Automated Ice Hockey Player Tracking

Douglas Code

University of San Diego

AAI-521: Applied Computer Vision for AI

Siamak Aram, Ph.D

December 9, 2024

Contents

Introduction **3**

Dataset **4**

Preprocessing **4**

Model Architecture **6**

YOLO 6

ByteTrack 6

Training **7**

Detection Model Hyperparameter Tuning 7

Inference and ByteTrack Hyperparameter Tuning 9

Visualization **10**

Evaluation **10**

Conclusions **11**

Appendix A: Project Links **15**

Appendix B: Hyperparameter Configuration **15**

Introduction

The ability to track ice hockey players using computer vision presents a number of opportunities. Data about player positions and movements can be used for advanced statistical analysis, and team-level positioning and movements can be used to examine the strategies employed by different teams. For live broadcasting, player tracking allows more engaging visualizations by allowing players to be highlighted in real time for commentator discussion or graphical overlays. This project aimed to build a model that could take a single game video feed as input and perform five primary tasks:

1. Use object detection to locate the position of all active players in frame.
2. Track players across frames to maintain consistent identification.
3. Classify each player as either a skater or goalie.
4. Classify each player as belonging to either the home team or away team.
5. Be performant enough to be usable in real-time applications.

Hockey presents a number of challenges for detection and tracking; players move quickly and occlusions are frequent and long-lasting. Players also largely look similar due to matching uniforms, making it more difficult to track based on appearance. For real-time usage, inference must also be fast enough to process 30 or 60 frames per second in order for the model to work in a live broadcast setting.

To address these challenges, a model was developed using YOLO and ByteTrack to perform player detection, classification, and tracking. The resulting model shows strong

performance across both all three areas, providing a capable baseline for future enhancements and visualizations.

Dataset

The McGill Hockey Player Tracking Dataset (MHPTD) is a collection of 25 gameplay clips from the National Hockey League (NHL), the highest level North American hockey league. These clips were recorded in 30 and 60 frames per second with 1280x720 resolution by a single fixed-position camera for broadcast. Each clip captures a continuous sequence of gameplay with no cuts or stoppages in play. In all, this represents 82,305 frames with 632,785 instances of a player in the frame (Yingnan Zhao, 2020).

All clips are annotated using the Computer Vision Annotation Tool (CVAT) in XML format, with tracks for each unique player detailing their location in each frame. The annotations also include a player ID, whether they are on the home or away team, and whether they are a skater or goalie. The position classes (skater, goalie) are significantly imbalanced due to there typically being five skaters and one goalie on the ice at a given time for each team. This results in ~8.4% of the player instances being goalies while the rest are skaters. Occlusions are tracked in the annotations as well, with 13.5% of player instances being occluded in some way.

Preprocessing

To get the annotations into the format expected by YOLO, the coordinates for each player-frame were parsed and normalized to a [0, 1] range relative to the frame's dimensions. Each instance was then encoded with one of four classes: home skater, home goalie, away skater, and away goalie.

Method	**Description**
Hue Adjustment	The hue of the image is modified
Saturation Adjustment	The saturation of the image is modified
Value Adjustment	The brightness of the image is modified
Translation	The image is translated horizontally or vertically
Scaling	The image is scaled to be larger
Flipping	The image is flipped left-to-right
Mosaic	Four images are combined into a single composite image
Erasing	A portion of the image is erased
Cropping	Crops a portion of the image

Table 1

Data Augmentation Techniques

Clips were split into train/validation/test datasets using a 60/20/20 split. Splitting at the clip level allowed continuity between frames to be preserved for the tracking part of the model, and provided a better test of model efficacy by ensuring that no frames from the validation/test clips had been seen during training. To improve the robustness of the model, the training data frames were then augmented using a number of techniques. The augmentation techniques that were used are described in Table 1 (Ultralytics, 2024).

All images were resized to a resolution of 640x360, preserving the original aspect ratio. This was done to improve training and inference speed while maintaining a high enough resolution for the model to be effective.

Model Architecture

The model is composed of a two-step pipeline: YOLO for object detection and classification and ByteTrack for object tracking.

