HOW TO USE CONVOLUTIONAL NEURAL NETWORK TO BUILD AN AI REFEREE

RuiGuo

Culver Academies, Culver, Indiana, USA

ABSTRACT

The purpose of this paper is to solve a significant problem in fencing: high-level referees are expensive, and low-level referees might not make correct and consistent calls. This research project compared a computer-based program to a referee's performance to solve the problem of inconsistency and frequency of errors. The paper has four sections, including background information, methodology, results, and a conclusion. The introduction includes the motivation and location for the research. The background consists of the basic saber fencing rules, defines machine learning, and describes the Convolution Neural Network tool. Next, the methodology section includes four critical subject areas human post estimation, the referee, training an Artificial Intelligence (AI) program, and testing the program's accuracy by examining fencing videos. Only two of the videos tested were judged incorrectly among all fifteen videos tested, and both had valid reasons. Therefore, the program was relatively accurate.

KEYWORDS

Saber Fencing, Machine Learning, Convolutional Neural Network, Referee

1. INTRODUCTION

Data analysis has been widely used in many sports. However, with fewer participants, fencing has not yet tested the benefits of data analysis. Refereeing a fencing match is an opportunity for data analysis. The inspiration for this research originated from a fencing tournament during the summer when an inexperienced referee made calls in error, changing the match's outcome. Therefore, the idea of creating a program that can replace a fencing referee to make correct calls e merged. It is critical to extract data from fencing and write rules as codes to be compatible with the data for implementation. For instance, the inbox rules, the fencer with a faster hand and foot will get the point, can be converted into data science by calculating and comparing the speed of each fencer's foot and hand movements. The basis of an AI referee and data analysis is extracting data from still images extracted from videos. The significance is separating videos into single photo graphs and documenting the positions of the human joints in each image. This step requires machine learning and human post estimation, which a previous study had already completed. A Convolutional Neural Network (CNN), a neural network AI model created on Github, can be trained to detect human movement, blades, and other objects in the image using supervised learning. In addition to writing codes to implement fencing rules, training a customized AI program to identify blades in the picture is also possible. Further explanation of the tools and datasets essential to accomplish this goal is introduced in the methodology section.

There is a similar product online called Allez Go, created by a user called Jason Mo from Medium.[1] He used Openpose to identify fencers in the images. His product does not produce results of who wins the game, Instead, there will be arrows on the screen to indicate which fencer has the right of way. Besides, his model cannot detect blade contacts in the videos. The work presented in this paper is based on detectron 2 from Github. It is a CNN model that has an existed

dataset to identify human figures in the images. It also has a customized version that allow users to train their own models.

After construction, the model was tested with the Culver School Fencing Team and compared to an expert referee. The intention was to determine the accuracy of the program and make improving modifications. Further discussion is contained in the methodology session.

2. BACKGROUND INFORMATION

2.1. Fencing Rules

The critical background information for this project is fencing rules. It is essential to understand how to referee a fencing game. Below are the included fencing regulations used in the program: When there is only one light on, the person who gets the light wins the game.

When both lights are on, the fencer who has the right of way will get the point. Below is the list of situations that a person can gain or for feit the right of way:

1. When two fencers meet in the box, the fencer with the faster first step and who extends the ir arm more quickly gains the right of way.
2. When attacking, the person who holds hands or raises hands loses the right of way. Therefore, the other fencer earns the right of way
3. The fencer who does not have the right earn the right of way by hitting the upper part of the opponent's blade. For example, as demonstrated in figure 1.3, the left fencer gains the right of way by parrying, or deflecting, the opponent's blade.
4. When a fencer positions themselves in a fencing line, when the armandblade are aligned and horizontal to the ground, they automatically gain the right of way. The other fencer can earn the right of way back by hitting any part of the blade.

These are the basic rules for fencing, which are essential to understand AI referee coding discussed in the second part of methodology.

2.2. Machine Learning and Neural Network

This project's most critical and challenging part is object detection, which means locating a specific object among various goods in an image. The proposed tool is the Convolutional Neural Network (CNN), which includes RCNN and masked RCNN. First, however, it is essential to understand the concepts in machine learning (ML), neural network (NN), and deep learning (DL) before introducing CNN.

Machine learning is a significant part of AI and the basis for NN. Unlike simply coding to make a computer execute specific tasks, ML allows the computer to as similate how to perform them by learning large quantities of data. The next step is creating a model with some sample data establishing the variables to as certain how to under take a task. Then, set three categories: training, validation, and test data. Training and validation data can modify the model to fit the situation, predict next week's weather, recognize human facial features, or calculate the probability of winning a gamble. Test data can examine the accuracy of the model.

There are several types of machine learning: supervised, unsupervised, and semi-supervised. Supervised learning means the machine or model is equipped with labeled data with inputs and outputs, usually represented by x and y accordingly. The model compares its predicted results

with the actual results and then modifies the coefficients and uses a weighting system to fit the data better. Unsupervised learning is when the data is not labeled; only the input, x, is provided. The machine or model must find the pattern within the data by itself. Semi-supervised learning uses both labeled and unlabeled data. The NN and CNN used in this project are both supervised learning.

