NBAction for Real-time Basketball Action Detection and Classification using Computer Vision

Simon Lin
Department of Computer Science, Faculty of Science
Toronto Metropolitan University
Toronto, Canada
s1lin@torontomu.ca

***Abstract*—Basketball is a high-paced game requiring precision, coordination and quick decision making. Understanding player actions is crucial for analysis and enjoyment of the sport. There are few systems available for live analysis of actions within dynamic sports environments which are common in basketball games. This paper presents *NBAction*, a real-time basketball action classification and detection system utilizing computer vision techniques and frameworks. NBAction classifies common basketball actions such as shooting, scoring and defending, while also keeping track of the players, basketball and the location of the net. Our system employs a combination of self-trained and preexisting deep learning models for object recognition and classification to ensure high accuracy in a variety of basketball scenarios.**

I. Introduction

Basketball's dynamic nature creates a demand for standardizing the understanding of players' actions which is imperative towards detailed analysis and consumer involvement. Despite the exponential growth of computerized systems and deep-learning in improving sport systems, the usage of real-time classification and detection in dynamic and complex environments has yet to be implemented into many basketball-related systems. Specifically, our system includes tracking the basketball, current player and defender positions, and hoop locations. NBAction runs in real-time, displaying a uniform, live, and accurate stream of information to the user. Our approach employs a fine-tuned object detection model we trained through data in the form of images and videos collected by our team. The model was trained with a preexisting deep-learning architecture, leveraging Roboflow for labeling, classifying, augmenting and managing high-quality input for training. Roboflow streamlined the process of preparing and diversifying our dataset, allowing accurate performance in various lighting conditions, backgrounds and angles. To train our model, we used Ultralytics YOLOv8—an object detection framework for Python, recognized for its speed and accuracy in detection and classification. By tailoring our model to detect specific actions in basketball games, NBAction achieves high accuracy and reliability. We integrated our model with the OpenCV library for real-time video processing, object tracking, and labeling actions. OpenCV enables NBAction to dynamically relay visual feedback by rendering object bounding boxes, displaying ball trajectories and stabilizing the hoop for accurate results. The remainder of this approach centers around our scoring algorithm to analyze spatial relationships between the basketball and hoop to determine when a successful shot is made. Finally, our system displays all relevant information to the user through a simple, intuitive, and engaging video overlay.

II. Related Work

Deep-learning and action detection and classification in the field of basketball or sports is fairly minimal. While researching preexisting models to expand off of we came across several computer vision models trained specifically to track players, referees or the basketball during games. However, there was little to no focus on detecting specific actions in games. These works focused primarily on being easily transferrable between sports, rather than acting as a tool to help define basketball games. For instance, a vision model can be easily generalized to track players, a ball, and referees. However, without additional context, it could be unknown whether it is analyzing basketball or a similar sport, such as soccer. Many implementations of computer vision in basketball are on a smaller scale and leave much to be desired in both effectiveness and practicality. Such projects may track the basketball or identify players but fail to focus on vital actions that define the sport, such as a basketball's trajectory as it enters a net or a prominent defender. As such, a combination of deep-learning models and computers visually processing player actions is the next logical step in innovating basketball and sports technology. A current utilization of deep-learning models in the National Basketball Association (NBA) is ball-centric camera [1] framing—a system employing object detection models to track the ball and center it in the frame of a camera. The development of this system has vastly improved the viewing experience for fans of the sport by offering smooth focus on the most crucial element of the sport: the ball. Aside from ball-centric camera framing, many systems can yield accurate results, but are often computationally expensive, limiting their accessibility to those with powerful technology or budgets from professional leagues [1]. In this work, we aim to develop a system offering a similar level of revolutionizing how basketball is analyzed and experienced. NBAction aims to fill the gap in specialized sports models and focus on detecting definitive actions commonly performed in basketball.

