpenCV match template. This method produces a number between $(0, 100)$ describing the degree of correlation between images and the pattern. OpenCV then performs convolution operation on the entire image plane and finds the best matching point. In example 5, a correctly labelled keyframe has a score significantly higher than other random frames. Hence, we can use the score as a good feature to classify keyframes. A predetermined threshold can be used to determine which frames are keyframes.

Image Segmentation and Classification

The screen capture images are processed to return relevant information. These screenshots are cropped using a predefined grid to obtain images for each cells 6:

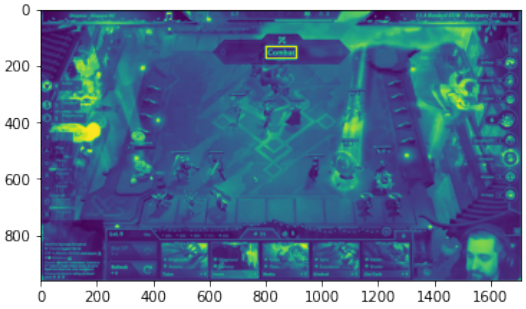

Figure 4: Identified Keyframe with match score = 0.99

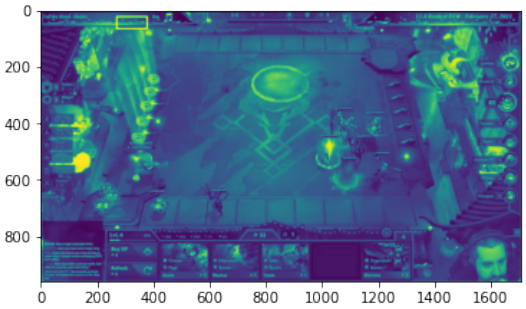

Figure 5: Random frame with match score = 0.54

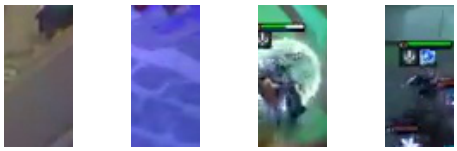

Figure 6: Sample Cropped Image

Information in each of these images are automatically selected using a trained CNN (Foret et al. 2020). The champion classifier used to classify each champion on the grid is based on a pre-trained Res-18 network on PyTorch. ResNet is a recent convolution neural network architecture that learns residual functions from the layer inputs for each of its layers (He et al. 2016). The detailed structure of ResNet is shown in the image below 7.

The major benefit of using a residual network in this study is that ResNet is easier to optimize compared to other CNN network (e.g. VGG). Additionally, since classifying computer generated images are taken from a similar angle, a shallow model is preferred. Hence the Res-18 architecture is the most applicable model architecture to use. Here are some adjustments that are made to the original Res-18 architecture:

- The output size changed from 1000 to 86 for the last fully connected layer, representing 85 different champions plus 1 background.
- Cropped grid image are used as inputs and one-hot representation of the champions as the targets. A custom PyTorch data-loader is developed to accommodate model training and evaluation.
- For model training, batch size of 4, learning rate of 0.01 and momentum of 0.9 is used with the Stochastic Gradient Descent optimizer.
- For this multi-class classification problem, the Cross Entropy Loss is used.
- For model evaluation, 20% of the total data-set is randomly sampled to perform accuracy testing. Additional analysis is done using the confusion matrix to check any biases in the model prediction.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 7: Residual Network

In the model training processes, it is unrealistic to label all of the examples due to the time constraints. Additionally, since more than 80% of the images cropped is representing the background, it will be time-consuming for a human to label from scratch. Hence, a "human-in-the-training-loop" method is used for improving model accuracy while minimizing time spent. The workflow is as follows:

1. Train model M_0 with initial labelled image batch $\mathbf{B}_0$; Create a test image batch $\mathbf{B}_t$ which includes every champion.
2. Repeat steps (3-6) until test accuracy on $\mathbf{B}_t$ is acceptable.
3. Use confusion matrix produced by $\mathbf{B}_t$ to identify biases in the model; Fetch a new image batch $\mathbf{B}_i$ with focus on fixing the model biases.
4. Use the last trained model M_{i-1} to make prediction of the current image batch $\mathbf{B}_i$.
5. Manually iterate through the images and correct the predictions of $\mathbf{B}_i$ to produce the ground truth.
6. Retrain the model using $\mathbf{B}_i$, obtain trained model M_i.

Using this method, 240 000 data points is labelled. Each data point contains champion name and their position on the board.