The neural network is a branch of machine learning. The basis of NN is multivariable linear regression, which means many different inputs determine the output. The NN classifies all the input variables as one layer and all the variables of the output as another layer. Additionally, there is a hidden layer between the input and output layers. Like a neuron in the human brain, each input layer is interconnected with each perceptron, a neural classifier, in the hidden layer. The perceptron is also connected to each neuron in the output layer. Therefore, the data training can alter the weight of the classifiers with in the hidden layer and then change the output to fit the data.

Figure1 shows a graph to explain this relationship visually. Figure 2 shows them a th representation of NN. X is the input, W is the perceptron in the hidden layers, and y is the output.

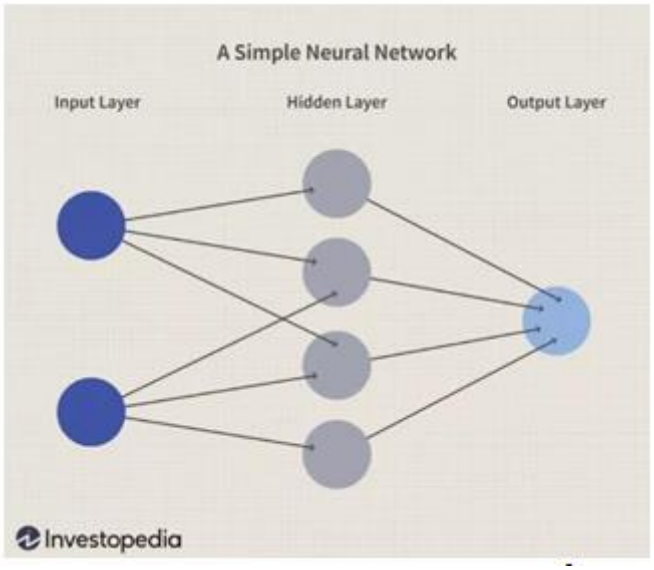

Figure 1. Simple Neural Network[2]

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1,$$
$$\mathbf{o} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2,$$
$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}).$$

Figure 2. Math Representation of Neural Network

Deep learning (DL) is a more complicated version of NN. Instead of one hidden layer, DL includes multiple hidden layers between the input and the output. Multiple layers allow more changes in variables. It also allows the inputs to produce multiple outputs. Figure 3 is a visual representation of DL. Because of multiple hidden layers, there are several W and several x.

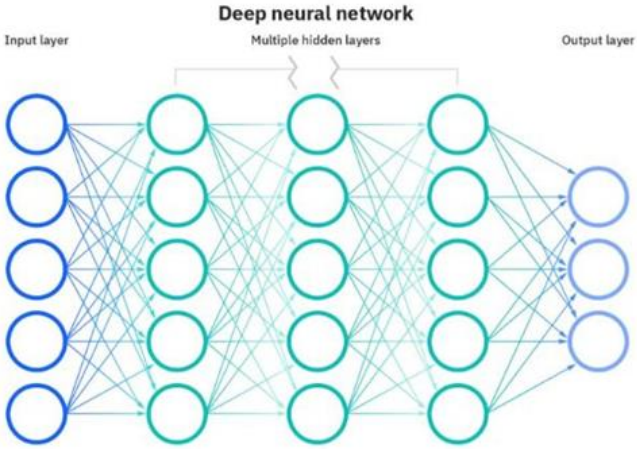

Figure 3. Deep Neural Network[3]

Convolutional neural networks are NN types that focus on visual detection and recognition. Therefore, it has an input, hidden, and output layer. If each pixel is processed by each neuron and connected to the other neuron in the hidden layer, the magnitude of calculation required is probably dozens of powers. Compared to different image recognition tools, CNN has a significant advantage. It reduces the amount of data and analysis to a measurable amount. In most situations, not all the objects in the image matter. Instead, only several factors need to be detected. As a result, the edge of the image, lacking a target object, is detectable by 25 or100 pixels at a time, dramatically reducing the magnitude of calculation. The CNN achieves this through a convolutional kernel, which will scan an area of pixels instead of a pixel. A multivariable equation in the kernel that determines the area the kernel will scan each time. Some variables in the equation include bias, which is b; z^l, which is the output of the last scanned area, also called the feature map; and fandp, which are the kernel coefficients. Figure 4 shows the complete equation.

$$\boldsymbol{Z}^{l+1}(i,j) = \left[\boldsymbol{Z}^{l} \otimes \boldsymbol{w}^{l+1}\right](i,j) + \boldsymbol{b} = \sum_{k=1}^{K_l} \sum_{x=1}^{f} \sum_{y=1}^{f} \left[\boldsymbol{Z}_k^{l}\left(s_0 i + x, s_0 j + y\right) \boldsymbol{w}_k^{l+1} \quad (x,y)\right] + \boldsymbol{b}$$

Figure 4. Full Equation for the Kernel[4]

There is also a simpler and more understandable version of it. N = (W-F+2P)/S+1. N is the output of the coverage area; w is the input of the coverage area; f is the size of the kernel; p is the padding value, which is part of the weights of the model; s is the length of each step moving to the next area. With this method, the areas can be scanned and selected.

