Incorporating a Computer Vision System to a Web Server to Parse Real-time Data for Valorant Esports

Jan Jozef R. Laguer
Ateneo de Manila University
Quezon, Metro Manila
jan.laguer@student.ateneo.edu

Jenilyn A. Casano
Ateneo de Manila University
Quezon, Metro Manila
jagapito@ateneo.edu

ABSTRACT

There is currently no tool that allows broadcasting teams to visualize in-game events in Valorant. Unlike its competitor, Counter Strike, Valorant supports no such feature that allows a software to request information pertinent to Esports broadcasting. This paper made a system to get information from Valorant by making use of computer vision to parse frames and web servers to distribute it. Previous systems were explored to serve as a foundation and to see if any improvements can be made. A Convolutional Neural Network and Optical Character Recognition engine were used to perform Image Classification and text extraction, respectively. Although a lot of information can be extracted, there exists data that is not visually represented and cannot be extracted. Due to the nature of the game being a fast-paced tactical shooter, there is a need for the tool to be able to parse and deliver the data at a high speed. The system was tested on its execution speed and its accuracy by testing the individual modules and the system as a whole. After collecting the needed metrics, it is found that the tool is accurate and is able to return data at a near real-time speed.

KEYWORDS

Neural networks, web server, esports, broadcasting

1 INTRODUCTION

Esports tournaments are important to gaming culture. Kerttula has said "Online streams and even some television shows create a spectacular atmosphere around these tournaments and the best players are now celebrities" [15]. These tournaments help grow the games and communities they are a part of and entice new people to join in the event. Events hosted by the game's creators especially have high showing. The PGL Major 2021 and the International 2021, a Counter Strike: Global Offensive (CS:GO) tournament and a DOTA 2 tournament, respectively, both hosted by Valve had an average of 590 thousand and 850 thousand viewers, respectively [6, 7]. More recently, the Valorant Champions Tour (VCT), a Valorant tournament hosted by Riot Games, had an average of 460 thousand viewers [8].

While Valorant has grown from the VCT, the community has also helped grow the player base through their own tournaments. This includes different content creators and communities like MaguCup and ValorantPH's public games. AcadArena, an organization for Campus Gaming and Esports Education in Southeast Asia [1], hosted their tournament, the Alliance Games, which included Valorant where the finals boasted 26 thousand views not including the live audience [2].

With how young the game is, there are many features present in other esports titles that are not present yet in the current game itself. One such feature which this study focused on is a publicly available method to get data on a currently active game. CSGO has this feature called Game State Integration (GSI) which can expose "all game state, and send an update notification as soon as the client game state changes, to any local or remote HTTP POST endpoint using JSON as the game state structure" [27]. Currently, Valorant has no feature like this found on the official developer dashboard [9]. This feature is what powers other games' broadcasting tools use for their visualizations.

The goal of this study is to create a tool in order to address not being able to access the needed data as well as elevate the community's broadcasts and production similar to other tools used for other games to be similar to VCT. To fulfill this goal, the tool uses computer vision, machine learning, and web development. Another goal is to evaluate it to ensure that it is both accurate and has fast execution speed because this would be used in an environment where it is important to have both accurate and timely information.

To evaluate the tool, it must meet certain standards. One standard, accuracy, comes from ISO 5725. This describes accuracy as the combination of both precision and "trueness" [12]. This means that to be accurate, the evaluation must be close to the actual value and the spread of points must be close together. Another standard is that events that happened in the game must reflect on the tool with little delay. The Advanced Television Systems Committee (ATSC) has established synchronization limits where the delay is 45±15ms [16]. Additionally, the concept of real-time is often associated with the lack of delay and synchronization between two things. However, the term real-time is not strictly defined. One paper has described their tool, VisWiz as near real-time with the delay being less than 20 seconds [3] while another described real-time by stating how many frames can be processed within a second or frames per second (FPS) which in this case was between 24 and 60 FPS [14]. This is roughly a delay between 16ms to 41ms.

2 REVIEW OF RELATED LITERATURE

Each game has its own way of giving information which include the player's state in the game, current standings, and other elements specific to a certain game to spectators. For an in-game example, DOTA 2 allows spectators of a game to access different statistics that players in game have no access to. Other games, instead of having this type of tool built into the game, allow getting information about a current match outside the game. An example of this would be CS:GO's GSI Integration as this opens up a web server on a computer currently running CS:GO and exposes "all game state, and send an update notification as soon as the client game state changes, to any local or remote HTTP POST endpoint using JSON as the game state structure" [27].

DOTA 2 provides different statistics that can be shown like net worth, experience rate, and amount of last hits and denies. By default, each option shows the raw numbers for each statistic but can be transformed into a graph in order to make it easier to read and find patterns. Most of these statistics and visualizations aren't available to the players playing. At most, the players only have the statistics for their own team and selves' kills, deaths, assists, and current gold.

