EECS 442 Live Hockey Puck Detection

University of Michigan

Donovan Below, Meitang Li, Connor McKinley, Julie Saia

dbelow@umich.edu, meitang@umich.edu, cmmck@umich.edu, saiap@umich.edu

1. Introduction

For our EECS 442 final project, we decided to improve the live hockey broadcast experience by using object detection to track the puck's motion, and superimposing a trail and bounding box on top of the existing broadcast.

1.1 Motivation

One of the main difficulties when watching hockey, especially for newcomers, is keeping track of the puck's position. Hockey is a very fast paced sport, and the puck is very small relative to the camera's field of view. Coupled with the average viewing experience of a far-away TV or a small mobile device screen, it can sometimes be difficult to make out the details of the game.

With recent advancements in computer vision, it has become more feasible to make object detectors that can perform better than human eyes. Even with limited computing resources, training a neural network using deep learning can produce very accurate results for approximating the location of an object given a frame of the broadcast. Furthermore, by using more computing resources or a smaller model, the predictions can be done fast enough to support a standard 30 frames per second broadcast in real time. Deep learning models are still not fully accurate, especially when the puck goes behind players or their sticks. To alleviate this problem, we introduce a method of extrapolating from previous known points to preserve a smooth tracking experience.

1.3 Impact

Making the puck easier to see in a hockey broadcast greatly lowers the barrier for entry to enjoy watching a game. This has implications for growing market share for large hockey league broadcasters like the NHL. Especially in America, hockey falls behind in popularity compared to other sports like football and basketball [1]. Adding this technology to existing broadcasts can help grow the sport and attract new fans.

1.4 Feasibility

Object detection is one of the forefront applications for computer vision research. By leveraging convolutional neural networks, after training on manually labeled images we can reliably output accurate labels and bounding boxes for the puck. With recent technological advances, even a consumer-grade laptop GPU can both train and predict in a reasonable amount of time.

1.5 Related Work

Ball and puck tracking is not a novel idea. The NHL has introduced a hardware solution for tracking, using a custom made puck that uses infrared to communicate with cameras in the venue. Because of the difficulties with optically tracking a hockey puck, especially its small size and easily camouflaged color, using computer vision for this task is rare in the sport. In other sports, like soccer, the ball is much bigger and has more recognizable features for a camera to detect. In baseball and golf, the ball is usually up against a contrasting background like the sky or field. For this reason, optical ball detection has become a commonly tackled problem in the literature. Many groups have applied convolutional neural networks for this problem, as we have for hockey. For example, in 2020 researchers from the University of Kansas and Ryerson University evaluated models like YOLO v3 and SSD for golf ball tracking [2].

1.6 Method Summary

We first trained the Ultralytics YOLOv8 CNN (convolutional neural network) model on hockey game footage gathered from Youtube, using 200 manually labeled images auto oriented and scaled to 1080p. Using the CNN detector as a baseline, we extrapolate any missing data points using the puck's previous known positions. This is all done in real time for a seamless live broadcast experience.

2. Approach

Our approach uses a custom-trained neural network to have a baseline of the puck's position. We then filtered the results to get the most accurate path of motion, assuming that the path is continuous. We then filled in any gaps left by the detector by extrapolating the puck's motion from earlier data points.

2.1 Gathering Data

To train the neural network, we needed a set of images from hockey broadcasts, as well as a bounding box on the puck in the image. We first gathered videos of hockey broadcasts from ESPN and TSN hosted on Youtube, downloaded with the tool youtube-dl. We split the videos into individual frames with ffmpeg, giving us around 200 images of hockey broadcasts. We took care in including borderline cases to improve the robustness of our model. Some examples include blurred frames where the puck is moving at high speeds, and frames where a player is handling the puck. Puck handling was an especially important case, as it is very common in the footage. The puck is also not very distinct from sticks of the same color, so these cases are difficult for a neural network to detect.

2.1 Labeling Images

We then set out to add bounding box annotations to each of the 200 images. For this we used the LabelImg tool, which automatically generates text files defining the starting position, width and height of a user-drawn bounding box. Going through each frame manually, we generated bounding box annotations for our neural network to train on. The dataset was then divided into training and validation sets, with a 90:10 split.

2.2 YOLOv8 Convolutional Neural Network

The cornerstone of our puck tracking implementation is the You Only Look Once (YOLO) convolutional neural network [3]. YOLOv8 is a state of the art model that focuses on maintaining accuracy with a small size. This small size was especially important for us, as we wanted to apply this for live broadcasts. With enough computational power, YOLOv8 is one of the only CNNs capable of detecting objects at a rate comparable to a broadcast's usual frame rate of 30 frames per second.

YOLOv8 is based off of a CSPDarknet53 backbone, with many additional layers added as described in Figure 1. BCE (binary cross entropy) is used for classification, and a combination of DFL (distributional focal loss) and CIoU (complete intersection of union) loss is used for regression of the bounding box coordinates [4]. SGD was used for our optimization.

