ShotSpotter

Aaron Chan[1], Ahmad Khan[1], Benjamin Fredette[1], Charles Kelsey[1], and Ryan Shiu[1]
[1]Toronto Metropolitan University, Toronto
{a20chan, ahmad.t.khan, bfredette, charles.kelsey, ryan.shiu}@torontomu.ca

Abstract

ShotSpotter is an intelligent system designed to automate basketball game score recognition by analyzing video footage using advanced computer vision techniques. The system identifies scoring events, by detecting object locations and classifying actions in real-time. Our approach integrates object detection, motion tracking, and activity recognition to achieve high accuracy in differentiating objects on the court and tracking made baskets. Unlike existing systems, which rely heavily on manual input or specialized hardware, ShotSpotter leverages readily available video feeds, offering a cost-effective and scalable solution for basketball analytics. The system's unique capability to automatically update game scores provides significant value for sports broadcasters, coaches, and analysts. Preliminary testing has demonstrated promising results, with the potential for further improvements through advanced machine learning models and larger datasets. Future applications could extend beyond basketball, supporting automated analysis in other sports.

1. Introduction

The integration of computer vision and artificial intelligence (AI) into sports analytics has revolutionized how games are understood, analyzed, and consumed. In recent years, advanced video analysis techniques have enabled automated tracking of players and objects, event recognition, and performance evaluation in various sports, including soccer, tennis, and basketball [1, 2]. These technologies provide unprecedented insights for coaches, analysts, and fans, offering tools to optimize strategies, enhance performance, and enrich viewer experiences.

In basketball, precise scoring and performance tracking are essential for real-time game analysis. Existing systems often rely on manual inputs or specialized sensors, which can be expensive, time-intensive, and prone to human error [2, 3]. While AI-powered solutions such as player and ball detection have been explored, challenges like occlusion, fast-paced gameplay, and environmental variability still hinder their widespread adoption [1, 3].

Our research aims to address these challenges by developing ShotSpotter, an intelligent system designed to autonomously recognize and classify basketball scoring events—such as three-pointers, two-pointers, and free throws—from video footage. By leveraging computer vision and deep learning, ShotSpotter offers a scalable, cost-effective solution for automated scorekeeping and analytics. This project contributes to the field by addressing a critical gap in real-time sports analysis and providing a foundation for future innovations in automated sports technologies.

Motivated by the growing demand for accurate and efficient tools in sports analytics, our research seeks to enhance the reliability and accessibility of AI-driven systems. ShotSpotter has the potential to transform how basketball games are analyzed, benefiting stakeholders across the sports ecosystem—from broadcasters and analysts to coaches and players.

2. Materials

- **Datasets:** Roboflow-preprocessed datasets containing labeled images for training object detection and shot classification models.
- **Programming Language:** Python was utilized as the primary language for implementing the system.
- **Image Processing:** OpenCV was employed for foundational tasks such as frame extraction and court mapping.
- **Shot Classification:** TensorFlow and Keras were used to train a convolutional neural network (CNN) for classifying shot types—free throws, two-pointers, and three-pointers. Transfer learning with pre-trained models was incorporated to enhance accuracy.
- **Object Detection:** YOLO, a real-time object detection algorithm, was employed for detecting and tracking the basketball, players and referees. It is widely recognized for its precision in identifying objects within video frames.
- **Tracking:** A deepSORT tracking algorithm was used to augment shot trajectory data by increasing the number of data points derived from YOLO's output.
- **Model Training:** A high-performance GPU was utilized to train the models efficiently.

- **Testing and Implementation:** A standard computing system was employed for testing and implementing the developed solutions.

3. Methods

3.1 NBA API Data Collection

To construct a comprehensive dataset for training our computer vision model, we developed a two-step data acquisition process. This process involved extracting detailed shot event data from NBA games and then downloading the video clips. The first script, fetch_nba_shots.py, was crafted to collect and categorize shot events, while the second script, adapted from an existing tool [5], facilitated the retrieval of video clips associated with these events.