After scanning each area, it is processed in the pooling layer, which is critical to reducing the magnitude of the calculation. The scanning areas are merged and condensed into the lower dimension or smaller size in the pooling layer. There are several types of pooling. The most common type is lp pooling, which filtrates the pixels by an equation with set coefficients. There are also mixed pooling and random pooling, which are different ways of extracting the pixels during the filtration process. The pooling output can process then ext area and further identify the hidden layers. This scanning, pooling, and processing in the hidden layers will be iterated in every scanned area. Finally, identifying the objects in the image will combine with the output

Because of the advantages of CNN, it is combined with many other tools to produce favorable results. The first case is RCNN, which combines CNN with image classification. The image is first processed through selective search to extract hundreds or thousands of regional proposals. Then, each regional proposal is processed by CNN to find the bounding box and reflect the result into feature maps after the pooling layer. Next, the results are classified through SVM to separate the regional proposals into different categories. Finally, the categories are trained by bounding box regression. However, this method has a significant problem: it takes a lot of time and storage in the computer to process the proposals in CNN and categorize the min SVM.

Because of the drawbacks of RCNN, FAST-RCNN is developed. The first steps are the same as RCNN, and the feature maps are produced after the pooling process in CNN. The feature maps are then run through Region of Interest (ROI) pooling, which standardized the pooling size for training. The result can then be trained through several methods, including bounding box regression, Softmax Loss, and Smooth L1 Lost. Compared with RCNN, Fast-RCNN skip the step of classification by SVM, which takes a tremendous amount of time. Additionally, users can process the standardized pooling result more efficiently with the training method. Another more advanced version, FASTER-RCNN, is also developed to improve on FAST-RCNN. Instead of using selective search to extract the regional proposals, this tool uses Regional Proposal Network (RPN) to generate regional recommendations.

It is also necessary to introduce some tool sand programs used in trainings and applications related to CNN. The first tool is COCO annotator, a program designed to label objects and critical points in images for the training process in machine learning. It can use rectangular boxes and a key point tool to mark specific objects' names and locations in an image. This information can be used to train through RCNN. However, COCO annotator can not run without docker engine, a supporting program. The methodology section includes the links for downloading COCO annotator and a docker engine. Some specific functions in the COCO annotator packet, such as COCO evaluator, were used to predict and check the results. Tensorflow, an open-source machine learning library, was also used in the code to graph the prediction results.

3. METHODOLOGY

Author names are to be written in 13 pt. Times New Roman format, centred and followed by a 12 pt .paragraph spacing. If necessary, use superscripts to link individual authors with institutions as shown above. Author affiliations are to be written in 12 pt. Times New Roman, centred, with email addresses, in 10 pt. Courier New, on the line following. The last email address will havean18 pt. (paragraph) spacing following.

3.1. Human Post Estimation

The first part of the methodology focused on extracting images and labeling them. Pytube is downloaded and installed from Github at this part of the process[5]. Next, Pytube was used to download videos from YouTube and extract them with OpenCV. The next step was importing some essential tools like Pandas and OpenCV. YouTube videos can be downloaded by obtaining links using the download function from Pytube. Finally, the downloaded videos can be used by uploading to a Google drive and synchronized by using Google Colab.

Then the videos were spliced into images using a python function. In the function. the video capture tool from OpenCV could read the videos and separate them into images. After creating an empty set, a loop was introduced to implement a video capture tool to read the frames and name them accordingly. Once all the frames were named, the frames were returned to the empty set. It was 60 images per second. These images were saved in another file for labeling.

The next step was using a prediction model to estimate human poses in the images extracted from the YouTube videos. The program predicting human figures in the images was Detectron 2. It was used by Face recognition in Facebook, incorporating with Pytorch and Tensorflow. For acknowledgment, the Detectron model for the human figure was trained by Yousry Mohamed, an engineer. He used thousands of images of human figures in different situations to train the model to recognize them. He implemented a Docker engine and used a labeling tool called COCO Annotator to train the model. The model used a data file from the COCO annotator file, as indicated in the red characters. A line in the code was used to adjust the threshold for the model. If the threshold was closer to1, human figures far back would be identified. A threshold of 0.3 or 0.4 could only identify humans close to the camera.

However, the model itself was not good enough, as it only predicted the position of the human body in the images. The x and y-axis were established in each image to quantify the data. Additionally, vital points were labeled and given coordinates to the joints. Matplotlib was used to plug in the key points. CV2 could read the images in the im_name file labeled by the Detectron model. Then the pred_keypoints tool in the model was implemented to label the joints. Eachpoint's x and y coordinates were displayed accordingly in the empty set "outputs_predict_list," which made each point easier to be located. There was an x-axis and a reversed y-axis and 17 joint points for each fencer. The scope of the images could be customized by changing the variables in the x and y axis and time.