Production tools will be defined as tools that allow a broadcaster to visualize something to show in broadcast automatically. These tools make use of data in order to show a broadcast's audience pertinent data. For the case of esports, these tools visually display a game's current state at certain times or when the production team deems it necessary. A popular tool that is used in the CS:GO tournament scene is Lexogrine [17]. This tool takes in data, parses it, and displays it depending on the event received from the GSI server. The GSI server makes the data available for both teams; however, a player will only have their team's data available at any time. The data for the other team is only updated during the start of each round. The figure also shows some data that is not received from the GSI server. These data include the team names and team logos, where Lexogrine handles the state of each one.

Valorant has currently no way of getting this data for a normal user, as there's no in-game solution like DOTA 2 or an outside solution like in CS:GO. This means that production tools for Valorant are scarcely available. The reasons include a lack of an official implementation, the age of Valorant, and the difficulty of real-time data scraping itself. In order to build such a tool, there is a need to take a look at other Production Tools, especially those relating to Valorant. The lack of a production tool to assist those wanting to broadcast their community's games results in a big difference in audience experience.

2.1 Similar Systems

The closest application to this study is the Lexogrine HUD Manager. The backend for Lexogrine was written in JavaScript and used node.js and express.js for the server [17]. Lexogrine also has a local database using NeDB.js to store any local data [17]. This meant that each broadcaster had to run their own instance of the Lexogrine HUD Manager. The communication between the server and the HUD is mainly via a notification that a change in data has occurred via a websocket connection. This is followed by sending a GET request to the server for the new data.

Another 2 projects that aim to create production tools despite Valorant's lack of data are deepsidh9's Live-Valorant-Overlay [5] and tugamars' AcheronObs [26]. Each project used a Python server that used computer vision to extract data from the screen. Where each project differed was in the frontend, as they use either a JavaScript and HTML frontend or a Java frontend. It is of note that these 2 projects are no longer being maintained, as both projects have not been updated in at least 2 years as of March 2023.

A common point for these two project's backend is the use of a web server in order to deliver the data to the frontend. This is very similar to CS:GO's GSI feature though it was not written whether the developers used that as an inspiration for this backend. Both projects use Python to create the backend though the framework for the web server for each project differs. deepsidh9's implementation uses the Flask framework, while tugamars uses a newer framework, FastAPI.

These projects have different methods in order to deliver the data to the frontend. deepsidh9 implemented a web-socket using Socket.io as the main method of sending data. This implementation lets the backend control when the data gets sent. tugamars has a different approach wherein the frontend sends a request which triggers the image processing. This implementation lets the frontend control when the data gets sent.

The backend for both these projects have the same role, getting data from the current Valorant window, though each project has a different way of doing it. Tugamars' implementation uses another program to capture Valorant in order to start the scraping process. The program is called OBS, which is an open source software for capturing screens and or windows and broadcasting to the internet [23]. This method requires OBS to be less than version 28 as the plugin used by tugamars is not compatible with the later versions. deepsidh9 has a different approach as it uses the Windows API library in Python in order to access the Valorant window itself. After some experimentation, this method does not seem to work for the current release of Valorant, as it is possible the way Valorant renders its screen has changed. After capturing each frame, the backend then parses it using computer vision via the OpenCV Python library.

2.2 Computer Vision Implementation in the Similar Systems

Each project goes about this process differently. deepsidh9's implementation was able to collect information such as a player's life status, shield type, primary weapon, and ultimate status, and current game score and spike status [5]. tugamars' implementation has a shorter list with only being able to get spike status, player's hp value, and whether a player has their ultimate up or not [26]. The reason for this discrepancy is that the former was able to scrape data from the scoreboard, while the latter had not implemented that feature. Additionally, the latter's implementation required the observer to go to an area of the map where the screen has only one color, black.

The advantage of this was the ability to get a numerical value for the health of a player, as there is currently only one element found on screen relating to the health value of a player. This element is a bar that decreases and reveals the background as the value becomes smaller. Due to the element's property of having a transparent background, collecting hp data can become inaccurate as the background could lead to the element looking like it is full when in actuality it is not. The disadvantage for this method is the computer scraping data cannot be used as an observer for the game, as the only thing the computer will see using this method is just a black background along with the in-game HUD. This would require another computer to act as an observer in the game that will receive data from the backend server. As deepsidh9's implementation does not use this method, only one computer would be needed in order to fully utilize the whole production tool, at the cost of not being able to get numerical values for HP.

Another area where these projects differ is how each project handles optical character recognition. deepsidh9 uses the python library EasyOCR while tugamars uses Tesseract [5], [26]. According to Liao's experimentation, EasyOCR is more accurate when compared to Tesseract when it comes to recognizing numbers, while the latter is more accurate when it comes to recognizing alphabet characters [18]. They also mention that Tesseract's processing time is faster compared to EasyOCR when it is done on a CPU [18].