III. Methodology

A. Preliminaries - Data Collection and Training the Model

The object detection model was trained on a dataset consisting of 936 unique total images (736 before augmentation). Images were primarily gathered through taking pictures and recording videos of games at local indoor and outdoor basketball courts. Other data was manually uploaded from past recordings of university basketball games or stock images of certain basketball actions being performed. The dataset divides objects of interests into five classes:

- Basketballs (644 occurrences in the set)
- Basketball Hoops (390 occurrences in the set)
- Players (1253 occurrences in the set)
- Defence (300 occurrences in the set)
- Shooting (344 occurrences in the set)

The overall split of the dataset was:

- Training set - 81% of the dataset (756 images)
- Validation set - 13% of the dataset (121 images)
- Test set - 6% of the dataset (59 images)

After applying augmentations from Roboflow [2], 936 unique total images were produced for the dataset. We employed preprocessing to scale low resolution images up to a resolution of 640x640 pixels.

B. Preliminaries - Video Capture Attributes

The ability for NBAction to track key elements such as the location of the ball, hoop and players relies on the initial processing of individual frames from a video capture. Using OpenCV's *VideoCapture* function [3], frames are processed one-by-one where our program actively performs real-time detection and classification. We define the following variables for frame processing: C, an array of three dimensional values where each element represents individual pixels in the current frame [5].

- Total rows in C correspond to the Height of the frame.
- Total columns in C correspond to the Width of the frame.
- The third dimension of the array is a color channel [5] represented by their [R, G, B] values.

We also define T, the total number of processed frames in a given video. T keeps track of when an object was last detected in a frame. Next, we setup functions for our shot detection algorithm.

C. Representing Ball and Hoop Locations

We define *ball* (1) and *hoop* (2) as 2D coordinate pairs:

$$ball = (x_{\text{ball}}, y_{\text{ball}}) \tag{1}$$

$$hoop = (x_{\text{hoop}}, y_{\text{hoop}}) \tag{2}$$

Where x and y correspond to current pixel locations of the object in a given frame. We also define L, a list containing an object's recent coordinate pairs [5].

D. Stabilizing the Ball and Hoop in a Frame

A common issue when tracking objects in a dynamic environment is the difficulty in identifying and following objects that travel at a high velocity. When combined with rapid and unexpected camera movements, this can lead to inconsistencies when detecting basketballs that travel quickly across the screen in a matter of frames. Stabilizing the ball and hoop locations can filter any anomalies, resulting in smoother and more reliable displaying of data. We model the stabilization as such:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tag{3}$$

$$d_{\text{max}} = 4 \cdot \sqrt{w_1^2 + h_1^2}, \tag{4}$$

Where d is the is distance measured between the two most recent positions of a ball using the ball's Euclidean distance [6]. We also define d_{max}, where if:

$$d > d_{\text{max}} \tag{5}$$

Then the most recent point is considered an anomaly and therefore removed from the list L. The value of d_{max} scales according to how large the ball appears in a given frame. The same is done for stabilizing the hoop, however the factor for determining d_{max} for a point is reduced to 0.5. This is because basketball nets do not move nearly as much as a ball does.

E. Determining the Scoring Region

The hoop radius is defined as:

$$r_{\text{hoop}} = \frac{1}{2} \cdot \max(w_{\text{hoop}}, h_{\text{hoop}}), \tag{6}$$

where w_{hoop} and h_{hoop} are the width and height [6] of the detected hoop, respectively. We use r_{hoop} to determine the valid scoring region, where a shot is only counted if a ball's center is within this radius.