Champions Embedding

Champions embedding reflects the champions' contribution to the team and identifies the natural clustering of champions that share similar traits. Moreover, multiple traits can be carried out by a single champion. Hence, it is important to find a meaningful embedding to show the relationship between them. Similar to the Word2vec algorithm, our champions embedding are pre-trained by a "Champions2Vec" algorithm and refined by the Win/Lose Predictor model.

The Word2vec algorithm is used for learning word vectors based on the probability distribution of the words' appearances in each sentence. This technique is known for introducing large improvements in model accuracy, and it is a common practice in the field of natural language processing. (Mikolov et al. 2013) Using this method as inspiration, pre-trained champion vectors are produced with the hope of improving the Win/Lose predictor model.

In the implementation of Word2vec, champions' "name," "costs," and "traits" were used as the context. Different from the original content predicting the Word2vec algorithm, we only incorporate word pairs that involve the names and other attributes (ignoring the correlation between traits). The parameters used for the Champion2Vector algorithm are: embedding dimensions of 4, 120 epochs, and a learning rate of 0.01.

The output of the Champion2Vector algorithm are two embedding matrices U, V of sizes $(\|\mathbf{C}\|, \|\mathbf{E}\|)$, $(\|\mathbf{E}\|, \|\mathbf{C}\|)$ respectively, where $\mathbf{C}, \mathbf{E}$ are sets of champions and embedding vectors. One of the averages of matrices U, V can be used as the embedding matrix. In this implementation, only U is taken as the embedding matrix.

The Win/Lose Predictor Network

A general DNN (David, Netanyahu, and Wolf 2016) is used to predict the probability of winning. The model is compared with other possible supervised learning methods to determine the best model.

The structure of the DNN is shown below 8. Initially, each board of the data is of size $(4, 7)$ preserving the spatial information. In order to feed the data into the neural network, all of these boards are flattened such that the input is in a special format where the player's and opponent's boards are shown as one-dimensional tensors of length 28. For each board tensors $\mathbf{T}_b$, the $\mathbf{T}_b[i]$ is the one-hot representation of the champion on position $(i//7, i\%7)$ in the actual board. For example, $\mathbf{T}_b[8] = 2$ represents that there is a "Camille" (Champion with one-hot representation 2) on the second row, and second column.

Then the input is piped over an embedding layer where each one-hot representation of champions are mapped to a vector with an embedding size of 4. The embedding weight can be initialized by the Champion2Vector algorithm using the champion's traits. The neural network will continue to learn the embedding weight in the training process.

Moving on, the embedded board tensors (player's and opponent's) are flattened to separate one-dimensional vectors of size $28 \times 4 = 112$. Then the tensors are pipelined into a fully connected layer with an output layer size of 25. Note that the player's and opponent's boards are piped into the same fully connected layer separately. We then apply concatenation, flattening, and LeakyReLU activation layer on the two resulting tensors to create a one-dimensional vector of size 10. Then a fully connected layer is acted as the classifier to predict the outcome of the game.