Template matching is a technique absent from tugamars implementation and present in deepsidh9's. They use this technique to analyze and figure out what images represent. Specifically, they use this in order to figure out each player's character, armor type, and primary weapon. This can be further optimized as the current implementation loops through a set of templates for each character, weapon, and armor types. In the worst case, it's possible that the correct template would be at the end of the loop, which makes the system perform suboptimally. A notable use of template matching was detecting if the scoreboard was visible on screen, as there's only one template that needs to be checked.

3 METHODOLOGY

3.1 Programming Language

In fulfillment of this study's objectives, the researcher has outlined the use of Javascript and Python as programming languages.

The choice of JavaScript is due to its prevalent use in web applications and is, as Crockford wrote, the "Language of the Web" [4]. The reason for the need of a browser-based application is that OBS has a feature that allows for them to be integrated into the program. This integration allows the use of transparent backgrounds and elements for the frontend. While using chroma keys can achieve the same transparent backgrounds and elements, this method requires that a certain color is removed entirely from the UI. This would prove not possible as there are numerous agents and may result in an agent having transparent elements which would be distracting from the game that this HUD is supposed to be enhancing. In using this programming language, the application is able to communicate with a server and is able to change different elements based on what the application receives. This acts as the HUD that displays data that it receives from the server.

The choice of Python is due to the programming language's different modules that includes different web development frameworks [24] and computer vision and image processing tools from OpenCV [20]. Web development frameworks were mentioned due to how compatible they are with web applications. The compatibility comes from the server included in these web development frameworks, which enables communication between the said server and the web application as the client [19]. Computer vision is a requirement as this is the main method in order to get data from Valorant's game screen due to the game's unavailability of a method to get current game state [9]. All in all, this allows getting data from Valorant's game screen and sending it to the HUD.

Python was also chosen in order to create the image classification models. As mentioned previously, there are many libraries and modules in the Python ecosystem, and the library that was used to create an image classification model would be TensorFlow. According to Johnson, TensorFlow is an open-source end-to-end platform for creating Machine Learning applications and is focused on training and inference of deep neural networks [13].

3.2 Image Classification Model Creation

There are currently two visual elements that showcase an image in order to represent something. With this in mind, the researcher would need to create two image classification models, one to identify a player's character and another to identify a player's primary weapon. Using TensorFlow, the models would be created using a Convolutional Neural Networks (CNN). O'Shea and Nash wrote that CNNs are mainly used for their pattern recognition in images making them suitable to image focused tasks [21]. Another method would use Template Matching similar to what previous projects used but in Hmdaoy and Ahmed's study comparing the two, they have found that using a CNN was both faster and more accurate compared to template matching [11]. CNNs commonly use 4 layers, the input layer, the Convolutional Layer, the Pooling Layer, and the Fully-connected Layer.

3.2.1 *CNN Architecture.*

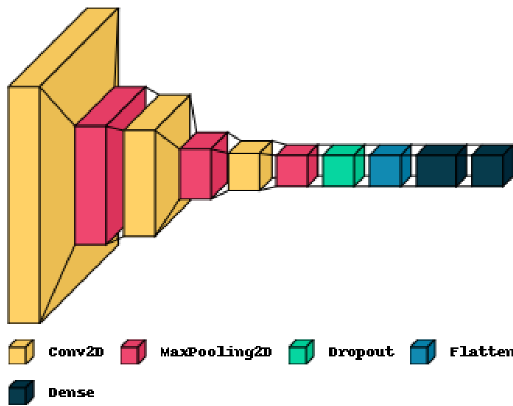
The Architecture of the CNN for creating the models would follow a common way to set it up. The most common way of setting up the CNN Architecture is to set a Convolutional Layer followed by a Pooling Layer multiple times, then following it up with Fully-connected Layers. The specific Architecture for this thesis' CNN would be as follows: an input layer, three pairs of Convolutional Layer with rectified linear (ReLu) activation and Pooling Layer, A Flatten Layer to transform the image to a 1 dimensional array, A Fully-connected Layer with ReLu activation, and finally a Fully-connected Layer with softmax activation. In order to prevent overfitting, a Dropout Layer with a rate of 20% was added after the last Pooling Layer. Refer to Figure 1 for the visualizations of each model.

