# A Novel Approach to Object Tracking using Convolutional Neural-Network Trajectory Prediction

Abhiram Tirumala
Georgia Institute of Technology
abhiram.tirumala@gatech.edu

Shaashwat Sharma
Georgia Institute of Technology
shaashwatsharma@gatech.edu

Pranav Manjunath
Georgia Institute of Technology
pmanjunath9@gatech.edu

## Abstract

*The state of art object tracking algorithm, You Only Look Once (YOLO), is computation-heavy and requires high-end computers to run at a high frame rate. However, a faster and lightweight approach would allow object tracking to be done at a high frame rate using mobile devices with significantly less computing power. We propose a novel architecture which uses intelligent object trajectory prediction using a convolutional neural network (CNN) in tandem with calls to YOLO once every ten frames to have the accuracy of YOLO with the amortized speed of a standard CNN. In this way we decrease the number of frames in which YOLO is called. Our proposed model runs much faster than YOLO v4 while still maintaining competitive results for object tracking accuracy.*

## 1. Introduction

The current best method for live object-tracking in a video is with the You Only Look Once (YOLO) algorithm. While it has a high accuracy rate, its downside is that it is computationally expensive. As a result, it has a maximum speed of 65 Frames per Second (FPS), a speed that is only achievable on high-end CPUs or GPUs [1]. Because of this limitation, it isn't viable to use YOLO for tracking objects in high frame-rate live video. In this project, we have attempted to resolve this issue by creating a novel algorithm that combines YOLO and intelligent trajectory-prediction methods to achieve smooth and accurate object-tracking in high frame-rate videos. Our trajectory-prediction model has been trained solely on videos of people performing various tasks, where in each video a singular person is the sole object being tracked. However, this algorithm could be generalized to achieve similar trajectory-tracking success with other objects. If this project is successful, it could improve object-tracking using low-computing power smartphones. This would make it possible to create seamless augmented reality experiences, as augmented reality applications often require object-tracking, but are usually run on low-computing power devices such as smartphones.

### 1.1. Dataset

Our dataset consists of 100 240 Frames Per Second (FPS) videos of varying length in real-world scenarios of various objects moving on the screen. For the purposes of our implementation, we limited the videos to those only tracking humans, so that our CNN architecture would learn the typical movement behavior for humans.

### 1.2. Approach

To solve this problem, we started by attempting to reduce the number of frames in the live video that YOLO would be used on, hoping to save computing power. However, we needed to find a way to ensure that the frames in between each pair of calls of YOLO were still tracking the object accurately. We looked at a paper that accomplishes this task by utilizing a Convolutional Neural Network (CNN) [2]. After doing so, we believed that we could successfully use such a CNN in conjunction with YOLO to improve the speed of object-tracking. However, unlike our intended algorithm, the model delineated in this paper takes in 8 previous locations and predicts the next 12 all at once. This was unsuitable for us, as our goal was to implement a trajectory-prediction model that takes in the previous ten locations of the object to predict where its center will be in the next frame, so we had to modify it to fit our goals. We also decided to use PyTorch without any starting code or models to implement our new architecture from scratch.

When designing our algorithm architecture, we trained our model on trajectories of people moving in real-world environments in several videos. Our input data has four features: the X-coordinate of the center of the bounding box, the Y-coordinate of the center of the bounding box, the width of the bounding box, and the height of the bounding box. This data is passed in in time-series format, with each data point representing the location of the object in one frame. We pass in the previous ten data points (X, Y, width, height) where the object has been, with the model outputting a prediction for where the object will be at the

next timestamp. Then, we take this predicted point and append it to the previously passed-in list of input data. We then use the ten most recent frames of the updated list as the input for the next iteration and once again call the modified CNN (See Fig. 1 for CNN architecture), producing the prediction for the location of the object at the next timestamp. Note that our Convolutional Layers have learned parameters and that we utilized standard Mean-Squared Error as our loss function. We used an AdamW optimizer. As for our optimization hyperparameters, we used a constant learning rate, and arrived at it by testing multiple values and performing binary search until we arrived at the value that provided the optimal learning rate: 0.0001. However, in the future, we will investigate adding a dynamic learning rate using a learning rate schedule which would reduce the learning rate as the epochs advance, making it possible to save time until convergence while preventing overshooting. We complete such successive iterations until ten predictions of locations have been made.