At last, the coordinates of the critical points were saved in a python data file and downloaded. The data would be helpful in the following two parts of the methodology.

3.2. Referee

This part introduced how to use the data and images obtained in part 1 to referee a fencing game. Some basic setups were introduced: First, a python library called NumPy was used to create arrays and locate the data points. Next, Matplotlib, another python library was used to graph the data points. The next step was importing the Python data file downloaded in part1.

Matplotlib was able to graph out the data points in theNPY file in each frame. The NPY file was a three-dimensional array. All the frames can be displayed by changing the number of frames, with 0 representing the first frame. Frame_key points was defined as the data point in the first frame. X was defined as the x coordinate of the 34 points and y as the y coordinate. The x and y limits set up the coordinate system and the scatter function in Matplotlib to graph the points. However, the coordinates of the points were unclear unless they were labeled. Therefore, using the label tool to adjust the text, as indicated in a loop.

The next step was obtaining each frame of the fencer's joints, including the coordinates and data points. However, before using the points and applying fencing rules, it was essential to determine when the bout started and ended. Determine the start as easy as usual, and the first frame was the start of the bout. However, most of the videos did not stop immediately when the fencers made the touch. Hence it was essential to determine when the fencers made the touch and stop right there.

A loop of 70 was introduced as most videoswereshorterthan70frames. As the x coordinates of the point in each frame defined x23, y23 was also defined as all the y coordinates. The 16th and the 33rd points, which represent the feet of each fencer, were taken. To hit each other, the two fencers needed to be close enough. After several tests, it was found that the correct distance that two fencers hit each other was approximately 80. Hence, a loop was used to check every frame and stop until two fencers have a distance shorter than 80. After these steps, the frames between the

first and the last determined the result.

The most important part was using the frames and implementing fencing rules to determine the result. The first step extracted the coordinates for the critical points to the referee's decision. The significant parts of the body were hands and front foot (see the background for more information). The coordinates of the fencer's hands and feet in the first image and the image that their distance was shorter than 80 were required. From the picture of the scatter plot above, it was apparent that the corresponding label of joint points were 16 and 32 for each fencer's foot; 10 and 27 for each fencer's hand.

The list of coordinates, which was the keypoints_list, was used. x0 was defined as all the x coordinates of the points and y0 as all they coordinate. Then the 16th and 32nd from the x0 andy0 were obtained and printed. The 0 number in the square bracket in the first roll represents the first frame. Any frame could be received and displayed by changing the number there. So, if the loop output x, from the prior session, was placed in the bracket, then the end frame coordinates could be acquired. The brackets in the fourth and fifth and 9th and 10th line mean the label of the points, which were 16 and 32 feet from each fencer. To obtain the coordinates of the fencer's hands, change the number to 10 and27.

Nonetheless, there was a problem with the Detectron program. Typically, points from 1 to 17 represent the left fencer, and18-34 are indicative of the right fencer. However, in some cases, the label switch position (1-17 represent right and 18-34 represent left) in the later frames. Figure 5and6 shows an example of the situation.

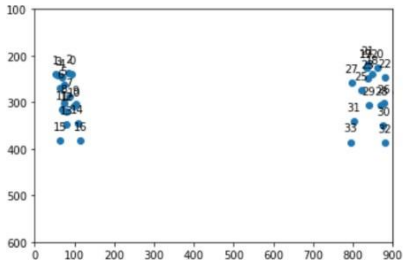

Figure 5. 10th Frame

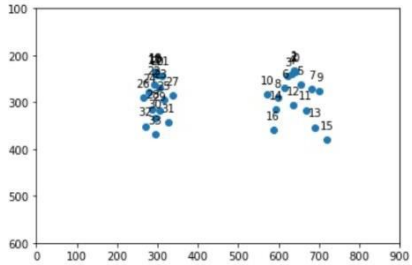

Figure 6. 50th Frame

The image on the left was the 10th frame; on the right was the 50th frame. It was evident that on the 10th frame, points labeled from 1 to 17 were on the left while it was on the right for the 50th frame.

Listing all the situations where the labels switch positions could be a temporary solution. Take the

example above; for example, x_50_16 - x_10_33 would provide the result of the distance moved by the right fencer from the 10th frame to the 50th frame.

The difference of the same joint of both fencers was essential to determine what exactly the situation was. For instance, in the 10th frame in the example above, the x value of point 33 was more significant than point 16. On the other hand, in the 50th frame, the x value of point 33 was less than point 16. Therefore, if the value of x_33 – x_16 changes from positive in one frame to negative in the other frame, the label switches position. Thus, changing the compare signs and the corresponding joint point labels could solve all the situations.

