AimBot using Object Detection

Final Project Report

Walid HADRI

Abstract

Artificial intelligence came to mimic the way human beings process problems in different areas, the applications have been increasing with the evolution of the discovered algorithms and techniques. One of the areas, where emulating the human intelligence using a computational source has been interesting is Video Games. The interaction between AI and Video Games, made the benefits for both sides; board games for instance (chess, Go and Atari) have always been linked since the emergence of the discipline as they provide a fertile ground to evaluate the different AI methods and algorithms developed. In the other hand, research in AI has mainly led to the improvement of video games, by creating Non-Player-Character and understanding more the user behaviour and experience.

The goal of this project is to focus on category of Video Games (FPS Games) and automate the main task of the players. The approach used is Object Detection based on the Yolo Architecture, thus an annotated dataset is needed to train different Yolo Architectures for different time inference and performances to compare the results after. I started from Object Detection on Images, to detection on videos and then to detection in real time. The project takes an original approach motivated by my interest in both the game and Computer Vision.

1. Introduction and Motivation

First-person shooter (FPS) is a sub-genre of shooter video games centered on gun and other weapon-based combat in a first-person perspective, with the player experiencing the action through the eyes of the protagonist and controlling the player character in a three-dimensional space. The genre shares common traits with other shooter games, and in turn falls under the action game genre. The gamer is expected to propel his avatar through the game by moving it forward, backward, sideways and so on using the game controller for PC player it would be the keyboard and the mouse. Forward movements of the controller result in the avatar moving forward through the scenery, usually with a slight left-right rocking motion to properly simulate the human gait. In order to increase the level of realism, many games include the sounds of breathing and footsteps in addition to the regular sound effects.

Figure 1. Modern FPS Game

Figure 2. Old FPS Game

The most popular and trending FPS Games are online multiplayer counter strike games with two teams competing to win. When it comes to counter strike video games, aiming at the enemies is the crux skill needed to be good at the game. These games are mostly played on computers, the keyboard is used to move and the mouse is used to aim. I am going to be developing more about aiming in the next section. But for now, aiming should be seen as placing the crosshair of the weapon on the target.

Making the process of aiming at the enemy automated has been always a way to cheat in such games, but the automation is mostly done by interacting with game and ex-

tracting the exact information about the location of the enemies on the screen, the same information used to display them on the screen in first place. This method needs interacting with sockets comping up from the server and game developers make sure to make this interaction not possible, by restricting them and also defining banning rules regarding the use of any third party to communicate with the game so that cheating is not allowed.

The goal motivation behind the project is to develop ways based on Computer Vision to be able to detect enemies on the screen based on the same information a human player is receiving on the screen without any additional information. The detection should be enhanced so it could be performed in real time too, in different background with different frames per second rates. Creating an AI bot this way would be different from the classic cheaing Bots, as what they have as inputs are the same inputs a human being receives on the screen when playing.

One of the most played counter strike video games nowadays is Valorant made by Riot Games. In this project, I will be considering this game that I know very well to make an AimBot based on Computer Vision.

2. Problem Definition

As introduced in the previous section, aiming is the main task that players perform and should be good at when playing FPS counter strike games, Valorant in particular.

2.1. The Crosshair

The crosshair is indicated as the center of the screen with some styled shape. As you can see in the figure **Indicate the figure**, the crosshair is point out with the cyan square. The crosshair is used for aiming at the targets.

Figure 3. Crosshair is the cyan square (Valorant Gameplay)

2.2. Aiming and targets

The process of aiming is moving the crosshair on the targets, in Valorant the main targets are the enemies or the bots if you are on the training session. Pro-players takes hours each day training their aim, by making the process of moving the crosshair on the targets more **accurate**, more **faster** and more natural by developing **muscle memory**.