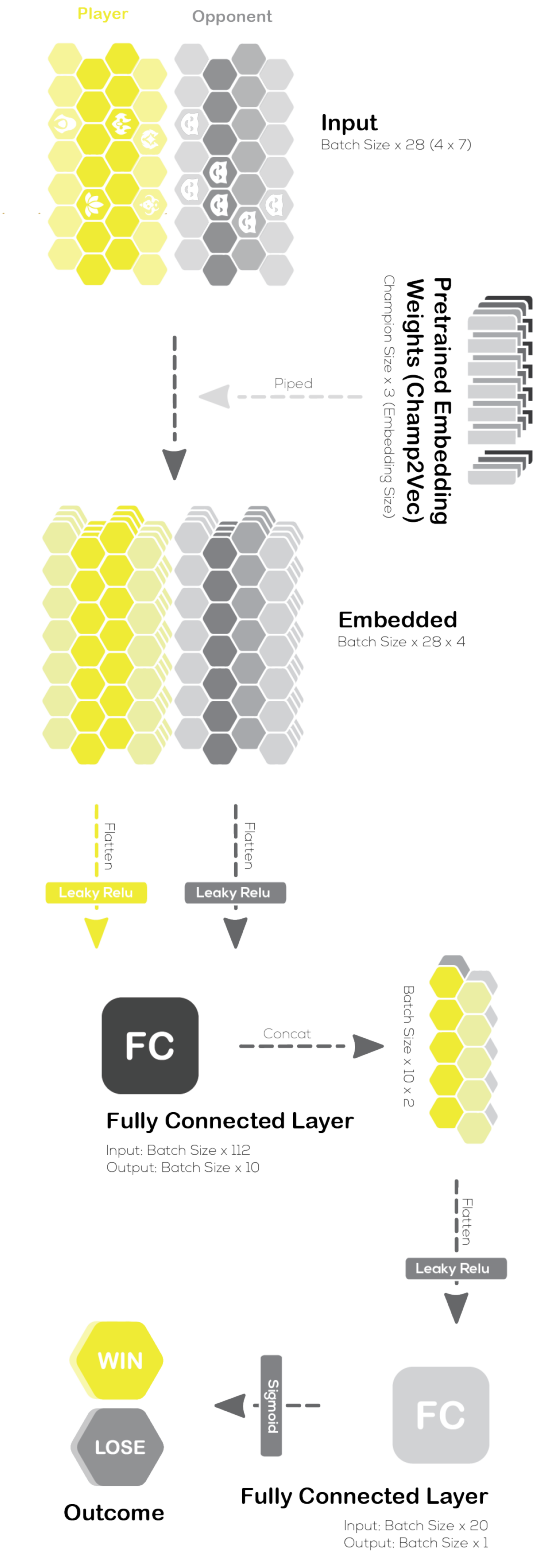

Figure 8: The Win/Lose Predictor Network

- For model training, batch size of 1, learning rate of 0.01, and momentum of 0.9 are used with the Stochastic Gradient Descent optimizer.

- Since this is a binary-output problem; a binary Cross-Entropy Loss is used.
- For model evaluation, 20% of the total data-set is randomly sampled to perform accuracy testing.

Results

In this section, we will be focusing on the result from the model described in the methodology sections above.

Champions Classification Net

ImageSegmentation	Accuracy
Champions	0.4684
Background	0.9553
Overall	0.7840

Table 1: Breakdown of accuracy from the confusion matrix

In order to produce these results, 20% of the data are sampled randomly as the test set and the remaining as the training set. Then, using the confusion matrix, the accuracy of the champions and backgrounds are calculated separately.

Win/Loss Prediction Net

Type	Model	Accuracy
Neural Networks	DNN with Init [1]	0.7844
	DNN without Init	0.7634
Conventional Method	Logistic Regression	0.7774
	SVM	0.7855
	K-Nearest Neighbors	0.7576
	Decision Tree	0.6736
	Random Forest	0.7855

Table 2: Accuracy measure of multiple models

The models mentioned in 2 are compared by accuracy. Here, accuracy denotes the number of correct predictions over the size of the data set. Each model used 20% of the data as the test set and the remaining 80% as the training set. The details of the models are as follows:

- DNN with Init represents the original DNN with pre-trained embedding weights calculated using the Champion2Vector algorithm.
- DNN with Init represents the original DNN where the weights of the em bedding layer are learned through the training process with random initialization.
- Logistic Regression with L2-loss.
- SVM with a linear kernel.
- K-Nearest Neighbors with number of clusters being 10.
- Decision Tree with criterion Gini impurity each node split on. the "best" rather than random.
- Random Forest with max depth of 5.

[1] "Init" means using pre-trained champion embedding vector as initial value for embedding layer

Discussion

CNN Limitations

The custom CNN produced satisfactory results given the initial batch sizes and time constraints. There are a few things to consider:

- There are 84 champions and 1 background to predict. Thus, an accuracy of 0.4684 is a sufficient starting point.
- The background accuracy is extremely high. Some reasons for this include the sheer number of empty spaces vs champions (maximum number of champions on the board is usually 9).

Additionally, the data used in training and classification are noisy. For example, in the following figures 9 in left to right order, it shows an image with popups blocking the entire champions, a background image with multiple champions parts showing, a background image with incomplete champions parts showing, and champions image with champions away from the center. Therefore, the image classifier must have a high-level knowledge about champions and backgrounds in order to make accurate predictions.

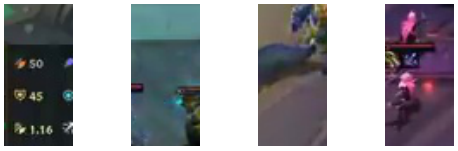

Figure 9: Sample Noisy Image

Some champions had significantly more appearances in the training dataset than others. Thus there is a clear bias on predicted new images. For example, the champion Warwick appeared 28 times, while the champion Sion only appeared eight times. In Figure 10, a sample of 5 images is taken for each champion. The top row shows that Warwick is predicted correctly more often, while the bottom row shows that Sion is incorrectly predicted for all 5 of the images.