The most common situation in saber was in boxbout, which means two fencers meet in the middle of the strip and hit each other. In this situation, although the speed at the beginning was critical, the acceleration process, or the change of speed, was more critical. The equation to calculate acceleration concerning velocity was the derivation f velocity (dv) means the rate of change over time. The equation of velocity over time was. As a result, acceleration was the second derivative of the equation for distance changed over time. The most accurate way to calculate the acceleration was extracting the x-coordinate of a fencer's hand or foot in each frame, plugging them into a scatter plot, and finding the second-degree polynomial regression. The second derivative of the equation of the regression line was the acceleration. Table 1 and 2 shows the data extracted in a video for both the left and right fencers.

Table 1. Left Fencer

	left
1	521.05737
2	527.32953
3	549.49017
4	563.2066
5	578.80103
6	595.16315
7	611.76178
8	627.90259
9	637.76819
10	646.73865
11	652.12103
12	655.23694
13	655.46777
14	656.00336
15	657.29529
16	657.6087
17	658.86743
18	662.18152
19	664.7843
20	666.39856

Table 2. Right Fencer

	right
1	216.50954
2	216.19743
3	216.74138
4	217.68121
5	218.04636
6	217.52507
7	217.9493
8	217.69798
9	218.96249
10	222.11031
11	225.23923
12	232.12729
13	240.92894
14	254.03465
15	270.6965
16	286.11227
17	299.31381
18	306.48978
19	306.64795
20	308.13843

The trend line tool in excel could generate a second-degree polynomial based on each chart. Figure7and 8show the scatter plot as well as the polynomial regression.

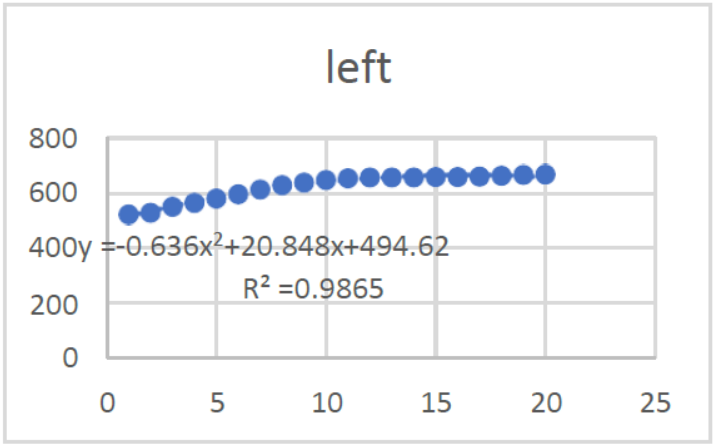

Figure 7. Left Fencer Scatter plot

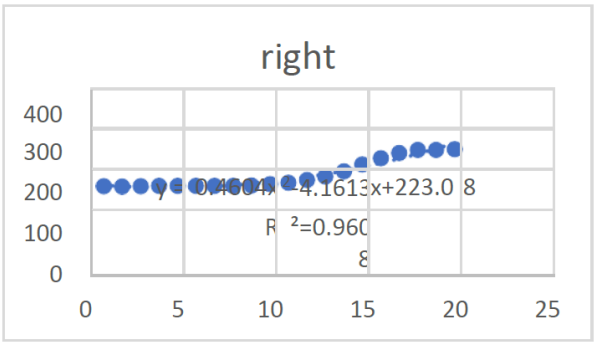

Figure 8. Right Fencer Scatter plot

The second derivative for the left fencer was -1.272, which indicates that the fencer was slowing down. The second derivative for the right fencer was 0.9208. Therefore, the right fencer wins in this situation.

However, such accuracy is unnecessary to determine who wins the about. Instead, merely comparing the difference of frames is sufficient. To calculate e acceleration, extract the data from the frame in the middle of the first and last frames, so if the last frame was frame 44, the middle frame was 22.. Then the hand and foot distance moved from the first frame to the middle frame and from the middle frame to the last frame were calculated. If the difference between the first and middle, and the middle and last was significant, the acceleration was great.

Halfx was defined as x/2 and rounded resulting in the number of the middle frame. The middle frame data could be extracted using the same code above. Other points could be extracted by entering the number in the square brackets. These coordinates are used to calculate acceleration. Although the calculation of only three frames (first, last, middle) might seem inaccurate, it was sufficient to determine which fencer was faster. Suppose higher accuracy of calculation was required, which was the case when analyzing the characteristics of each fencer. For instance, the coordinates of ¼ frame or ¾ frame could be used to calculate and compare the difference.

The first step to calculate acceleration was to find the distance moved by each fencer from the first frame to the middle frame and from the middle frame to the end frame. For instance, x_23_32 represents the x coordinates of the left fencer in the last frame; x_11_16 represents the x coordinates in the middle frame; x_0_16 represents the x coordinates in the first frame. Therefore, a represents the difference between the end and middle frames. Letters were used to mark the difference between the frames for later use. The difference between the middle and first frames is represented by aa. As band bb calculate the right fencer in the same way. The value of a, aa, b, and bb were printed. Calculations of hand movements were completed using the identical method. The only difference was that the points were no longer 32 and 16: they were 10 and 27, representing the hands of left and right fencers.