Figure 4. Aiming at the enemies

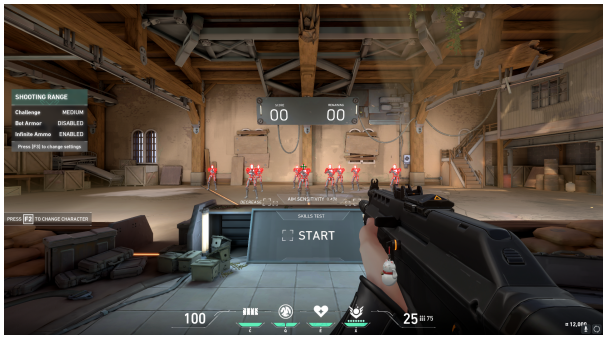

Figure 5. Aiming at the bots (used for practice and training)

However, one thing to keep in mind that aiming at different part of the body deals different damage for the same weapon. For Valorant, there are three parts of the target you can aim at:

- **The head** with the most damage that can be done
- **The body** with the second most damaged part
- **The legs** with the less damage that can be done

So, I will be interested in detecting the whole target in first and then detect the head afterwards. Then detect both of them at the same time.

So the problems the project is trying to address are:

- Target Detection: with constraints on the **accuracy** of the detection, on the **time** used to make the inference, the possibility to make the detection in real time and compete with the human gameplay.
- Detect part of the targets: as the damage dealt differ depending on when the crosshair is placed, the goal

here to be able to detect the head with two approaches using the object detection by detecting both the whole target and the head at the same time, and the second one is by detecting first the target and then use the results of this detection to capture the positions of the head.

- Make the detection on a reduced part of the screen using the help of a human player to guide the crosshair, and the goal of the AI Bot is to complete the task of aiming on a smaller part of the screen to make the detection more faster and more accurate, this way the human player is being assisted at the hard part of the gameplay.

The main goal of the project is given an input image, we should make an algorithm that will give us the coordinates of the targets (center of the target, then we are supposed to get two coordinates (x,y)) in first place then afterwards the coordinates of the head. This algorithm should be accurate to some degree that I will define later and fast so that the inference could compete with the human gameplay.

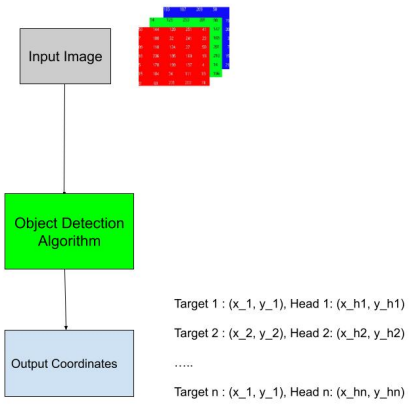

Figure 6. Formal definition of the problem

This problem falls under the umbrella of detecting the bounding boxes around the object that we are interested in. And we can formulate the problem as in the Figure[7].

3. Related Work

There are two ways to see the related work with this project, from the objective point of view of making an AI AimBot and from the technical one that is object detection on images:

3.1. From a goal point of view

As discussed in the introduction section, when it comes to making AI AimBot to automate the process of aiming

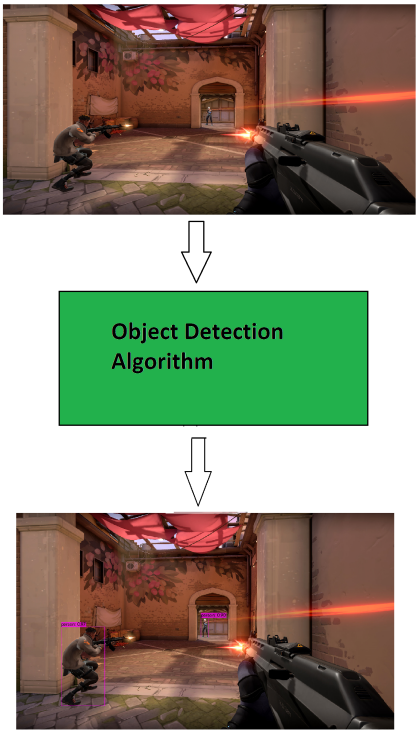

Figure 7. Illustrated formal definition by drawing the bounding boxes

in FPS Games, the most used AimBots are created based on the interaction between the server and a third party program.