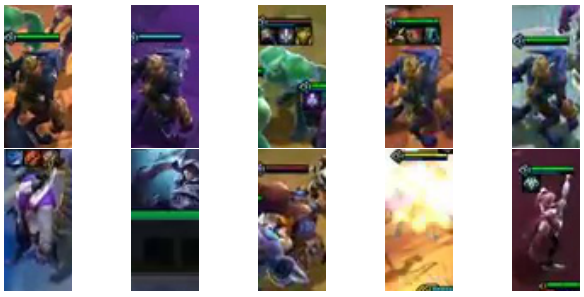

Figure 10: Predicted Labels on unseen examples

Model	Accuracy
DNN with Init	0.7844
DNN without Init	0.7634

Table 3: Model with/without Embedding Initialization

Embedding vs Random Initialization

The table above 3 shows that the pre-trained embedding vector does help improve the model test accuracy by around 2%. The Champion2Vector algorithm described in provided a good starting point for champion embeddings in the win/lose prediction model. With embedding dimensions of 2, the embedding vector is able to learn the inter-relationships between each champion. As shown in the image below 11, the champions with the same traits are closely clustered together. An embedding dimension of 4 is used to initialize the weights for the win/loss prediction net in order to improve the accuracy.

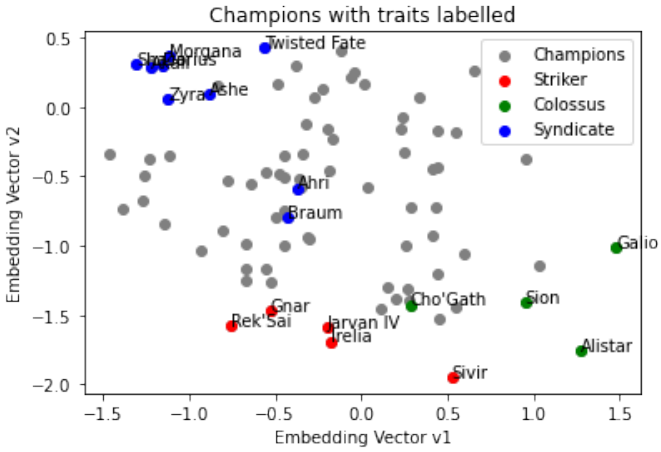

Figure 11: Champions with traits labelled

By initializing the champions embedding, the neural network is provided with prior knowledge of how champions are similar or different from each other. Therefore, it may accurately predict the result even with champions it has never seen before. However, the accuracy improvement is not obvious due to the limited context provided in the training embedding vector. Namely, The Champion2Vector algorithm only clusters champions by their "traits," but it is also important to look at other dimensions. For instance, the champion's special abilities, health points (HP), mana points (MP), and potential combination attacks.

DNN vs Conventional Models

As shown in table 2. Our neural network model is among the top-tier performing models, slightly under-performing against SVM and random forest. All of the models show significant performances in the testing set, which is above the threshold of 50%. It is clear that the gaming result is very predictable given the data generated from the video data pipeline.

However, the conventional method's accuracy is close to the neural networks, which is against our null hypothesis. We conclude the following reasons for such result:

- The input data dimension is low (only $4 \times seven \times 2$), considering most of the input tensors are paddings (as most cells are empty), the truly informative dimensions are usually less than 18 (each side can usually have at most nine champions). Therefore, it is possible that the result is quite linearly separable.
- The input data is biased since all of the training and testing model is based on a limited amount of YouTubers who tend to play similar strategies in each game. It is possible that the win rate for a specific strategy is around 75%.
- Noisy input due to under-performing champion classification network. In the champions classification network, the accuracy for champions recognition is 46%. As a result, the win/loss prediction net is likely to misinterpret the board in such ways that it learns the game in a not meaningful way.

Comparison to Previous Study

The main difference between our approach and the approach developed by Casian and Zannato (Casian and Zannato 2020) is the method of acquiring the data and the features chosen. Using the RiotGame's API for Teamfight Tactics, Casian and Zannato are able to access clean data denoting the final fight before a player won/lost. The features include champion name, item, etc ... There are a few limitations due to the fact that the API only provides data for the "last battle."

- Occurrences of low-cost units will be low since final boards will usually consist of expensive and high-stared units.
- Predictions on battles with few champions are unpredictable since the last fight will usually consist of 7+ champions (unless the player loses the game early on).
- The final battle will usually consist of the "best team compositions," which will lead to a lack of diversity in champion combinations.