The formula could produce a value for acceleration by calculating the distance between the end frame and the middle frame divided by the number of frames and the between the initial frame and the middle frame divided by the number of frames. The four letters, e, f, g, h, all given the initial value of 0, were used to represent the result. Some more letters were used to represent the distance moved in the start process and acceleration process. A was the start process of the leg of the left fencer, and aa was the acceleration process. Then band bb were used for the right fencer. Next, c was the starting process of the hand of the left fencer, and cc was the acceleration process. Likewise, d and dd were for the right fencer. Hence, four letters, o, p, q, and r, were used to mark the calculation result. Expressly, o and p represent the index for leg, and q and r represent hands.

The next step was to compare the index of legs and hands between the left and right fencers to determine who was faster. Because the speed of the leg was more important than hand in saber fencing, the weighted method was used to determine who was faster accurately. The ratio between the weight for leg and hand is 7:3. Therefore, e represents the leg index if the left fencer was faster; f represents the leg index if the right fencer was faster; g represents the hand index if the left fencer was faster; h represents the hand index if the right fencer was faster. Calculating the values of the four letters indicated the result.

Next, each of the fencer's hand and leg indexes were added, determining response speed. The next step included comparing the total values. However, when the difference between the speed is too small, referees tend to call a draw, not rewarding points to either fencer. As a result, this situation was added to the code. Therefore, if $e<2$ and $g<2$, the difference between the speed was minimal, the result would be simultaneous. Otherwise, the fencer with them or eextensive total index wins the game.

Additionally, the second rule was referencing holding, was added. As previously mentioned, holding is called when the attacking fencer holds their arm back or raises it. If both fencers make contact, the fencer committing the hold would automatically lose the advantage and the point. The point corresponding 10 and 4 for the left fencer and 20 and 27 for the right fencer were used to determine a hold. The middle frame was crucial because it was usually holding if a fencer’s arm did no text end in them idle of the attack. If the waist and hand distance was smaller than10 in the middle frame, the fencer was committed to holding. The code was then merged with the original code that determined who was faster. However, the problem of switching labels between left and right fencers also exists. As a result, all the situations need to be listed and solved.

3.3. Train an AIReferee

The first two parts of the fencing rules are included in the code now. However, the following two parts involved in the cutting blade were more complex. The reason was that the current Detectron model could only identify human figures from the images. Therefore, a customized AImodel was necessary to identify blades in the images. This lengthy process involved preparing a large amount of data and training and testing it.

Sizable data was required to train an AI program to identify blades. This data included images containing as a berblade. Some basic setups need to be done before creating and labeling the data. First, install docker from the website[6]. Another critical tool for data labeling was the COCO annotator, which could be downloaded in Github[7]. Once registered an account on the Docker engine and COCO annotator, they could label images and create datasets. However, sometimes the page for COCO annotator would not load. Some commands in the command prompt could solve this issue. The first step is to locate the COCO-annotator file. Then utilize the "docker-compose down" command to shut down the docker engine. Finally, use "docker-compose up" to restart Docker engine. After that, the COCO annotator should be ready to run.

The first step to label the image in COCO annotator was to create a category. Users could customize the category's name, which was saber in this case. The next step was to create dataset. The name of the dataset might vary, but the default category for the dataset must be the one just made, which was saber in this case. Three datasets were created to put train, validation, and test data. After creating a dataset in COCO annotator, some images containing blades were placed into the folder. The images could be obtained by downloading fencing videos and running the program in part 1 that different videos to images. These images would be placed in the file with the name of the data set in the COCO annotator folder. The images would appear on the webpage after several refreshes.

The rectangular box and critical points were used to label an image. First, the rectangular box could locate the object's position in the image. Then, key points were placed on the object for more accurate recognition. This step consisted of hundreds or even thousands of images for correct labeling. After labeling, the images and data were export edin to a JSON file and downloaded to the computer. The JSON files were then placed into a folder containing three files: train, validation, and test. The original images of each corresponding category would also be placed in the files. Finally, the folder with three files was zipped and uploaded to google drive to process in a customized AIprogram.

The first step is installing some basic setups. First, the zip file was imported after connecting toColab. Unzipping the zipped folder involved using a tool called zip file. Next, the three files in the folder would be read and processed by the COCO annotator instance. The train data was then read by metadata and placed in a separate catalog. The descriptors of top, middle, and bottom were given to the three critical point soft he blade. The training data would be separate edin to many iterations, and the model would run each iteration at a time. After each iteration, validation data would be used to test if the model meets the standard in the iteration. If it does not, then the iteration would run again.

To train the data, the CFG model was imported. The model was based on CNN to separate the layers of each image and detect the objects. However, if the CFG model were completely standardized initially, it would take a long time to train it because no initial weight was given. Therefore, a COCO file labeled human body was imported to provide the initial weight for the model. The weight would then be changed as each iteration ran. Some other factors were set too. The warm-up iteration, which means the iteration used with the human labeled file, was set to be1000.The iteration for the training data was 500 because there were approximately 500images Right now. The batch coefficient, which means the number of groups of images processed at each iteration, was 2. The basic linear regression coefficient was 0.0002. The smaller this number, the more accurate the model and the longer it took to train.