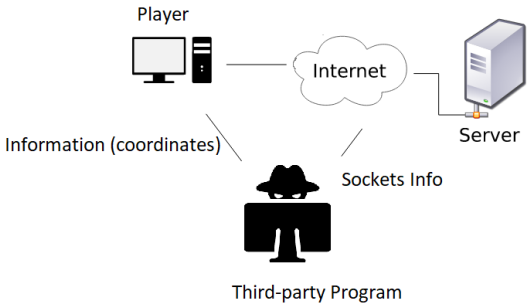

Figure 8. Third-Party program to get information the player does not have and the game is hiding

The approach considered in this project does not rely on any interaction with the server or with the files of the game, we are going to having the same inputs a human player is having that are in this case the images displayed on the screen.

3.2. From a method point of view

The methods related to Object Detection either they are based on Artificial Neural Networks or on Image Processing without deep learning have been used widely in different practical areas.

For the methods that do not use deep learning, we can refer to Template Matching for example as a way to detect objects on images given a pattern image. I considered this approach as part of my project in the VIC course given by Maria Vakalopoulou inspired by some previous work in the literature [2][3].

In this project, I will be considering algorithms based on deep learning. There are different architectures in the literature to make object detection that is the task of locating the presence of objects with a bounding box and types or classes of the located objects in an image. The two most popular families for this task are the **R-CNN Model Family** [5] and the **Yolo Family** [4] (the original paper for the Yolo architecture that I used during the project to understand how Yolo works), both of them based on anchors. Knowing that difference between the two is that RCNN offers a regional of interest region for doing convolution while YOLO does detection and classification at the same time. I will be considering Yolo as it is faster than R-CNN and that YOLO appears to be a cleaner way of doing object detection since it is fully end-to-end training without too much involvement.

4. Methodology

I focused on using the Yolo Family more precisely the tiny architecture as it is the more suitable to make the real time inference faster and be able to compete with the human gameplay. For training, I needed to create my own dataset.

4.1. Creating the dataset

Creating dataset to train Yolo means offering the bounding boxes information. In object detection, we usually use a bounding box to describe the spatial location of an object. The bounding box is rectangular, which is determined by the x and y coordinates of the upper-left corner of the rectangle and the such coordinates of the lower-right corner, that is the case for the annotated data for Yolo and yield outputs.

To create the dataset for traning the yolo architecture. I used at first the Yolo Label [7] and labelImg [8] both support the input format for training. But I figured later that the annotation is faster on LabelImg and more easier.

I annotated about 2K images taking from streams and Youtube videos, the annotation took about 2 days.

To get the following files, used by the Yolo training process, where we have the two classes (player and head) and the coordinates of the bounding boxes. The **0** class stands for the player and the **1** stands for the head class. This annotation is coming from an image where we have two players, thus we had to annotate 4 bounding boxes, 2 for players and 2 for their heads.

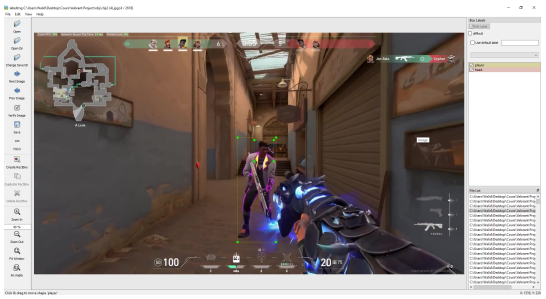

Figure 9. Image annotation using LabelImg

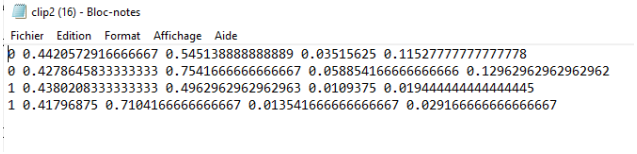

Figure 10. Image annotation output

Even though I thought about data augmentation to increase my dataset, the most used method like flipping, cropping, rotating, translation would effect my bounding boxes and the only left methods are for example adding a Gaussian noise, play on the intensity and the clarity. I referred to related work and I was sure for my task that 2K labeled images are already enough to get good results.