YOLO

You Only Look Once (YOLO) is an object-detection technique that performs detection in a single pass rather than using two stages, greatly improving inference speeds. This is done by dividing the input image into a grid and predicting bounding boxes and classifications for items centered in each grid cell. These bounding boxes are then combined using intersection over union (IOU) thresholds and non-max suppression to get a final predicted bounding for an object (Redmon et al., 2016). YOLO was chosen for this application due to its ability to handle real-time inference while maintaining high bounding-box and classification accuracy.

ByteTrack

ByteTrack is a multi-object tracking approach that differs from competing approaches in its use of low-confidence detections. ByteTrack takes an initial pass to use high-confidence detections to establish motion information along with appearance similarity baselines. It then incorporates low-confidence detections by using the established motion and appearance information to identify which low-confidence detections are likely to be valid (Zhang et al., 2022). Incorporating these detections instead of discarding them makes ByteTrack especially effective at handling occlusions compared to other techniques, which is crucial due to the frequent occlusions in ice hockey recordings.

Training

During training, the nano, small, medium, and large versions of the Ultralytics YOLO v11 implementation were compared (Jocher et al., 2023). The medium model offered the best balance of model complexity and performance, with the nano and small models underperforming on detection and classification for this task. The large model was significantly more resource-intensive to train and use than the medium model, and also was not able to achieve better predictions. To address the class imbalance, each class was weighted using its inverse frequency for the classification loss function.

The trained models showed a strong tendency towards overfitting, with the training loss decreasing steadily while there was minimal improvement in validation loss even in very early epochs. This was addressed by introducing weight decay and dropout, as well as a 3-epoch patience interval to stop training when the performance on the validation dataset stopped improving. Decreasing the batch size was also found to be effective in decreasing the tendency to overfit, with a batch size of 8 being used in the final model.

The learning curves, shown in Figure 1, for the final model show much less indication of overfitting, with similar improvements in both the training and validation datasets across the entire training period. Improvement across all metrics largely plateaus around epoch 20, and training was stopped by the early stopping mechanism at epoch 25.

Detection Model Hyperparameter Tuning

Hyperparameter tuning for the YOLO detection model was performed using Optuna, which uses a Tree-Structured Parzen Estimator (TPE) tuner for Bayesian optimization of the hyperparameters (Akiba et al., 2019). Using Bayesian optimization was