The models are mostly similar with some differences. The convolution layers had a kernel size of 3x3 with "relu" activation. The agent model had 16, 32, 16 filters while the weapon model had 16, 32, 64. Both models have the same pooling layers with a pool size of 2x2. The second to last dense layer has 256 units for the agent model while the weapon model had 128. Refer to Figure 2

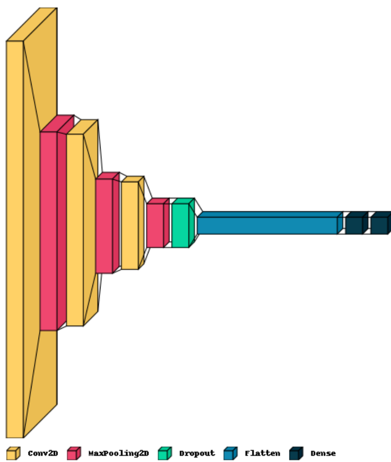
3.2.2 *Collection of Training Images.*

Before creating the CNN to create the model, the training images would need to be created first. The researcher first collected the display icons with transparent backgrounds for each Valorant agent. Figure 3b shows an example of a Valorant agent picture. The researcher then created a Python script to put each image in a directory named for the Valorant agent they represented. In each directory, the display icons were resized to a 40 pixel by 40 pixel image, mirrored, and had different background colors applied. The specific colors applied are as follows: #26493f, #5a221f, #494834, #53755d, #9aa36f, #90a366, white, black, blue, red, #307368, #5b2929, #285e54, #4a2323. After that, each image was copied and then resized to a 30 pixel by 30 pixel image. The resizing of pictures is done because there are visual elements that are these sizes and was made to ensure the model has no problem interpreting them regardless of size.

The collection of the training images for the weapon classifier had a different methodology. A custom game with the scoreboard in constant view was recorded in order to capture the different



(a) Agent Model



(b) Weapon Model

Figure 1: Visualization of Models using visualkeras [10]

weapon icons. The recording contained footage of the researcher traversing the map with the scoreboard in the center of the screen. For consistency, the same route was done for all weapons. The game was recorded in 1920x1080 resolution with 60 frames per second. Using a Python script, frames from the recording were extracted by getting every 6th frame and cropping the frame to only the cell in the scoreboard that shows the primary weapon of the player. Figure 3a shows an example screenshot. These images were saved in a directory with the name of the weapon it is representing as the name.

The dataset for the agent and weapon classifiers resulted in 5128 and 5648 images respectively. The dataset for the agent classifier resulted in 20 classes with an average of 256 images per class. The dataset for the weapon classifier resulted in 18 class with an average of 300 images per class.

3.3 OCR System

The Python module and library ecosystem allows for convenient installations for OCR. One of the previously used OCR Systems,

Name (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 38, 38, 16)	448
max_pooling2d (MaxPooling2D)	(None, 19, 19, 16)	0
conv2d_1 (Conv2D)	(None, 17, 17, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 6, 6, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 16)	0
dropout (Dropout)	(None, 3, 3, 16)	0
flatten (Flatten)	(None, 144)	0
dense (Dense)	(None, 256)	37120
dense_1 (Dense)	(None, 20)	5140

(a) Agent Model Summary

Name (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 118, 16)	448
max_pooling2d (MaxPooling2D)	(None, 15, 59, 16)	0
conv2d_1 (Conv2D)	(None, 13, 57, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 6, 28, 32)	0
conv2d_2 (Conv2D)	(None, 4, 26, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 2, 13, 64)	0
dropout (Dropout)	(None, 2, 13, 64)	0
flatten (Flatten)	(None, 1664)	0
dense (Dense)	(None, 128)	213120
dense_1 (Dense)	(None, 18)	2322

(b) Weapon Model Summary

Figure 2: Model Summary

(a) Example Screenshot

(b) Example Agent

Figure 3: Example of Image Collection

Tesseract, has a different way to install it. This installation involves downloading a separate binary to ensure the respective module

works. To simplify the process, another OCR system would be chosen on the basis that it is on par or better in terms of accuracy and speed and has a more simple installation process.

The OCR Engine that was used for development is PaddleOCR which can be installed via pip, the built-in way to install packages in Python environments. It self-documents as a "practical ultra-lightweight OCR system" with considerations with regard to the balance of accuracy and speed [22]. The process for this system includes first detecting texts and setting up the bounding boxes and then performing text recognition on each of those boxes by using Differentiable Binarization (DB) and Convolutional Recurrent Neural Network (CRNN) [22].

Once the output has been generated from PaddleOCR, post-processing is done to keep the output consistent for each column of the scoreboard that uses OCR. The output also has any whitespace stripped from the beginning and the end.

OCR would be used on every section of the scoreboard, except for the agent portrait column, weapon column, and the ping column on each row. For sections that only have numbers, any special characters or letters are removed. One special case is the ultimate status column of the scoreboard. Majority of the time, this column only features numbers and a slash. Once a player's ultimate is ready to use, the scoreboard would show "READY" instead of the numbers. The post-processing would make sure to turn the "READY" into a number to keep things consistent. The number would be dependent on each agent, as each agent has a different number of max ultimate points.