4.2. Yolo Architecture

Compared to other region proposal classification networks (fast RCNN) which perform detection on various region proposals and thus end up performing prediction multiple times for various regions in a image, Yolo architecture is more like FCNN (fully convolutional neural network) and passes the image (nxn) once through the FCNN and output is (mxm) prediction. This the architecture is splitting the input image in mxm grid and for each grid generation 2 bounding boxes and class probabilities for those bounding boxes.

The YOLO model is made up of three key components: the head, neck, and backbone. The backbone is the part of the network made up of convolutional layers to detect key features of an image and process them. The backbone is first trained on a classification dataset, such as ImageNet, and typically trained at a lower resolution than the final detection model, as detection requires finer details than classification. The neck uses the features from the convolution layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates.

The head is the final output layer of the network which can be interchanged with other layers with the same input shape for transfer learning. As discussed earlier, the head is an S × S × (C + B x 5) tensor where S is the size of the grid, C is the number of classes, B is the number of bounding boxes, here multiplied by 5 as we include (x, y, w, h, confidence). The head is 7 × 7 × 30 in the original YOLO research paper with a split size S of 7, 20 classes C, and 2 predicted bounding boxes B. These three portions of the model work together to first extract key visual features from the image then classify and bound them.

The reason behind choosing the Yolo Architecture is the advantages that offers:

- Speed (45 frames per second, thus we can make a real time inference)
- End-to-End training with few changes
- Faster version (with smaller architecture, the tiny version) — 155 frames per sec but.
- Different applications to get inspired from.

4.3. Training the Yolo Architecture

I have done the training on Google Colab using a GPU for free. The steps followed are:

- Setting up the environment by cloning the darknet github repo and fortunately Google Colab comes with cuda installed and set up.
- Define the model configuration and Architecture, I have used Yolov4-tiny.
- Training on the prepared dataset.
- Inference and evaluation.

I have done the training first without the class **head** to only detect **players** and then use the upper part of the bounding boxes detected as the head part. Then I trained using both classes to be able to make multi-class detection.

Among the limiting problems that I faced with this approach is the time taken to train, as it is between 1 day to 2 days.

5. Evaluation

5.1. Method of evaluations

To evaluate the output of the Yolo Architecture, we have different choices, either to choose the Yolo score that is the confidence and it is defined as $P(object)\text{x}IoU$.

The Intersection Over Union (IoU) is used to evaluate the object detection algorithm. It is the overlap between the ground truth and the predicted bounding box, i.e it calculates how similar the predicted box is with respect to the ground truth.

Each grid cell also predicts C conditional class probabilities $Pr(Class_i|Object)$. It only predicts one set of class probabilities per grid cell, regardless of the number of boxes B. During testing, these conditional class probabilities are multiplied by individual box confidence predictions which give class-specific confidence scores for each box. These scores show both the probability of that class and how well the box fits the object.

$$Pr(Class_i|Object)\times Pr(Object)\times IoU = Pr(Class_i)\times IoU$$

The final predictions are encoded as an S x S x (Bx5 + C) tensor.

The other possible metrics for evaluations could be the Average Precision(AP) or Mean Average Precision(mAP), in this project I used the first described metric.

Figure 11. Image annotation output

5.2. Using the pre-trained Yolo weights on coco-dataset

As the characters in the game look like human beings, and the coco dataset contains the class **person**, I tried first the inference using the weights of the different architectures of Yolo trained on the coco dataset. It was interesting to see what I could also as detected objects using these weights.

Actually, with these weights, I was able to detect the players in the **person** class and other sometimes useful objects.