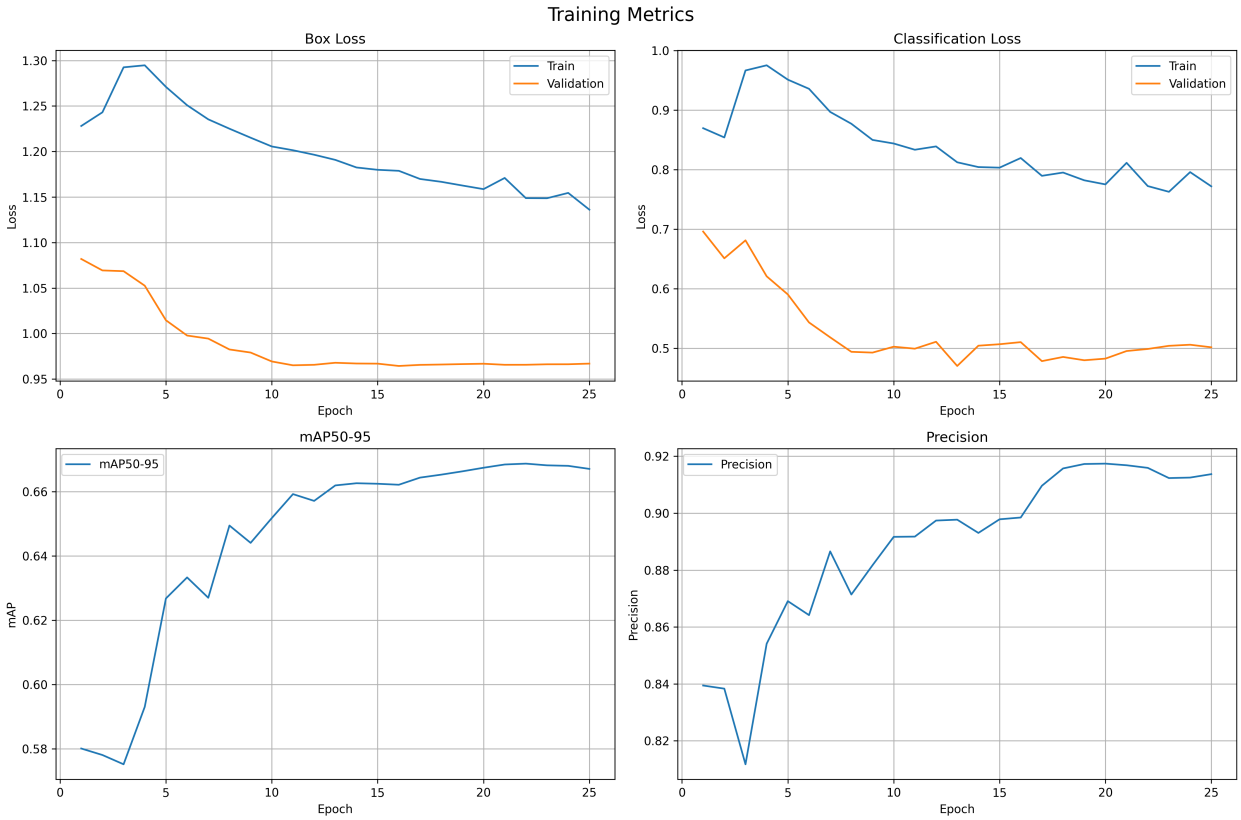

Figure 1

Detection model learning curves.

particularly valuable for this project due to the relatively long amount of time needed for each training run. This allowed the tuner to converge on a high-performing set of hyperparameters much more quickly than would have been likely with a random or grid search.

The metric used for tuning was maximization of the F1 score. This was chosen to balance precision (making sure detected players are actually players) and recall (making sure actual players are detected). The full hyperparameter ranges used and final values can be found in Appendix B. One notable result was the relatively high dropout value (0.7), validating the observation that overfitting was a significant issue.

Inference and ByteTrack Hyperparameter Tuning

Key parameters for YOLO inference and ByteTrack were also tuned using Optuna. This tuning was done as a separate process once the final hyperparameters for the detection model fine-tuning were found. Running two separate hyperparameter tunings allowed the hyperparameter space to be greatly reduced for the time and resource-intensive process of fine-tuning. The YOLO detection confidence threshold and ByteTrack parameters were then able to be rapidly tuned using the pretrained detection model. The tuning metric for this secondary hyperparameter tuning was the harmonic mean of the precision, recall, multi-object tracking accuracy (MOTA), and multi-object tracking precision (MOTP):

$$H = \frac{4}{\frac{1}{P} + \frac{1}{R} + \frac{1}{MOTA} + \frac{1}{MOTP}}$$

The use of MOTA and MOTP is discussed further in the Evaluation section. Using the harmonic mean of these metrics allowed the tuning process to target a balance of precision and recall across both detection and tracking.

The full hyperparameter ranges used and final values can be found in Appendix B. However, one notable result is the extremely low value for the ‘track_low_threshold‘ parameter, which is used to determine which detections to attempt to associate with the track after the initial high-confidence pass is completed. The very low value here helped improve track preservation by allowing very-low confidence detections caused by occlusions to be considered for inclusion in the track. Since hockey involves frequent physical contact, players passing behind each other, and players passing behind the boards and stanchions, this ended up being an effective way to handle these scenarios.