The result would be printed after the model ran, usually taking several minutes to several hours. There were approximately 500 images in total, more than 20 iterations. Each row shows the number of images in the corresponding iteration. The time it takes to run the iteration, accuracy, loss coefficient, and other information were also included. As more images were included in the iteration, the lose coefficient, the total_loss in the image, decreased. The decrease shows that the model was more accurate as more and more images were placed. A scatter plot of loss coefficient vs. the number of iteration and linear regression was generated. Figure 9showsthescatter plot.

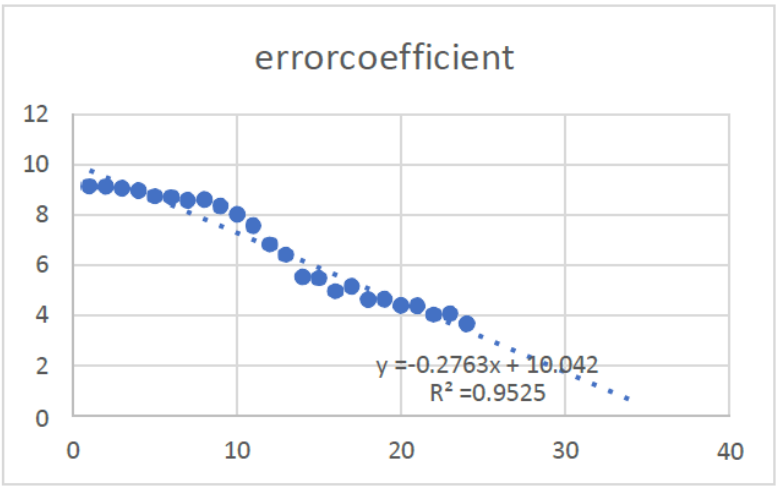

Figure 9. Error Coefficient

As shown in the graph above, the lost coefficient decreased as the number of images increased. The linear regression predicts that the lose coefficient would reach 0 in approximately 35iterations, which means the accuracy would be very high. Tensor Board, a visualization tool kit enabling the tracking of metrics could also generate some more information. Figures 10 and 4.3 were some examples of it.

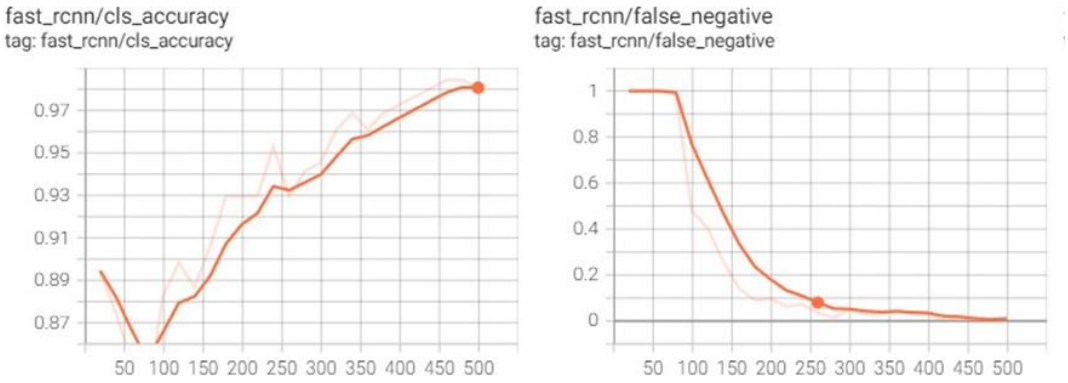

Figure 10. RCNN Evaluation Accuracy Graphs

The x-axis in both graphs means the number of images been processed. As a result, the accuracy increased, and the loss coefficient decreased as the number increased.

Figure 11. RCNN Evaluation Accuracy Graphs2

The loss of the rectangular boxes and key points also decreases as the number of images increases. Now the model was trained. Finally, test data could be put into a Mata data catalog and test the model's accuracy.

3.4. Test

The AI referee program was first tested daily in the Culver saber fencing team. Before conducting the test, an Institutional Review Board (IRB) approval was required. There were three parts of the IRB application process: take the online ethical course, fill the IRB form, and get all the participants to sign the consent form. The IRB form and consent form were attached in the appendix. After the IRB process, videos of fencers fencing were recorded every day. The videos were then processed in the program and produced the results. The results were compared with the results given by a national-level referee in China to determine the accuracy of the AI program. There was a total of 15 videos. The predicting results and the actual results are shown in table 3.

Table 3. Predicting Results & Actual Results

video	AI referee result	actual result
1	right	right
2	left	left
3	right	right
4	left	right
5	left	left
6	left	left
7	left	left
8	left	left
9	right	right
10	right	simultaneous
11	left	left
12	left	right
13	left	right
14	right	right
15	left	left

There were only two mistakes out of the 15 games. The predicting result for the 10 th one was the right fencer win while the actual result was simultaneous. The mistake was caused by the standard for simultaneous in the program, which was 2, was too small. When the value was adjusted to 7 and rerun the program, it produced the result simultaneously.