F. Determining if a Shot is Successful

The Euclidean distance between the ball (1) and hoop centers (2) is calculated as:

$$d = \sqrt{(x_{\text{ball}} - x_{\text{hoop}})^2 + (y_{\text{ball}} - y_{\text{hoop}})^2}. \tag{7}$$

Where d is the distance between the ball and the hoop's center. The shot is successful if:

$$d \leq r_{\text{hoop}} \tag{8}$$

IV. Experimental Setup

NBAction was trained over five iterations before arriving at the final model used in the following quantitative and qualitative data. Each iteration was trained using the following hardware and software setup:

- GPU: NVIDIA RTX 3080 Ti (10,240 CUDA cores)
- CPU: AMD Ryzen 5 5600x Processor (6 cores)
- RAM: 32 GB DDR4

To significantly accelerate training times, the process was offloaded to the GPU. Training on the CPU alone would take

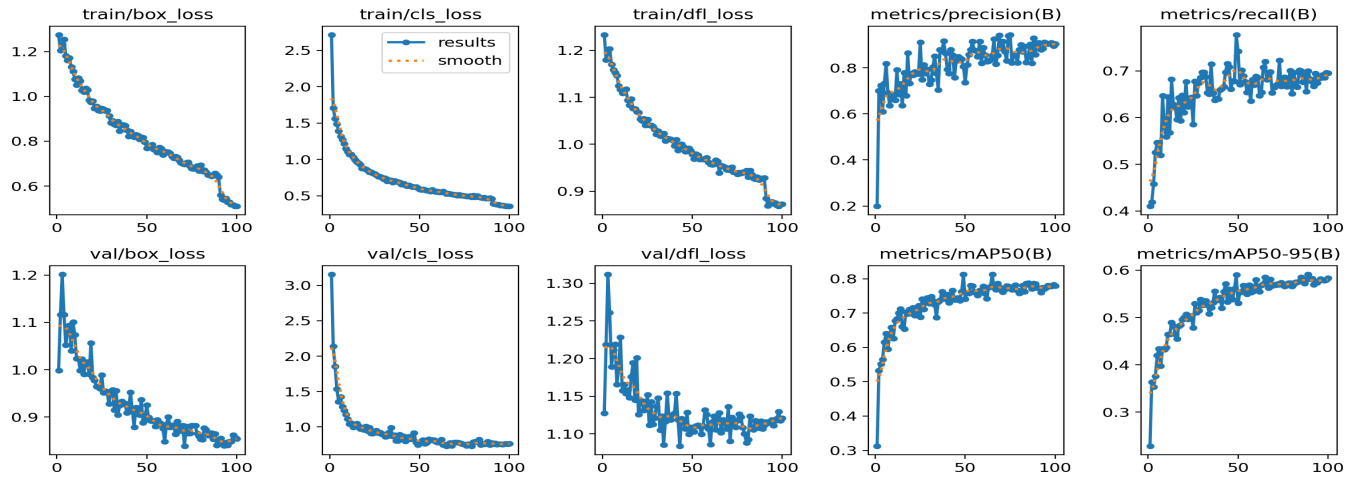

Fig. 1. NBAction Training and Validation Loss Metrics

Fig. 2. NBAction's viewing interface

approximately 2-3 hours per iteration, whereas leveraging the GPU reduced this to just 10-15 minutes per iteration. This dramatic speed improvement highlights the efficiency of using GPUs for deep learning tasks. The training process adhered to Ultralytics' recommended guidelines [4], including:

- Using 100 epochs to avoid overfitting.
- A scaled image resolution of 640 pixels, suitable for medium to large object detection models.
- Allocating 8 workers to ensure stable and efficient training.

The setup required ensuring compatibility between CUDA Toolkit, cuDNN, and PyTorch [7]. While configuring the environment for GPU acceleration posed some challenges, the performance benefits were well worth the effort. Training occupied approximately 18GB of memory.

V. Results

Fig. 1 displays the improvement of NBAction over the course of each iteration.