3.4 Development Frameworks

To facilitate a more efficient development, frameworks would be used for both the creation of the backend server and frontend application. While a framework was used in creating the backend server for the previous projects, there was none used for the frontend application. tugamars and deepsidh9 has used FastAPI and Flask respectively for their projects. Both of these are popular web frameworks used to create web servers.

The chosen backend framework for this thesis would be FastAPI. According to the benchmarks created by TechEmpower, FastAPI has a higher performance score compared to Flask in terms of single query and multiple queries [25]. This framework also automatically creates interactive API documentation on the /docs and /redoc URLs. Besides that, FastAPI also creates a schema using the OpenAPI standard for creating APIs. This allows for easy testing of individual modules and testing of the system as a whole.

The chosen frontend framework for this thesis would be NextJS. NextJS is a web framework using the React UI library. By using this framework, it is possible to create multiple pages as well as handle different requests to the backend server in order to mutate any data that is on the server as well as any internal data on the client. This framework also allows faster development as the routes are automatically generated from the file structure during development.

3.5 Testing

Once the product is complete, testing was done to determine the tool's performance compared to broadcasting real-time standards. These broadcasting standards would follow what the ATSC has established, where the synchronization limit delay should be in the range of 45±15ms [16]. The other standards mentioned previously would also be considered. Accuracy would determine the correctness of the information being displayed. Execution time would refer to the time it takes for the tool to parse data from the game and display it.

To illustrate, examining Figure 3a would result in the following data, which would be in a JSON format. The spike status would be false. The team on the left would be defending and the team on the right would be attacking with 1 and 0 players respectively. The player has Neon as an agent with 0 kills, deaths, and assists, has their ultimate ready to use, has 99,999 credits, and has the Sheriff as their highest value weapon.

These tests were done in order to evaluate the different parts of the tool in terms of accuracy and execution time. The accuracy tests were done on the models, both OCR and CNN. The tool parses two areas, the scoreboard and top bar, at the same time. The test for execution time was done on these two parts of the tool as well as the system as a whole.

4 RESULTS AND DISCUSSION

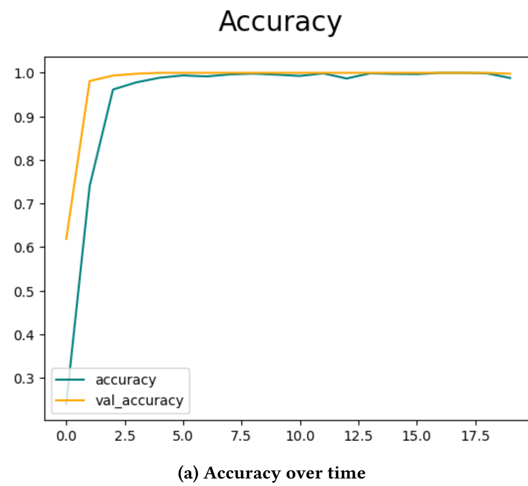
4.1 Accuracy

4.1.1 Agent Classification Model. The training data for this model was first loaded into an image dataset. This was done using the utility function "image_dataset_from_directory" method from the keras module which also separates the images into a number of batches. In that dataset, the images were then normalized to keep the values between 0 and 1. The dataset was then split into training, validation, and test datasets by splitting them into a ratio of 7:2:1. This divided the 161 batches into 113 batches, 32 batches, and 16 batches for the training, validation, test datasets. This ratio allows for the majority of the dataset to be used for training and validation. This also allows means that some images are not used for training or validation, which makes the testing more reliable as it is not recognizing images that it already has seen before.

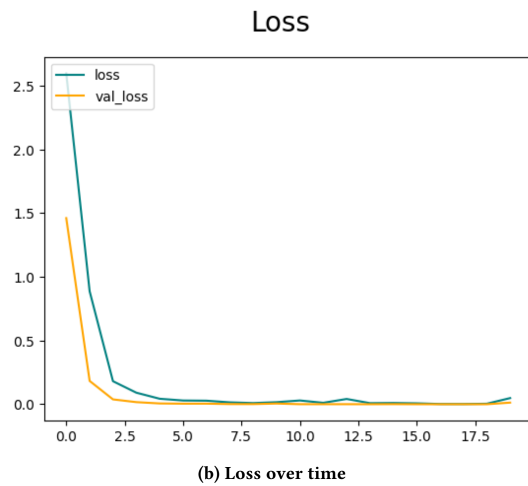
The model was then created using the architecture stated above. The Convolutional Layers for this model had 16, 32, and 16 filters respectively. Each of these layers had a kernel size of 3x3 with a stride of 1 and had ReLu activation applied. The Pooling Layers had a size of 2x2 which essentially results in the image being cut in half after each layer. The Fully-connected Layers had 256 and 20 units respectively, and when combined with the softmax activation allows the model to give each class a confidence level. The model was compiled with the adam optimizer and the categorical crossentropy loss function. Finally, the model was trained for 20 epochs.