3.2 Shot Event Data Collection

Using the fetch_nba_shots.py script, we accessed an API client of NBA's official API [6] to gather play-by-play data from the 2023–24 regular season games. The script filtered events to include only field goals made, field goals missed, and free throw attempts, corresponding to specific event message types. Each event was then categorized into one of four result types based on textual descriptions: "MISS," "TWO POINTER," "THREE POINTER," or "FT MADE." This resulted in clearly labeled data that could be of use to finetune a shot categorization model.

By randomizing the order of games and iteratively processing them, we collected shot event data until reaching our target count. In our initial run, we collected 10,000 shot events, creating a robust dataset suitable for training purposes. Each entry in the dataset included important information such as game ID, event number, event type, descriptions, score, game date, and the categorized result.

3.3 Video Clip Retrieval and Labelling

To obtain video footage corresponding to the shot events, we adapted a script from an open-source project [5], ensuring proper attribution. The modified script, download_video_by_gameid_date.py, utilized the shot event data to construct URLs for video clips hosted on the NBA's official platform. By automating HTTP requests and handling various potential errors, the script was able to download the clips; there were no observed instances where a clip was mislabeled.

The videos were saved with filenames containing the game ID, event ID, and result labels. To optimize storage and distribution, the downloaded clips were compressed into a ZIP archive. This method allowed us to collect tens of thousands of video clips with accurate labels, significantly expanding our dataset without needing to manually annotate data.

3.4 DeepSORT Tracking Algorithm

While the YOLO model effectively tracks the ball's motion, it does not generate enough detections to make accurate calculations on the shot trajectory to help with shot classification. We decided to develop a method to increase the number of detections by using an algorithm known as DeepSORT. [4] DeepSORT is an advanced object tracking algorithm that enhances the original SORT algorithm by integrating deep learning-based appearance descriptors. It combines motion information from a Kalman filter with appearance features extracted through a convolutional neural network, allowing for more robust and accurate tracking of objects over time. This approach effectively handles challenges such as occlusions and object interactions by maintaining consistent object identities across frames, which is important for applications requiring precise tracking.

In the context of our project, we implemented the DeepSORT algorithm to accurately follow the basketball's trajectory throughout video sequences. By integrating DeepSORT with our detection pipeline, we processed frame-by-frame detections of the basketball, filtering for high-confidence detections to improve reliability. The tracker was initialized with parameters optimized for our application, such as appropriate maximum age and confirmation thresholds, and utilized a lightweight embedder (MobileNet V2) for efficiency. This implementation enabled us to add detections to the trajectory data by associating detections across frames, even in situations involving rapid movement or partial occlusions of the basketball. The trajectory information can be used to calculate features such as velocity of the ball, distance from the hoop at release and other metrics associated with the ball’s trajectory.

4. Results

4.1 YOLO Ball Detection Model

For ball detection, a YOLO version 5 model was trained using an existing image dataset. Each image was labeled with applicable classes such as basketball, net, player, and referee. An example of the dataset is shown in Figure 1. The dataset contained 6,900 images, divided into 4,900 for training, 1,000 for validation, and 1,000 for testing, providing sufficient data for YOLO detection.

The model was initially trained for 100 epochs with an image resolution of 640 pixels and a batch size of 16. The training was conducted on a GPU with 8GB of memory,

which limited the ability to increase parameter values without causing memory overflow. These parameters were optimized through multiple trials to achieve the best results.

The model's accuracy was evaluated using precision and recall metrics. Precision is the proportion of all the model's positive classifications that are actually positive, and recall is the proportion of all actual positives that were classified correctly as positives [7]. By using the parameters described above, we were able to get a precision of 95.8% and recall of 94.4% which was our 25th epoch. The training model was stopped here to avoid overfitting the model, which is when the machine learning model gives highly accurate predictions for training data but not for new data [8].

The model's performance was validated on short video clips, showing accurate detections, as illustrated in Figure 2. Object confidence scores were consistent and reliable throughout the inferred videos.