A. Training and Validation - Box Loss

Both graphs display the error in predicting the bounding box coordinates of the objects. Initially, the training box loss starts at approximately 1.2 and steadily declines to around 0.4 by the 100th epoch. Similarly, the validation box loss decreases from about 1.2 to 0.6 over the same period. The steady decline indicates that our model was learning to accurately predict the locations of objects in the image. Additionally, the decreasing trend in the validation set demonstrates that our model was able to generalize better on unseen data, avoiding overfitting.

B. Training and Validation - Class Loss

These graphs display the classification loss, which measures how well our model was learning to classify objects (e.g., distinguishing between ball, hoop, and background). The training classification loss starts at approximately 2.5 and decreases to 0.5 by the 100th epoch. Meanwhile, the validation classification loss follows a similar trend, starting at around 3.0 [8] and steadily reducing to 1.0. The consistent downward trend in both graphs indicates improved classification performance over the course of the iterations. The significant reduction in validation loss over epochs demonstrates that our model was improving its ability to classify unseen data effectively.

C. Training and Validation - Distribution Focal Loss

Distribution Focal Loss (DFL) [8] measures the precision of bounding box regression. In the training DFL loss, the values start at 1.2 and gradually drop to 0.6 by the 100th epoch, while the validation DFL loss reduces from 1.3 to approximately 1.1. This downward trend in both graphs shows that the model is consistently refining its bounding box predictions, leading to improved precision in locating objects.

D. Metrics/Precision

This graph displays the precision of the bounding box predictions on the training data. Precision refers to the percentage of correctly identified objects in a given basketball frame.

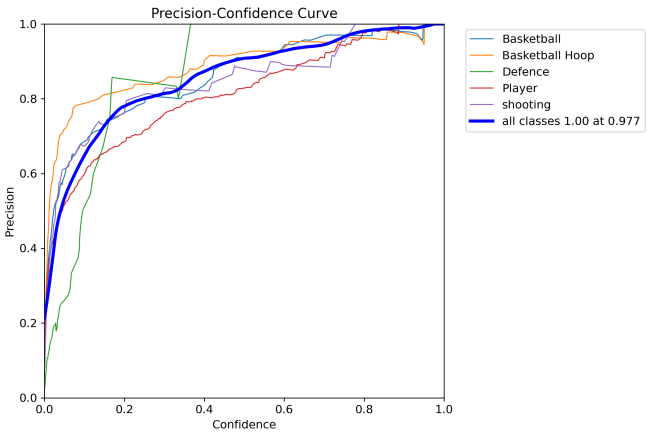

Fig. 3. Our Precision-Confidence Curve

The upward trend indicates fewer false positives. The Mean Average Precision (mAP) is at a 0.5 Intersection over Union (IoU) threshold [8] on the validation set. This metric helps us evaluate accurate object detection performance. The steady increase seen here indicates improved detection capabilities over our iterations. The mAP50-95 [8] is a stricter chart, and is used to display our models robustness, the upward trend shows increasing and consistent detection across 100 epochs.

E. Precision-Confidence Curve

Fig. 3 shows our precision-confidence curve [8], used to analyze our object detection model confidence metrics of predictions, and how precise it performs. The steep curves across each class represent better overall performance at lower confidence thresholds. The blue curve [8] represents our model's overall performance, we see that:

- The model achieves a maximum mean precision of 1.00 (perfect precision).
- This happens at a confidence threshold of 0.977 (97.7%).
- Our Basketball, Basketball Hoop, Shooting and Player classes perform consistently well across confidence levels, as their curves stay high for a wide range of confidence thresholds.
- The Defence class remains lower at some thresholds, we will cover this in limitations.

VI. Discussion and Limitations

NBAction was developed over the course of four months as a final project for CPS 843 - Introduction to Computer Vision. While our system demonstrates promising results in recognizing and classifying basketball actions, there remain some limitations and areas for improvement that could enhance its accuracy, robustness, and scalability.