Figure 4 shows the loss and accuracy of each epoch of training. The figure also shows that the training and validation lines converge, which means that the resulting model does not suffer from over fitting. Over fitting, in the context of machine learning models, is when the model is good at identifying what it was trained on but not on things that it has not seen.

The model was then evaluated using the test dataset. This was done with the metrics for precision, recall, and binary accuracy. Precision measures how often the model is correct every time it predicts positive for a class. Recall measures how often the model

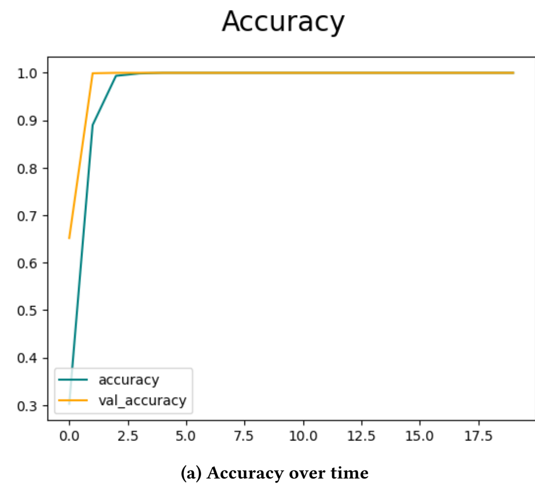


(a) Accuracy over time

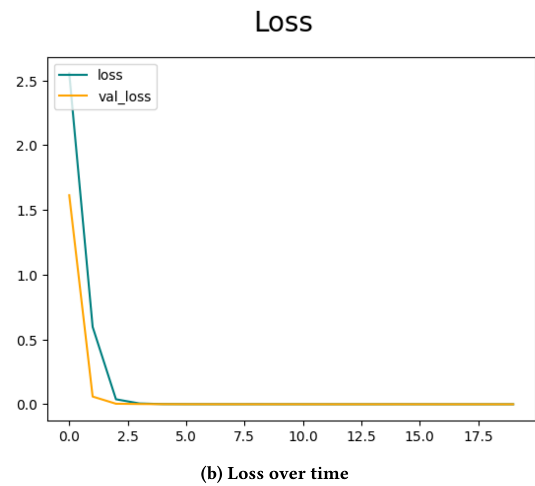


(b) Loss over time

Figure 4: Agent Classification Model Training and Validation Metrics



(a) Accuracy over time



(b) Loss over time

Figure 5: Weapon Classification Model Training and Validation Metrics

predicts correct every time a class appears. Binary accuracy is a measure on how often the model gets a prediction correct.

Below are the results for the agent classification model. All metrics are seen to be above 99% which means that the model is considered to be highly accurate.

- Precision=0.99553573
- Recall=0.99553573
- Binary Accuracy=0.99955356

4.1.2 Weapon Classification Model. Similar to the previous model, the training images were loaded into an image dataset. The images were then normalized to be between 0 and 1. The dataset was also split into training, validation, and test datasets with the same ratio in the previous model. This divided the 177 batches into 125 batches for training, 35 batches for validation, and 17 batches for testing.

The model was created using the architecture stated previously. The Convolution Layers for this model had 16, 32, and 64 filters respectively. The kernel size was 3x3 with a stride of 1 and had ReLu activation applied. Pooling Layers are the same. This differed as the Fully-connected Layers had 128 and 18 units respectively. The model was compiled with the adam optimizer and the categorical crossentropy loss function. Finally, the model was trained for 20 epochs.

Figure 5 are the loss and accuracy over each epoch of training. The figure shows, similar to figure 4, the training and validation lines converging. This means that the model also does not suffer from over fitting.

The model was then evaluated using the test dataset. Like previously, the evaluation was done using the precision, recall, and binary accuracy metrics. Below are the results for the weapon classification model. Similar to the previous model, these metrics indicate greater than 99% which means this model can be considered to be highly accurate.

- Precision=1.0

```
1  train1 = data.take(128)
2  val1 = data.skip(128).take(32)
3
4  train2 = data.take(96).skip(32).take(32)
5  val2 = data.skip(96).take(32)
6
7  train3 = data.take(64).skip(32).take(64)
8  val3 = data.skip(64).take(32)
9
10 train4 = data.take(32).skip(32).take(96)
11 val4 = data.skip(32).take(32)
12
13 train5 = data.skip(32).take(128)
14 val5 = data.take(32)
15
16 test = data.skip(train_size + val_size).take(test_size)
```

Figure 6: Python Code for Batch Distribution for K-fold Cross Validation

K-fold Iteration	Precision	Recall	Binary Accuracy
1	1.0000	1.0000	1.0000
2	1.0000	1.0000	1.0000
3	0.9871	0.9871	0.9902
4	0.9934	0.9934	0.9976
5	0.9959	0.9959	0.9995