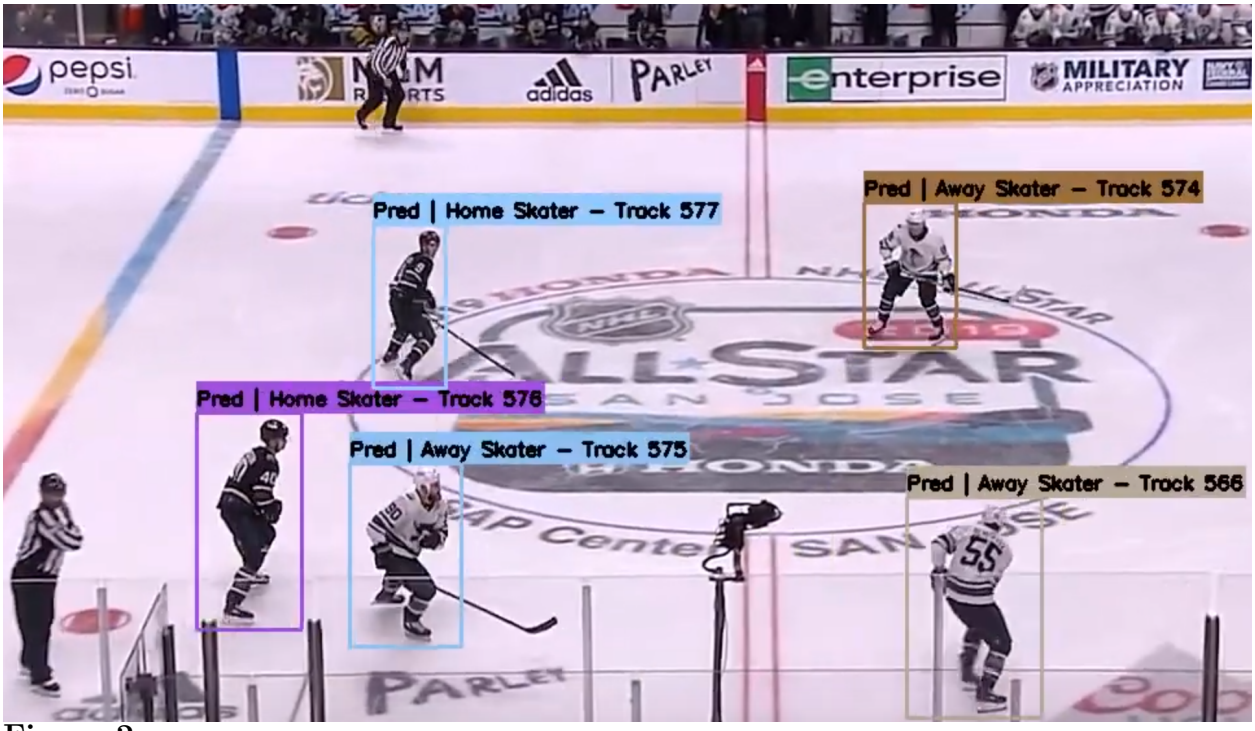

Figure 2

Example frame from visualized output clip.

Visualization

Predictions were then visualized on each frame using bounding boxes with classification labels and track IDs. Each frame in a clip was then combined to create a final visualization video that had the predictions overlaid on top of the original clip. Figure 2 shows a frame from one of the generated visualization clips, with each player annotated with a bounding box, their classification, and their track ID. The link to a full clip visualization is provided in Appendix A.

Evaluation

Six primary metrics were used to evaluate the model on the test dataset, as shown in Table 2.

Overall, the model shows very strong performance in precision (0.985), recall (0.939), and MOTA (0.923), being able to correctly identify and objects and tracks quite consistently. The MOTP is not as high at 0.823, but still indicates a large amount of overlap with the ground truth bounding boxes. Examining the visualizations, the accuracy of the bounding boxes appears to be close enough that the practical implications would be minimal.

The test dataset contains 430 tracks, meaning that the average track experiences ~1.16 fragmentations and ~0.5 track switches over the course of each clip. Given the frequency of players crossing paths, the relatively low number of track switches shows that the model does a fairly good job in maintaining tracks for nearby players. The relatively high number of fragmentations is likely due to the annotations preserving tracks for players when they're out-of-frame, whereas this model doesn't have any mechanism for identifying players themselves to preserve their tracks once the player is no longer visible.