The situation with the 13th was more complicated. The program indicates that the end frame was the 20th, as the distance between the two fencers was close. Figure11 shows the 20th frame.

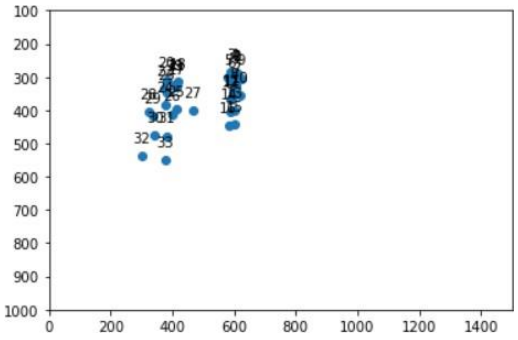

Figure 12. The13thSituation

Although key points from 17 to 34 represent the left fencer, 1-16 did not represent the right fencer. The error resulted from a mirror in the background. As a result, points 1 to 16 labeled the left fencer in the mirror. Figure 12 is a screenshot of the video, showing how the mirror affects the result. This problem could probably be solved by adjusting the threshold in Detectron 2 to ignore the figures in the mirror.

Figure 13. Screenshot

4. RESULT

The accuracy of the program for the Culver Fencing Team was 86.67%, which was acceptable. The accuracy is very high, considering the two mistakes caused by problems outside the code and can be easily solved. Figure13 shows a pie chart that illustrates the accuracy.

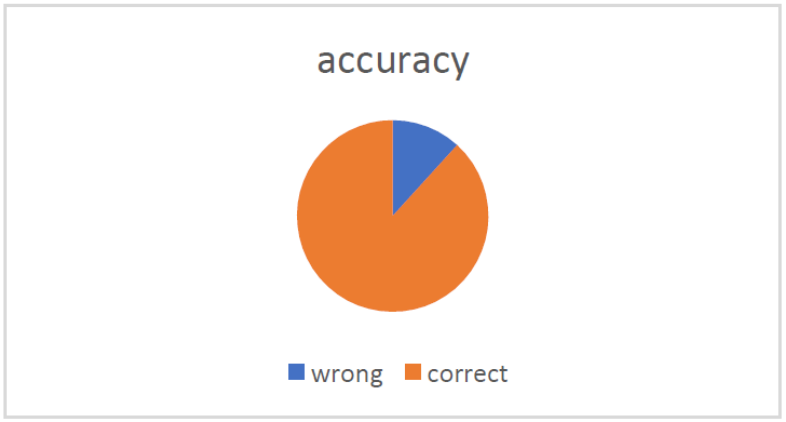

Figure 14. Accuracy PieChart

The accuracy of the customized AI program is also very high. The loss coefficient is decreasing, and with several more iterations, the precision should be good enough to detect the position of blades. Figure 6.2showssome examples of the prediction of the model.

Figure 15. Prediction Example

As shown in the images above, some predictions are accurate though some are not. The model will be more accurate as more data are trained.

5. CONCLUSION

The paper intends to solve the shortage of referees in fencing. The author is an experienced saberfencer and wants to develop a computer program to replace human referees in tournaments. There are four program sessions : identifying human subjects in the images, implementing saberfencing rules with the data; training a customized AIprogram ; test the accuracy of the program. The result is impressive: 86.67% of the games are judged correctly.

The success of AI referees has considerable implications in fencing. With an accurate computer program to produce the result, one can solve these fencing situations without professionals present. The programming ht be ideal for high school event organizers who struggle to hire referees who can give correct calls while not expensive. The AI program can also be an assistant to national-level referees. AI referees can provide a higher quality of competitions in low-level tournaments and encourage more people to the fence as the game is fairer than before.

REFERENCE

[1] Jason Mo, “An Intro to AI in Fencing”, Medium, Oct 28th, 2020. Available: https://thejasonmo.medium.com/artificial-intelligence-in-fencing-

6cd3b9a57859#:~:text=Founder%20of%20Allez%20Go%2C%20the%20world's%20first%20AI%20fencing%20referee.

[2] James Chen, "Neural Network", Investopedia, Dec 8th, 2021. Available: https://www.investopedia.com/terms/n/neuralnetwork.asp

[3] IBM Cloud Education, "Neural Network", IBM, Aug 17th, 2020. Available: https://www.ibm.com/cloud/learn/neural-networks

[4] "Convolutional Neural Network", Baidu, Dec 18th, 2020. Available:https://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%

[5] BD%91%E7%BB%9C/17541100?fr=aladdin

[6] Ssuwani,"Pytube",Github,May20th,2021.Available:https://github.com/ssuwani/pytube

[7] Dockerinstall,docker.Available:https://docs.docker.com/engine/install/

[8] COCO annotator, Github, Available: https://github.com/jsbroks/COCO-annotator

AUTHOR

Rui Guo, high school Junior from Culver Academies in Culver, Indiana. Very interested in STEM and computer science.