Table 1: Accuracy Results after K-fold Cross Validation

- Recall=1.0
- Binary Accuracy=1.0

These results appeared unconventional which resulted in conducting a k-fold cross-validation to scrutinize the robustness and reliability of the findings. Having a 100% result on these metrics is unrealistic and may suggest the model is unable to perform well in a real-world scenario. This may also suggest that the model had overfitted to the training set. This may not be the case as the the graphs in Figure 5 show the training and validation lines don't deviate greatly which indicate the model has not overfitted. The same dataset that was used previously was used to do the cross validation. The dataset had a total of 177 batches with 32 images in each batch. The split of data was the same as before, except the training and validation datasets were used for the cross-validation. Using 5 as k, the model was trained 5 times using the k-fold cross-validation method and the resulting model was tested against the test dataset. The reason for k equal to 5 is it results to each fold equaling 20% of the data which ensures each fold is large enough to capture meaningful patterns in the data. Having k equal to 5 is also a common choice when doing this methodology. The fold equaled 32 batches which resulted in the training, validation, and test datasets to have 128, 32, and 17 batches respectively. Figure 6 shows the Python code used to split the batches into the training, validation, and test datasets. The data variable represents the dataset as a whole. The variables train_size, val_size, and test_size were calculated via the ratio presented previously.

Table 1 shows the results of testing the model after each k-fold iteration. The results imply that the initial results may have occurred by chance. It can still be said that this model is highly accurate as, on average, the model achieved an accuracy greater than 98% on each iteration.

Section	Average (s)	Minimum (s)	Maximum (s)
Scoreboard	1.291314	1.252528	24.384231
Top Bar	0.000566	0.000909	0.001002

Table 2: Execution Time Results

The initial results may have been caused by what the model is being used for. Valorant weapon icons don't change or have any variability which is the case for icons in general. The model may have learned this and achieved a high accuracy because of this. As the weapons icons in Valorant don't change or are obscured in any way, the fact the model has learned this way should not be a cause for concern and may have a positive influence when the model is predicting on real life scenarios.

Additional tests may be able to further verify the model. Performing K-fold Cross Validation with K equal to different sizes may show patterns that could validate or invalidate the model further. Augmentations to the test dataset could be done to test the robustness of the model.

4.1.3 OCR Accuracy. Getting the accuracy of the OCR was done by using it on 100 different images, specifically cells from the scoreboard, to tailor the results towards the use case of this study. The test itself did not use PaddleOCR by itself. The test would use PaddleOCR along with the post-processing done to the initial output. Each cell was assigned an expected value which was the value of the cell and was compared to what the OCR outputted.

Results show that the accuracy of the OCR was 99%. This means that the result of the text parsing process the tool uses deviates very little actual values found on the scoreboard. The inaccuracy came from a cell that had a "4" which resulted in the initial output treating it as a "b". For future works, it would be possible to gain better accuracy by adjusting the post-processing to recognize this edge case.

4.2 Execution Time

This metric represents how fast the software can update the data to the latest point. There are two sections of the screen that house all the data that can be collected, the scoreboard and the area at the top of the screen which will be called the top bar for this study. Each section was timed separately, since these sections would be executed independently of each other. This means that updates to the scoreboard happen without needing to wait for the top bar to finish its process. To measure the execution time, both processes are run 100 times with the first time excluded to get the average execution time without the cold start. Table 2 shows the average execution time as well as the minimum and maximum execution time for each section.

The difference between execution times may be related to how different the two sections are. The scoreboard is very text heavy and unlike the scoreboard, the Top Bar does not use any optical character recognition as there are no text-based elements in that section. The Top Bar is also static which means it only needs to use

the agent image recognition once while the scoreboard needs to keep using it as positions change very often throughout the game.

Comparing the results to previous real-time standards, the average execution time of one part is outside some standards. The scoreboard average execution time is outside the range defined by ATSC of 45±15ms [16] or the delay established by Jung et al. [14]. This, however, is within range of the paper discussing VisWiz [3].

5 CONCLUSION

The results show that the system as a whole can be considered to be highly accurate and updates at near real-time speed. The accuracy of the image classification models created and the OCR engine used were above 95% and can be considered to be accurate. The time to execute for the system as a whole has updates under the limits set by ATSC. Individually, the scoreboard module was not able to meet the execution time standards.

This computer vision based system was implemented by using and improving previous projects. The base of this system was a Python web-server which received and sent data that held data from the game. To collect the real-time information from the game, computer vision was used. A CNN was used to recognize different characters and weapons from the game and OCR was used to collect textual data. The data came from the 2 sections, Top Bar and the scoreboard, which updates the Python web-server independently of each other. The created visuals come from the data sent by the web-server.