In the Figure[12], we able to detect two characters with a score confidence of 90% and 97%. In the next Figure[13], we are able to detect the two characters and a Tv monitor which an additional class that we do not have in the dataset I annotated myself. In the Figure[14], we completely miss

the character, but instead we detect a person from the images of the characters on the HUD, this however could be avoided using post-processing by ignoring the HUD panel from the detection.

Figure 12. Detected objects using Yolo weights trained on Coco dataset, two characters with 90% and 97% confidence score

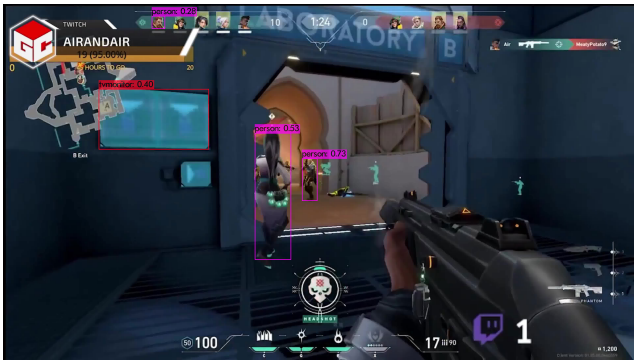

Figure 13. Detected objects using Yolo weights trained on Coco dataset, two characters detected with 53% and 73% and a Tv monitor with 40%

Figure 14. Detected objects using Yolo weights trained on Coco dataset, we miss the character and we detect person on the HUD

5.3. Using the trained tiny Yolo weights on the customized dataset

After training the Yolo tiny 4 on the annotated data, I ran some tests to evaluate the results.

The Figure[15] shows that the weights trained on the customized dataset are able in this image to detect the character and the head unlike the weights trained on coco dataset.

The Figure[16] shows that we are able to detect only the enemy without the head part but in a complex environment.

I noticed that detecting the heads when the character is far is less effective, especially on images with low resolution.

Figure 15. Detected objects using Yolo weights trained on customized dataset, we are able to detect the head and the character

Figure 16. Detected objects using Yolo weights trained on customized dataset, we are able to detect the character only but in a complex environment with smoke

5.4. Detection on Videos

The detection on videos could be seen in the github repository sent with the email.

5.5. Evaluation Results

I ran the test on 200 images, to score them based on the metric presented at the begining of this section. And the

results are as following:

DataSet	Coco Dataset	Personal/Customized Dataset
Architecture	Yolov4	Yolov4 Tiny
Character Detection Score	85,56%	87,76%
Head Detection Score	0%	48,6%

Figure 17. Results of the evaluation for the Yolo trained on Coco dataset and my annotated dataset

6. Conclusion

During this project, I have mainly addressed the main milestones to build to an AI AimBot for detecting targets on the screen then aim.

The next steps that I am willing to finish afterwards, are taking into consideration the detection of other objects useful for the gameplay such as the weapons, the smokes and other items. And also finish integrating the detection with monitoring the mouse to move the crosshair on the detected objects. I am willing also to make a better annotated dataset, by using the results I have from this detection and correct the bounding box if needed, also give more labeled far targets as the results showed a weak performance on far targets, this is coming from the labeled dataset of I focus more on near targets.

7. References

1. A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector, https://arxiv.org/pdf/1802.09567.pdf
2. Visual Object Recognition using Template Matching https://www.araa.asn.au/acra/acra2004/papers/cole.pdf
3. Multiple Object Detection in Images using Template Matching https://www.ijitee.org/wp-content/uploads/papers/v9i1/A5187119119.pdf
4. You Only Look Once: Unified, Real-Time Object Detection https://arxiv.org/pdf/1506.02640.pdf
5. Fast R-CNN https://deepsense.ai/wp-content/uploads/2017/02/1504.08083.pdf
6. Labelimg https://github.com/tzutalin/labelImg
7. Yolo Label https://github.com/developer0hye/Yolo_Label