In terms of real-time processing for broadcast, the model was benchmarked against the test dataset and was able to process 68 frames-per-second on a mid-range consumer GPU (Nvidia RTX 3060ti). This suggests that the 30-60 fps needed to maintain real-time processing during a live broadcast would be easily achievable with GPU hardware appropriate for a production setting.

Conclusions

Using YOLO and ByteTrack, a highly effective model for ice hockey player tracking and classification was able to be created. The minimal computational resource requirements for the model make it viable in both analytical and real-time scenarios.

The most valuable way to improve the model's usefulness in a production setting would be to add player identification. This would make the model much more valuable for both analytical and broadcast settings, as the tracking would be able to continue to follow a player even if they leave the frame and return or are occluded for long enough to lose their track. However, player identification is a difficult problem in this setting due to everyone of each team wearing the same uniform, as well as helmets and padding that obscure the face and body. One potential approach is to use jersey number recognition to identify players. This approach has shown some success, with researchers using the CRAFT text detection method to achieve ~87% accuracy in detecting jersey numbers (Chen & Poullis, 2023). Jersey number recognition does present its own issues, primarily in the fact that it requires the players' back to be visible, making it impossible to make an identification in some situations regardless of model quality.

Metric	**Description**	**Value**
Precision	The proportion of detected players that matched with actual players rather than being false detections.	0.985
Recall	The proportion of actual players in the clip that were detected by the model.	0.939
Multi-Object Tracking Accuracy (MOTA)	The overall tracking accuracy, accounting for missed detections, false detections, and identity switches. Ranges from -inf to 1.	0.923
Multi-Object Tracking Precision (MOTP)	The average overlap between the predicted bounding boxes and their matching ground truth boxes. Ranges from 0 to 1.	0.823
Track Switches	The number of times the tracker incorrectly transferred the track from one player to another.	214
Fragmentations	The number of times a player track was incorrectly split into multiple tracks by the tracker.	500

Table 2

Detection and tracking evaluation metrics.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2623–2631. https://doi.org/10.1145/3292500.3330701

Chen, Q., & Poullis, C. (2023). Tracking and identification of ice hockey players. *Computer Vision Systems: 14th International Conference, ICVS 2023, Vienna, Austria, September 27–29, 2023, Proceedings*, 3–16. https://doi.org/10.1007/978-3-031-44137-0_1

Jocher, G., Qiu, J., & Chaurasia, A. (2023, January). *Ultralytics YOLO* (Version 8.0.0). https://github.com/ultralytics/ultralytics

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. https://arxiv.org/abs/1506.02640

Ultralytics. (2024). *Train - Ultralytics YOLO documentation* [Accessed: 2024-12-09]. Ultralytics. https://docs.ultralytics.com/modes/train/

Yingnan Zhao, K. C., Zihui Li. (2020). A method for tracking hockey players by exploiting multiple detections and omni-scale appearance features. *Project Report*.

Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., & Wang, X. (2022). Bytetrack: Multi-object tracking by associating every detection box. https://arxiv.org/abs/2110.06864

Appendix A: Project Links

Github Repository: https://github.com/dcode15/ms-aai-521-final-project

Example Visualization: https://www.youtube.com/watch?v=Enrnuo74tcs

Appendix B: Hyperparameter Configuration

Hyperparameter	**Tuning Range**	**Final Value**
Learning Rate	[0.00001, 0.01]	0.00975
Weight Decay	[0.00001, 0.001]	0.00024
Dropout	[0, 0.75]	0.7
Detection Confidence Threshold	[0.1, 0.9]	0.72
IOU Threshold	[0.1, 0.9]	0.89
ByteTrack Track Buffer	[1, 120]	35
ByteTrack Match Threshold	[0.1, 0.9]	0.8
ByteTrack Track Low Threshold	[0.01, 0.5]	0.023
ByteTrack Track High Threshold	Low Threshold + [0.01, 0.5]	0.356
ByteTrack New Track Threshold	[0.1, 0.9]	0.025

Table 3

Hyperparameter tuning ranges and final values.