At this point in the process, we anticipated a potential issue. We realized that if we continued to use

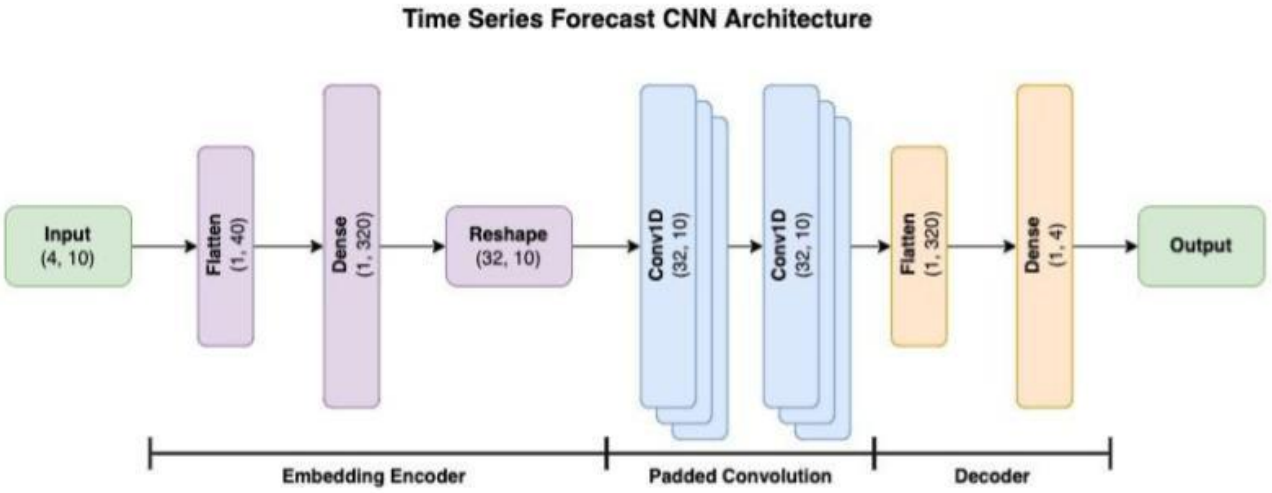


Figure 1: Our CNN Architecture

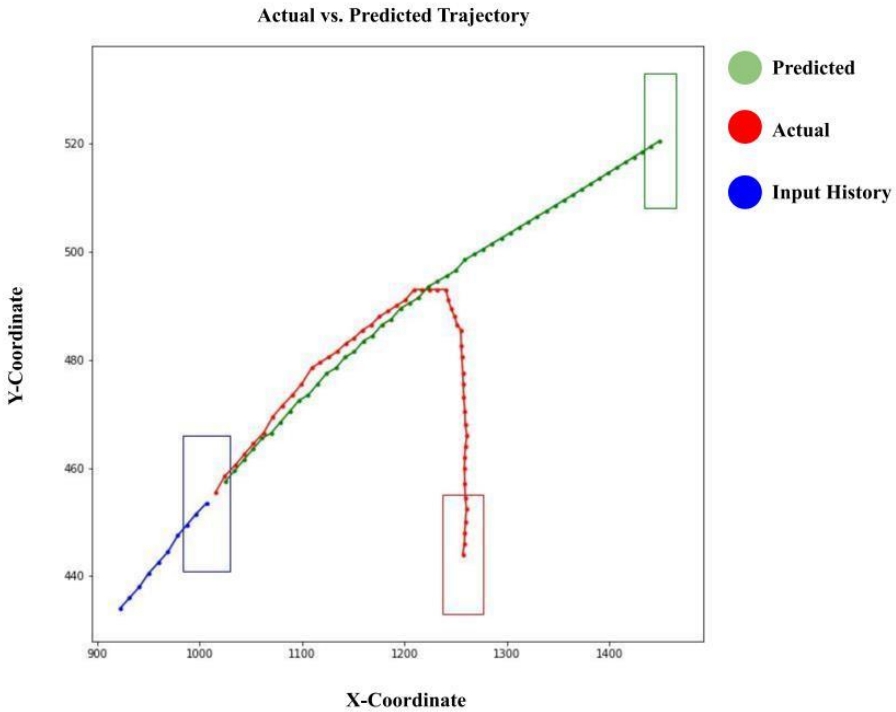


Figure 2: Actual vs. Predicted Trajectory of Single Human on Basketball Court

trajectory-predictions, the errors of previous incorrectly predicted data points would be compounded and lead to extremely inaccurate bounding boxes. The errors would be especially pronounced if the object being tracked started to move in a different direction during the timespan of these ten frames (See Fig. 2 for an example of this situation).