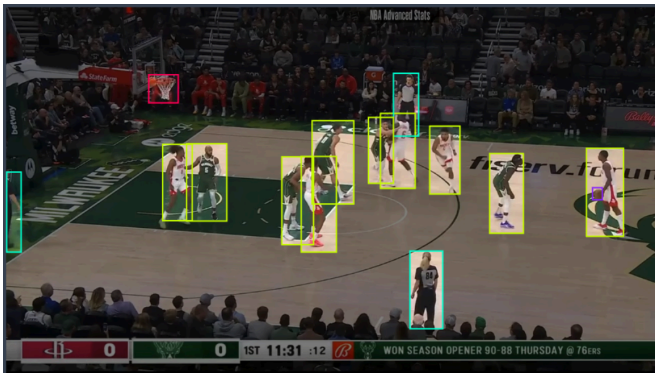

Figure 1. Dataset example image

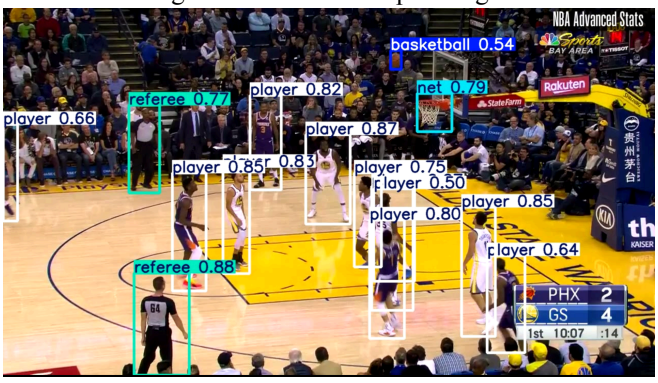

Figure 2. Detection training data inferred

4.2 Court Mapping

For court mapping, a dataset for key point detection of the court was used. The dataset was fairly small compared to the previous dataset as it had 211 images in total. This dataset was relatively small compared to the previous one, consisting of only 211 images. As shown in the sample image in Figure 3, the key points include the court's outer dimensions, the three-point line, the paint area, and the free-throw circle. Due to the limited size of the dataset, the number of training iterations was increased to 500, and the image resolution was reduced to 540 pixels.

Figure 3. Court mapping dataset sample

The model's highest iteration achieved a precision of 78% and a recall of 99%. While this indicates that the model successfully identified key points on important court lines, it also produced a significant number of false positives. These inaccuracies made it challenging to map consistent lines and boundaries for practical use. This issue is illustrated in Figure 4, where the model was inferred to a basketball clip. Although some points are correctly positioned on the expected lines, and all significant lines have at least some accurate points, numerous points are scattered randomly in incorrect areas.

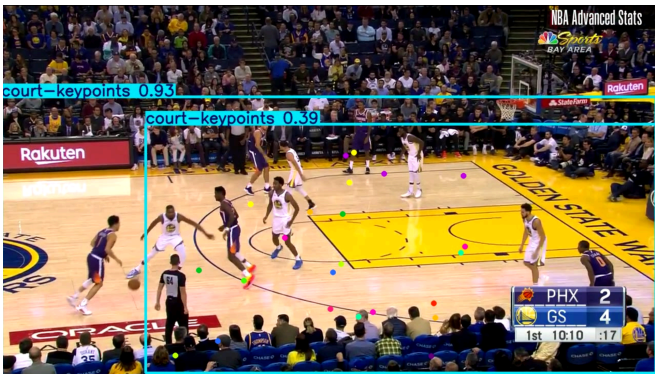

Figure 4. Court mapping inference video frame example

4.3 Frame Extraction and Analysis

For tracking the basketball, OpenCV was used to extract all video frames, and objects were identified within each frame. Classes such as players, nets, and referees were assigned unique IDs, enabling the system to track scores and determine how many points each player has scored. To enhance visual aesthetics, ellipses and triangles were used to highlight objects instead of traditional bounding boxes.

To count points, the basketball's position was tracked across video frames using the YOLO model combined with ByteTrack. Each detected position was stored in a structured format, linking frames with object bounding boxes. A ball interpolation function was implemented to smooth the ball's trajectory and ensure that interactions between the ball and the net were accurately captured. The nets were categorized as net 1 and net 2 to maintain separate scores for each side.

A shot was recorded as successful if the ball intersected with the basket while moving downward. If an intersection occurred but the ball was not moving straight down, the shot was classified as a miss. The system updated the score accordingly and displayed text in the video on the relevant frame to indicate whether a shot was made or missed.