A. Covering Wider Perspectives.

NBAction performs the best when provided a stable viewing angle from off to the side of a court. Top down or low angle

Fig. 4. An overview of a single training batch in our dataset.

viewing of basketball games poses a challenge for the model's ability to detect actions from unique perspectives it was not trained on. Variations in filming can distort the way players appear when performing actions and may occlude other objects that a wide angle would capture. Further data augmentation on the dataset could help the model generalize to these scenarios, but was not a desire in the current scope of the project.

B. Generalization of the Defence Class

Our *Defence* class refers to players taking a defensive action against another player with the basketball. We witnessed a rather low precision-confidence curve for our defence class which stems from the dynamic nature of basketball we mentioned earlier. Basketball defense is highly dynamic, with a wider variety of strategies and actions that players can take to block or disrupt other players. Unlike more defined actions like shooting, which have a clear set of visual cues (e.g., the shooter's arm formation and the ball's trajectory), defensive plays are often less predictable as it relies on the context of the game that we are unable to provide to our model. As such, the current scope of the project limited us to annotating simple forms of defence as seen in Fig. 3. In the future we aim to incorporate a form of contextual awareness to our model, allowing it to better generalize and detect unseen forms of defence.

VII. Acknowledgments

We would like to acknowledge those at Toronto Metropolitan University's (TMU) Recreation and Athletic Center (RAC) who provided us with the permission to record, upload and include in basketball footage for our dataset to train our model

on. We would also like to thank the TMU Bold basketball team for our usage of basketball games as test data featured in our demo of NBAction.

VIII. Conclusion and Future Work

In this paper, we presented NBAction, a real-time basketball action classification and detection system employing computer vision techniques and deep-learning models. Our approach focused on accurately detecting and classifying key actions such as shooting, scoring, and defense in the dynamic environment of a basketball game. Through training an object detection model with a diverse datasets and utilizing frameworks like OpenCV and YOLOv8, we achieved real-time action detection with reliable accuracy. NBAction represents a step forward in real-time sports analysis and can serve as a foundation for further innovations in basketball analytics and viewer enjoyment. Future work aims to enhance the model's contextual awareness and improve its ability to generalize to complex defensive plays. With additional training, data augmentation, and the integration of more contextual information for less precise classes, we believe the system can become more robust and scalable, offering deeper insights into player performance and game strategies. We hope this work opens up an increased interest of the applications of computer vision in the field of basketball and sports.

References

[1] Liu, Yang and Dinsdale, Dinsdale.R. and Jun, Seong-Hwan and Briercliffe, Creagh and Bone, Jeffrey "Automatic Learning of Basketball Strategy via SportVU Tracking Data: the Potential Field Approach," Submitted to Sloan Sports Analytics Conference, 2015.

[2] Dwyer, B., Nelson, J., Hansen, T., et. al. (2024). Roboflow (Version 1.0) [Software]. Available from https://roboflow.com. computer vision.

[3] G. Bradski, "The OpenCV library," Dr. Dobb's Journal of Software Tools, vol. 2000, Jan. 15, 2008. [Online]. Available from: https://www.drdobbs.com.

[4] G. Jocher, J. Qiu, and A. Chaurasia, Ultralytics YOLOv8, version 8.0.0, released Jan. 10, 2023. [Online]. Available from: https://ultralytics.com.

[5] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

[6] W. Xu, Y. Liang, W. Chen, and F. Wang, "Physica A: Statistical Mechanics and its Applications", vol. 539, p. 122996, 2020. [Online]. Available from: https://doi.org/10.1016/j.physa.2019.122996.

[7] NVIDIA, Vingelmann P, Fitzek FHP. CUDA, release: 10.2.89 [Internet]. 2020. Available from: https://developer.nvidia.com/cuda-toolkit

[8] G. Jocher, J. Qiu, and A. Chaurasia, Ultralytics YOLOv8 Performance Metrics, version 8.0.0, released Jan. 10, 2023. [Online]. Available from: https://docs.ultralytics.com/guides/yolo-performance-metrics.