We created a novel approach to solve this problem. To combat this, we start by calling YOLO on the eleventh frame to find the "true" location of the object after acquiring ten frames of predictions. Note that we pre-trained YOLO to track a person moving in various scenarios with learned parameters. Initially, we were thinking that every time we called YOLO to find the true location of the object (once every ten frames), we would add this point to our running history of ten points by having it replace the least recently predicted point, resulting in a set of ten points in which the first nine were predicted using our trajectory-prediction model and the tenth was the YOLO point. Then, our algorithm would predict the next point based on these ten points. However, we realized that up to nine of these could easily be wildly inaccurate because they were predictions of our trajectory-prediction model and not the actual locations. Thus, we needed to come up with a way to avoid compounding these potential errors and reset our running history.

Our solution was to create a second-degree parametric polynomial-fit between the eleventh frame and nine of the frames that were identified before we predicted ten more frames. Afterwards, our algorithm replaces the ten predicted locations with locations that are on the polynomial-fit line at each of the ten timestamps. Note that this polynomial-fit function does not use any learned parameters, allowing it to be very time-efficient. We do this because, as asserted above, the predicted values may have errors and now that we have a truth value from YOLO we don't want to continue using these potentially inaccurate data points as inputs to our model. Finally, we call the trajectory-prediction CNN again and pass in the ten data points that were on the polynomial-fit curve, and the process repeats. This enables our model to "reset" every time YOLO is called (See Fig. 3 for a visualization of this process).

While this algorithm ended up being the most accurate and performant method, we did attempt several other approaches, each of which came with their own set of issues. Before we decided to use a CNN for the trajectory-prediction, we came across a different paper that described predicting human trajectories in crowded spaces using Long short-term memory Recurrent Neural Networks (LSTM RNN) [3]. We thought that tracking people performing tasks among other people would be a perfect application of this algorithm.

However, we ran into a multitude of problems when attempting to implement it. One major issue was that in the paper, only the center of each human was being tracked as an independent object, while for our algorithm we wanted to create a bounding box to capture the size of the human on-screen as well. Additionally, this paper's model was meant to track multiple people's trajectories at once, while

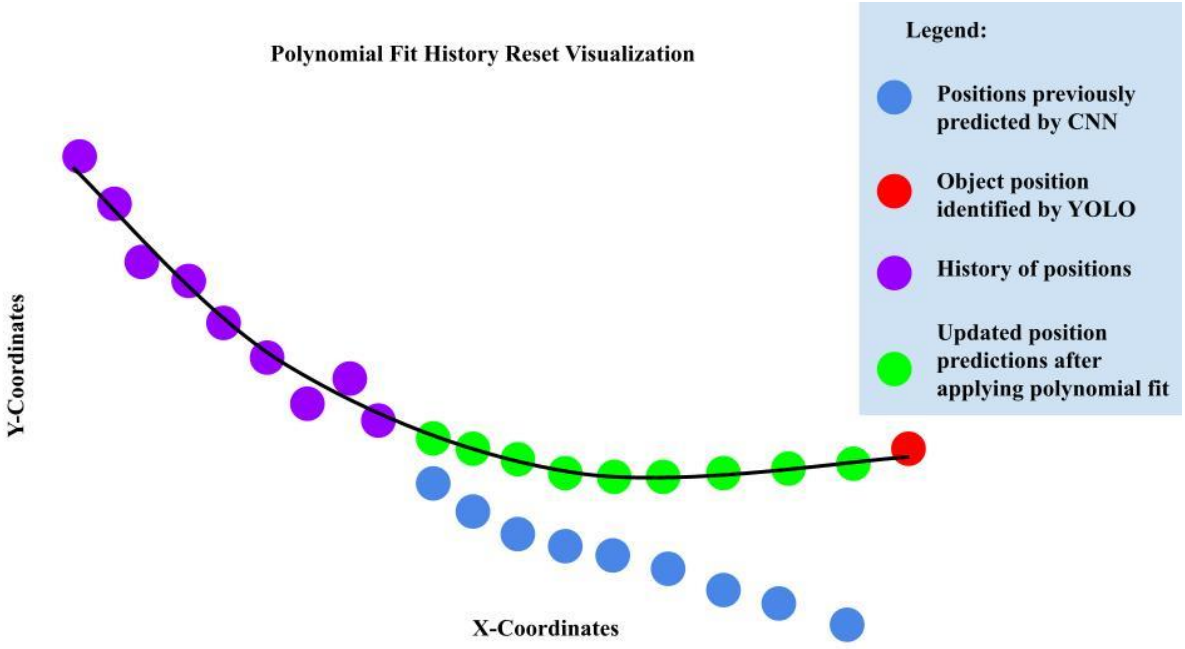


Figure 3: Demonstration of how issue of compounding errors of previous incorrectly-predicted points is prevented. A second-degree polynomial-fit line is drawn between the history of positions and the object position identified by YOLO. Then, the predictions are edited to be on this polynomial-fit curve before being passed as input into the next iteration.