An example of this process is shown in Figure 5. This algorithmic approach was necessary due to the lack of a pretrained model for identifying made or missed shots. Ideally, a pretrained model could be used to classify shots, with an algorithmic layer to refine predictions by adjusting confidence levels based on additional context.

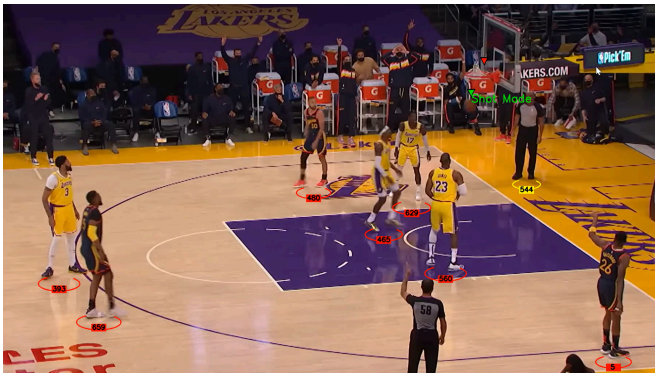

Figure 5. Final inference shot made frame

4.4 DeepSORT Tracking Algorithm

After providing the deepSORT algorithm with the YOLO detections, it was successfully able to increase the number of basketball position data points. With the increased number of detections, a trajectory was plotted to visualize the result. Unfortunately, the efficacy of this method was underwhelming, the model inaccurately predicts the ball to continue to rise when it is about to fall. After analyzing the data, there are simply not enough basketball detections for the model to predict that the ball is slowing down. To utilize this method, we will need to improve our object tracking model to include more high confidence basketball detection.

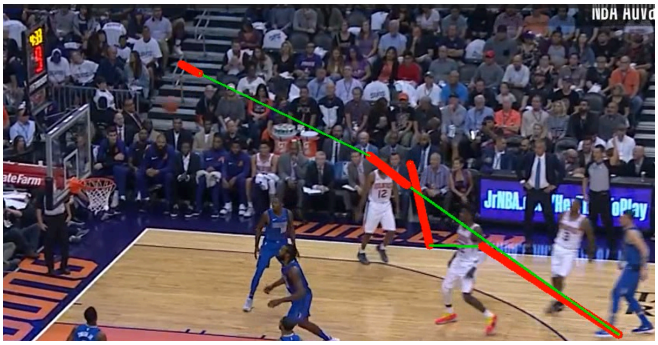

Figure 6. Example tracking result on shot trajectory

As you can see in Figure 6, since there is a lack of data points during the shot, the algorithm incorrectly interpolates the data into a straight line. This has made the trajectory data unsuitable for improving our model.

5. Discussion and Limitations

One of the primary limitations faced during the ShotSpotter project was court mapping. Due to the lack of access to high-performance GPUs, the team was unable to train large-scale datasets capable of accurately mapping the basketball court. As a workaround, smaller datasets were trained, but this approach yielded poor results. Additionally, the frequent camera movements in the footage further complicated the process, making it challenging to maintain consistent and precise court mapping.

This issue directly impacted the second major limitation: shot type classification. With accurate court mapping unavailable, determining shot types became increasingly difficult. While searching for pre-trained models capable of classifying basketball shots, the team found no existing solutions online. Training a model from scratch would have required months of effort and high-performance computational resources. The professor advised everyone to not resort to this type of solution. We also looked into classifying baskets using features extracted from the shot trajectory. Unfortunately, the data available to use regarding the shot trajectory was not accurate enough to make accurate calculations on features related to the basketball's trajectory during shots. Attempts to improve the data using solutions such as advanced tracking algorithms and Kalman filters proved unhelpful in our goal to extract detailed shot trajectory information. To address this, the team implemented a basic algorithm where shots were classified as three-pointers if the ball's trajectory exceeded a specific distance; otherwise, they were classified as two-pointers. Unfortunately, this simplified approach did not produce ideal results, underscoring the need for better hardware and more robust methodologies in future iterations.