There are a few points of improvement to consider. It was found during the tests that while the system as a whole updates within the standards set, the module handling the scoreboard fell behind. This can be improved upon by updating the main store as soon as a row in the scoreboard has been parsed. The way it was implemented in this study was to batch process the entire scoreboard via pipelining and then update the store once completed. The update function could be done per row so that updates from the scoreboard happen faster. It may also be possible to have each row be executed independently of the others to update more often and the time between updates would fall under the previously stated standards.

REFERENCES

[1] AcadArena. [n. d.]. About Us. https://www.acadarena.com/about-us. Accessed: 2022-9-28.
[2] AcadArena. 2022. DLSU vs ADMU - Alliance Games 2022 S1 Finals CONQuest.
[3] Jeffrey P Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and Tom Yeh. 2010. VizWiz: Nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, New York, NY, USA.
[4] Douglas Crockford. 2008. *JavaScript: The Good Parts*. O'Reilly Media, Sebastopol, CA.
[5] Deep. 2021. Live-Valorant-Overlay: An Overlay proof-of-concept Application for Valorant.
[6] escharts. [n. d.]. The International 10. https://escharts.com/tournaments/dota2/international-10. Accessed: 2022-9-9.
[7] escharts. [n. d.]. PGL Major Stockholm 2021. https://escharts.com/tournaments/csgo/pgl-major-stockholm-2021. Accessed: 2022-9-9.
[8] escharts. [n. d.]. Valorant Champions 2021. https://escharts.com/tournaments/valorant/valorant-champions-2021. Accessed: 2022-9-9.
[9] Riot Games. [n. d.]. Riot developer portal. https://developer.riotgames.com/apis. Accessed: 2022-6-26.
[10] Paul Gavrikov. 2020. visualkeras. https://github.com/paulgavrikov/visualkeras.
[11] Shayma A Hmdaoy and Hanaa M Ahmed. 2022. A comparison between the use of CNN and matching templates in recognizing the Iraqi license plate number. *Al-Nahrain Journal of Science* 25, 3 (2022), 43–50.
[12] ISO 5725-1:1994(en). 1994. *Accuracy (trueness and precision) of measurement methods and results — Part 1: General principles and definitions.* Technical Report.
[13] Daniel Johnson. 2020. What is TensorFlow? How it Works? Introduction & Architecture. https://www.guru99.com/what-is-tensorflow.html. Accessed: 2022-11-30.
[14] Ilchae Jung, Jeany Son, Mooyeol Baek, and Bohyung Han. 2018. Real-Time MDNet. In *Computer Vision – ECCV 2018*. Springer International Publishing, Cham, 89–104.
[15] Tero Kerttula. 2020. Early Television Video Game Tournaments as Sports Spectacles. In *Proceedings of the 2019 Esports Research Conference (ESC)*, Jason G Reitman, Craig G Anderson, Mark Deppe, and Constance Steinkuehler (Eds.). ETC Press, Carnegie Mellon University.
[16] Sara Kudrle, Michel Proulx, Pascal Carrieres, and Marco Lopez. 2011. Fingerprinting for solving A/V synchronization issues within broadcast environments. *SMPTE Motion Imaging J.* 120, 5 (2011), 36–46.
[17] lexogrine. 2020. Lexogrine HUD Manager.
[18] Chejui Liao. 2020. OCR engine comparison — tesseract vs. EasyOCR. https://medium.com/swlh/ocr-engine-comparison-tesseract-vs-easyocr-729be893d3ae. Accessed: 2022-10-12.
[19] Haroon Shakirat Oluwatosin. 2014. Client-Server Model. *IOSR journal of computer engineering* 16, 1 (Feb. 2014), 67–71.
[20] OpenCV. 2016. Opencv-python.
[21] Keiron O'Shea and Ryan Nash. 2015. An Introduction to Convolutional Neural Networks. (2015).
[22] PaddlePadle. 2022. PaddleOCR.
[23] OBS Project. [n. d.]. Open Broadcaster Software. https://obsproject.com/. Accessed: 2022-6-26.
[24] Sheetal Taneja and Pratibha Gupta. 2014. Python as a Tool for Web Server Application Development. *JIMS 8i-International Journal of Information, Communication and Computing Technology* II, I (2014), 77–83.
[25] TechEmpower. 2022. Web Framework Benchmarks Round 21. https://www.techempower.com/benchmarks/. Accessed: 2022-12-1.
[26] tugamars. 2020. AcheronObs. https://github.com/tugamars/AcheronObs. Accessed: 2022-6-10.
[27] Valve. 2020. Counter-strike: Global offensive game state integration. https://developer.valvesoftware.com/wiki/Counter-Strike:_Global_Offensive_Game_State_Integration. Accessed: 2022-6-26.