Figure 4: Bounding-Box Generated by our person-tracking algorithm

we were only concerned with tracking one individual's trajectory at a time. We tried modifying the algorithm described in this paper to fit within the requirements of our problem space but discovered that our LSTM was quite inaccurate and slower than expected. We presume that this is because an LSTM tends to learn long term time series patterns better, while we are concerned with short term time series prediction.

We also tried to use a transformer implementation for multistep time-series forecasting. However, we found that while a transformer is built primarily to understand long-term trends, it does not handle small windows of time very well. This is necessary for our live object-tracking with only a 10-frame history. We also discovered that it was significantly slower than the CNN implementation. The average inference time for our CNN implementation was 0.001043 seconds. The average inference time for the transformer implementation was 0.002280 seconds, making our CNN implementation more than twice as fast.

### 1.3. Experiments and Results

In this project, we were trying to optimize the speed of object-tracking while keeping similar accuracy levels as using the pure YOLO v4 architecture (the current most widely used approach). One way we measured success was by comparing the speed of our new algorithm with that of YOLO. Our results for this were very promising, as our architecture was able to run at an average speed of 8.08 frames per second, while YOLO could only run at 0.82 frames per second on our CPU-based system (which emulates a mobile device). However, this improvement in speed would be meaningless without maintaining a high level of accuracy. To measure the accuracy of the object tracking, we calculated the Euclidean distance between the person's actual bounding box coordinates over all time steps averaged together. Then, we repeated the process with YOLO in place of our algorithm. While we undoubtedly expected YOLO to have a lesser average error, we were pleasantly surprised to discover that the difference was not nearly as significant as we thought it would be. The average per-frame error for YOLO turned out to be 452.187, while our average per-frame error was 583.428 for the hand-labeled training data. To confirm these results, we decided to perform a qualitative investigation by plotting our bounding boxes as compared to the true bounding boxes of the individuals moving around in the environment (one individual per video). We rendered each video in 240 fps, which standard YOLO v4 was not able to run at on our systems. We found that our predictions were usually very close to the actual locations of the individuals being tracked (See Fig. 4 for a screenshot from the rendered video with our tracking algorithm applied). They would never be significantly off, usually being just a few pixels away from the true bounding box if not completely accurate to begin with. In addition, unlike YOLO, our algorithm always had a bounding-box prediction for every frame, even if the individual being tracked suddenly disappeared behind an

object or another individual for some short period of time during the video.

Additionally, we found that our model did not overfit. Our model converged at a Mean-Squared Error loss of 1.5441. We removed our dropout layer for this reason. We suspect the reason for our model not overfitting is because of the diversity of our training data; it had many different videos with various lighting situations and angles which allowed our model to generalize and not overfit to the training data.

Based on these results, we call our experiment a success. We were able to achieve higher speeds than the widely adopted State-of-the-Art (SOTA) YOLO architecture without compromising much on the accuracy. Additionally, we achieved our goal of being able to process live video at 240 fps while maintaining a consistently accurate bounding-box prediction.

### 1.4. Conclusion and Next Steps

Future implementation possibilities include building a proper module to process real-time video footage while simultaneously tying together YOLO and our model. Additionally, our CNN model architecture fits differently depending on the type of object one is attempting to track. So building a more generalized model trained to track almost any object would be ideal (such as all of the different object classes listed in the COCO dataset).

We would also like to experiment with different types of object-detection architectures besides YOLO v4, such as SSD, Faster RCNN, and RetinaNet. Additionally, we'd like to compare our results with other SOTA object-tracking algorithms such as ROLO, GOTURN, Deep SORT, TrackR-CNN, and Tracktor++ (Tracktor++ implementation is very similar to our proposed method of object-tracking).

Finally, our current implementation doesn't handle multiple object-tracking (MOT) but can be built to handle such cases, so extending our framework to handle MOT is a future endeavor that we intend to undertake.

### 1.5. References


[1] Anka, A. (2020, July 16). Yolo V4: Optimal Speed & Accuracy for Object Detection. Medium. Retrieved December 10, 2021, from https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50.

[2] Nikhil, N., & Morris, B.T. (2018). Convolutional Neural Network for Trajectory Prediction. ECCV Workshops.

[3] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 961-971, doi: 10.1109/CVPR.2016.110.