6. Conclusion and Future Work

In this study, we developed ShotSpotter, an intelligent computer vision-based system designed to detect and classify basketball scoring events in real-time. By leveraging modern object detection (YOLO) and classification models (CNNs), the system automates score recognition from video footage, offering a scalable and cost-effective solution for basketball analytics.

Despite the limitations encountered, including challenges in court mapping and shot type classification, the ShotSpotter project successfully achieved its primary goal of demonstrating a computer vision-based system for basketball score detection. The system was able to track key elements such as the ball, players, and net with an accuracy ranging between 70% and 90%. This accomplishment is significant given the constraints of computational resources and the scope of a computer vision course project. The project highlights the potential of leveraging computer vision techniques for sports analytics and provides a foundation for future development in this area.

Several improvements and extensions can be pursued to enhance the system's performance and applicability:

- **Enhanced Court Mapping:** Future efforts could focus on training larger datasets for court mapping using advanced GPU resources. Addressing challenges posed by camera movement through stabilization algorithms or multi-view approaches could further improve accuracy.
- **Improved Shot Classification:** Developing a custom deep learning model for shot type classification would enhance the system's robustness. Transfer learning with pre-trained models on similar tasks could expedite the training process and improve results.
- **Real-Time Optimization:** Optimizing the system for real-time performance by implementing model pruning and hardware acceleration will allow seamless integration into live applications.
- **Broader Dataset Diversity:** Expanding the dataset to include various camera angles, lighting conditions, and court configurations will improve the system’s scalability across different environments.
- **Advanced Analytics:** Adding features such as player performance tracking, play recognition, and statistical analysis could transform ShotSpotter into a comprehensive tool for coaches and analysts.

The ShotSpotter project demonstrates the feasibility of using computer vision for automating basketball score recognition and sets a strong foundation for future innovations in sports technology

Acknowledgment

We sincerely thank our professor, Omar Falou, for his invaluable guidance and expertise throughout our term project, "ShotSpotter." His mentorship and unwavering support were instrumental in overcoming challenges and successes of our journey throughout the semester.

We also extend our heartfelt appreciation to our dedicated team members—Ben Fredette, Ahmad Khan, Charles Kelsey, Aaron Chan, and Ryan Shiu—whose collaboration, skills, and commitment were essential to the success of this innovative project. Each member's unique contributions and unwavering pursuit of excellence were pivotal in achieving our project goals.

References

[1] B. T. Naik, M. F. Hashmi, and N. D. Bokde, “A Comprehensive Review of Computer Vision in Sports: Open Issues, Future Trends and Research Directions,” Applied Sciences, vol. 12, no. 9, p. 4429, Apr. 2022, doi: https://doi.org/10.3390/app12094429.

[2] W. Yan, X. Jiang, and Ping An Liu, “A Review of Basketball Shooting Analysis Based on Artificial Intelligence,” IEEE Access, vol. 11, pp. 87344–87365, Jan. 2023, doi: https://doi.org/10.1109/access.2023.3304631.

[3] P. Yao, “Real-Time Analysis of Basketball Sports Data Based on Deep Learning,” Complexity, vol. 2021, pp. 1–11, Apr. 2021, doi: https://doi.org/10.1155/2021/9142697.

[4] N. Wojke, A. Bewley, and D. Paulus, “Simple Online and Realtime Tracking with a Deep Association Metric,” arXiv:1703.07402 [cs], Mar. 2017, Available: https://arxiv.org/abs/1703.07402

[5] jackwu502, “NSVA/tools/download_video_by_gameid_eventid_date.py at main · jackwu502/NSVA,” GitHub, 2022. https://github.com/jackwu502/NSVA/blob/main/tools/download_video_by_gameid_eventid_date.py (accessed Nov. 30, 2024).

[6] swar, “GitHub - swar/nba_api: An API Client package to access the APIs for NBA.com,” GitHub, Oct. 29, 2024. https://github.com/swar/nba_api/tree/master (accessed Nov. 30, 2024).

[7] google, “Classification: Accuracy, recall, precision, and related metrics,” Google for Developers, 2024. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

[8] AWS, “What is Overfitting? - Overfitting - AWS,” Amazon Web Services, Inc., 2024. https://aws.amazon.com/what-is/overfitting/