The method of capturing keyframes removes these limitations. Although the results 2 are under that of using the Riot API, the underlying issue is the lack of time and resources. Even with a sufficient data-collection strategy, the win/loss predictions are quite similar. With enough time to refine the champion classifier, the win/loss prediction net is likely to improve greatly.

Additional Considerations

There is a randomness factor in Teamfight Tactics. Namely, the opponent is randomly chosen given a set of rules. The restriction is that for each stage, a player cannot meet the same opponent twice, given that there are enough players. Otherwise, it is possible to meet the same opponent. The model can be adapted such that it considers an optimal board that will maximize the probability of winning over all possible opponents rather than a single opponent. This is important

Model	Accuracy	Cross Validation Score
Logistic Regression	0.802	0.802
SVM rbf	0.866	0.872
K-Nearest Neighbour	0.772	0.791
Gaussian Naive Bayes	0.573	0.594
Decision Tree	0.883	0.897
Random Forest	0.914	0.924
DFNN	0.890	0891

Figure 12: Win/Loss Model Results (Casian and Zannato 2020)

since a specific positioning may maximize the probability of winning for one opponent but drastically reduce the probability of winning for another opponent.

Conclusion

Overall, the results align with our initial hypothesis that gameplay outcomes can be determined by using the champions' board position and name. In addition, the model accuracy is similar to the previous study (Casian and Zannato 2020). Despite the difference between the model accuracies being insignificant, we have shown that the current model has great potential in scalability and overall automation.

The biggest limitation to the models' performances is the lack of clean training data. The reasons include:

- Noisy input images due to random incidents (e.g., mouse hovering over champions, random information popups, non-champion units obstructing the board, player viewing another board rather than their own).
- The poor segmentation method of key-frames (e.g., parts of a champion in another cell appearing in the cropped image, misplaced bounding box if player zooms in/out of the default screen magnitude).
- Assumption that every YouTuber has the same editing style (e.g., some YouTubers will skip rounds completely).
- Only a small selection of YouTubers are used as input. (i.e., the strategies are limited to the selected YouTuber's playstyle).
- Small difference in interfaces in between each version of the game, which leads to bad backward compatibility of our current workflow (e.g., key-frames missed due to combat flag being of different font and size).
- Insufficient time and human effort to manually label training data.

The following proposals can improve the project in the future:

- Use the encoder-decoder structure to train directly on the unsegmented image as described in Deeplab model structure (Chen et al. 2017). Essentially the model is able to learn segmentation and champion classification at the same time. Therefore, this will greatly simplify our data input workflow and enable us to use significantly more gameplay data (especially the gameplays in the previous version).
- Use other champion-related contexts to improve the training of champions embedding vectors. For instance, perform an unsupervised deep learning algorithm such as the attention network invented by Vaswani (Vaswani et al. 2017) on Teamfight Tactic forums.
- Incorporate other gameplay data other than champions and positioning of the champions, such as augments, champion's level, items, and gold. Items empower individual units that equip the items and augments empower the entire board or the player (e.g., all units gain more health points, after a set amount of time, all units deal more damage). Adding these additional factors has a high potential of improving the accuracy of the model.
- Utilize information on the highest winning team composition to help determine which combination of items, champions, and levels will lead to a higher probability of winning.

References

[Bradski 2000] Bradski, G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[Casian and Zannato 2020] Casian, A., and Zannato, M. 2020. Predicting teamfight tactics results with machine learning techniques.

[Chen et al. 2017] Chen, L.-C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; and Yuille, A. L. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence* 40(4):834–848.

[contributor 2021] contributor, i. 2021. How do you become a chess grandmaster? the ultimate guide.

[David, Netanyahu, and Wolf 2016] David, O. E.; Netanyahu, N. S.; and Wolf, L. 2016. Deepchess: End-to-end deep neural network for automatic learning in chess. In *International Conference on Artificial Neural Networks*, 88–96. Springer.

[Foret et al. 2020] Foret, P.; Kleiner, A.; Mobahi, H.; and Neyshabur, B. 2020. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*.

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

[Mikolov et al. 2013] Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Pirogov 2022] Pirogov, S. 2022. Webdriver-manager.

[Shubham 2021] Shubham, K. 2021. Build a youtube downloader with python.

[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484–489.

[Simon 2022] Simon. 2022. Selenium.

[Vaswani et al. 2017] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems* 30.