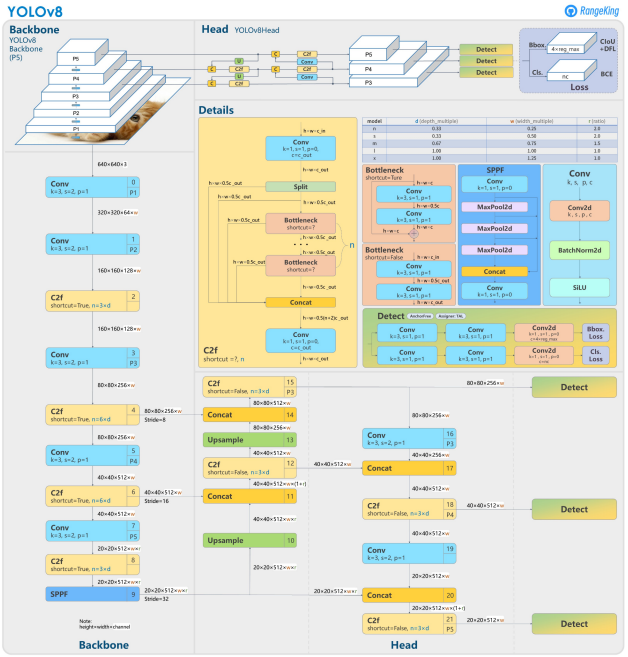

Figure 1: Architecture of the YOLOv8 CNN [4]

Training was somewhat difficult, as we were limited to using consumer-grade GPUs. Because the puck is so small, we were forced to use the full 1920x1080 images in training in an attempt to limit noise, which limited our batch size to 2. YOLOv8 includes 5 different models of increasing size, ranging from the 3.2 million parameter nano model to the 68.2 million parameter extra large model [3]. With our computing resource constraints, we were able to use the large 43.7 million parameter model. We used 100 epochs for our training duration.

2.3 Motion Prediction and Extrapolation

YOLOv8 provides a tracker right out of the box, but it fell short in some areas that led us to create our own on top of the raw detection data. For one, it would sometimes detect more than one puck in an image, which is obviously wrong. It also did not care for motion continuity, allowing the tracked puck to rapidly fly around the screen if some other portion of the image was erroneously detected. Lastly, the built in tracker only featured a box and detection id, not the trail that we wanted to implement.

To make the detection results into a more intuitive tracker, we first made a custom script that took each generated frame, overlaid a box around the most confident detection, and placed the new frame in a video buffer. We then added a trail that slowly decayed over time, which made the results easier to follow for the viewer.

We then tackled the problem of motion continuity. If the detected puck in one frame is more than 100 pixels away from the previous position, then we instead use an

extrapolation approach using the last 2 detected positions. Fortunately, a puck moving on ice is roughly frictionless, so it will usually be moving at a constant velocity with no interference. We shift the last detection box by the difference between it and the second last detection box in order to extrapolate the new position. In order to make sure this approach is only supplementary to the actual CNN detections, we keep the detection box still if there is a streak of 10 or more motion predicted frames. We also keep it still if it wanders more than 25 pixels away from the last CNN detected position. Finally, to account for scene transitions where there really is no puck, any detection with a confidence over 50% will override the motion prediction algorithm. This allows for the algorithm to start again after a long stretch of frames no detections.

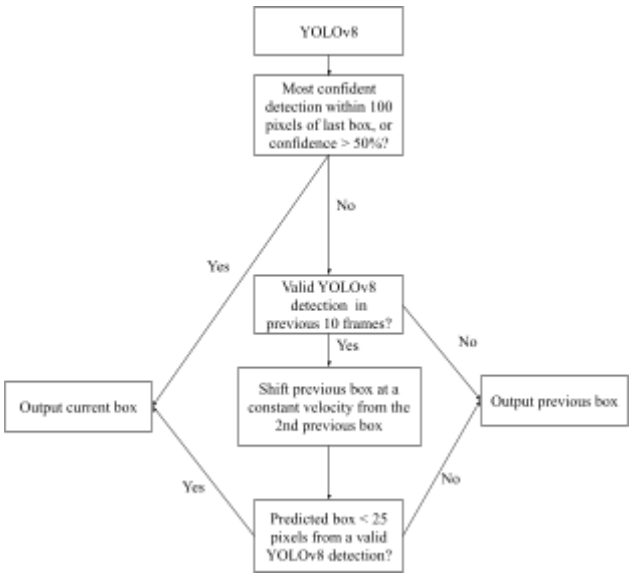

Figure 2: Complete algorithm flow chart

3. Experiments

Our experimental metrics are mostly qualitative, with some quantitative analysis available for the detection rate of the CNN in training, as well as the portion of motion predictions needed when testing. Performance is also easily analyzed with quantitative metrics.

3.1 Data

We gathered our testing data in the same manner that we gathered the training data. Instead of training on still images, we tested our implementation on full hockey videos, usually highlights to focus our attention on scenes that most viewers will care about. Testing on full hockey videos also allowed us to understand what our system would look like when deployed to a live game when there isn't necessarily a puck in each frame, for example a camera cut to the star player on the bench.

3.2 Metrics

To analyze the detection rate of our CNN model, we turn to the mAP (mean Average Precision) metric. We only use one category (the puck), so this is equivalent to just average precision. Our criteria for a true positive is 50% IoU (intersection over union): if the portion of the generated bounding box overlapping the ground truth box is more than half of the total area of both boxes, then the detection is counted as correct. Average precision is then calculated as the area under the precision-recall curve, with points plotted over 10 different confidence intervals from 50% to 95%.

As a holistic metric of tracking performance, we simply recorded the amount of frames with a reasonably accurate prediction, whether it was derived from the CNN's detection or motion prediction.

Another key metric we wanted to record was performance. Considering our goal of applying this to a real time live broadcast, we need to accurately measure how fast we can detect the puck. This was done using simple Python timing functions, averaging the amount of time it took to track an entire video over the total number of frames.

Finally, the intuitive nature of puck tracking allows us to clearly judge the smoothness and accuracy of our tracker simply by watching the puck with our eyes. In many cases, the action is too hectic to accurately judge (hence the need for our computer vision solution), but most of the time it is easy to catch any egregious errors. Qualitative human judgment is also very useful for analyzing the continuity of our motion prediction solution, as any large jumps in the trail will be easily noticeable.

3.3 Qualitative Results

Visually, testing on both our large and medium models looks passable. With a dataset of only 200 images sourced from YouTube videos from one panning TV camera, the results are surprisingly good. Implementation of predictive motion allows for several frames to be skipped while still deploying a smooth graphic that accurately depicts the puck's true motion. Even when the puck is being handled directly by a player, a generally hard case to detect, our system correctly detects the puck and displays its trail.

Figure 3: Example of motion continuity with large model, red detections are CNN and blue are motion prediction (a full 2 minute test video of our large model can be found at https://youtu.be/QhmDKYCIkGs)

3.4 Quantitative Results

After 100 epochs of training, our large model achieved a mAP of 74.8%, highlighting the robustness of our trained CNN model.

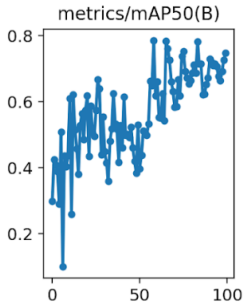

Figure 4: Plot of mAP over training epochs

In our 6812 frame benchmark video, our large model made reasonable predictions for 4553 frames. The medium model only made 4432 predictions, showing the marginal improvement of the deeper network.

The biggest difference between the model sizes was performance. The small model was able to make a prediction at about 32.1 FPS. The medium model achieved 13.0 FPS, and the large model was the slowest at 6.8 FPS. Given our goal of matching the speed of a live broadcast, this shows that at the consumer level only the small YOLOv8 model is feasible. With more resources, it is reasonable to expect the larger models to keep up with a 30 FPS video stream.

4. Implementation

Our motion prediction algorithm is completely custom made, and our data set was gathered and labeled ourselves by hand. We rely upon the YOLOv8 CNN model architecture as the basis for our neural network detection. We also used the built-in training tools in order to train on our custom data using the YOLOv8 architecture. While YOLOv8 has a built-in tracker, we made a custom implementation instead to better suit our needs. YOLOv8 was only used to generate the detected bounding box data. Details about downloading the hockey game footage and manipulating it to extract frames were handled by youtube-dl and ffmpeg. LabelImg was used to streamline the bounding box labeling process as we annotated the dataset. For recording metrics, we used YOLOv8's training tools to measure mAP and detection speed. We measured the portion of predicted frames and the speed of our motion prediction algorithm ourselves.

5. Future Improvements

Given that our system was geared towards being a proof of concept rather than a true deployment of computer vision to the consumer market, there are areas where our project can be improved. First, a dataset of 200 images is not sufficient enough to train a robust system ready for consumer deployment. More data should be collected to ensure that the system would work in all scenarios it might be deployed in. Secondly, with our current hardware we are only able to barely achieve 30 FPS on our test data. With more powerful and specialized hardware, it is very feasible to apply our solution to broadcasts at higher frame rates. Finally, our motion prediction and decaying puck trail do not account for camera rotation and translation, which is widely used and nearly unavoidable when processing broadcast data. This camera panning occasionally causes the puck trail to be inaccurate as it is mapping to pixels in the image plane, not to the actual location on the rink. There are established libraries such as Norfair [5] that can help to display the path relative to the rink, which would help eliminate the inaccuracies caused by camera panning.

References

[1] J. Norman. "Football Still Americans' Favorite Sport to Watch." *Gallup*, 2018.

[2] T. Zhang et al. Efficient Golf Ball Detection and Tracking Based on Convolutional Neural Networks and Kalman Filter. 2020.

[3] Ultralytics. "Ultralytics YOLOv8 Docs." https://docs.ultralytics.com/ (accessed Apr. 24, 2023)

[4] R. King. "Brief Summary of YOLOv8 Model Structure." https://github.com/ultralytics/ultralytics/issues/189 (accessed Apr. 24, 2023)

[5] J. Alori et al. "tryolabs/norfair: v2.2.0." *Zenodo*, 2023. https://doi.org/10.5281/zenodo.7504727