ByteTrack: Multi-Object Tracking By Associating Every Detection Box

Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan,
Ping Luo, Wenyu Liu

Abstract:

In this project, we worked towards a goal rather than a problem. We aimed to make an improvement on ByteTrack that could make a significant difference on processing time. We researched the most time-consuming stages of this project, which the project is already very successful in terms of accuracy, in order to make it work much faster. We found that the matching algorithms took the most time to run. We made improvements on these and achieved a performance increase of almost 100%.

Keywords: byte, MOT, tracking.

I. INTRODUCTION

While the existing ByteTrack demonstrably excels in object tracking accuracy, we recognized an opportunity to optimize its processing speed, a crucial factor in real-time applications. Despite its success, existing matching algorithms within ByteTrack constituted a significant processing bottleneck. To address this, we focused on streamlining the matching process, aiming to make a notable difference in overall performance without compromising accuracy. The results exceeded our expectations: we achieved a near 100% performance increase in processing speed while maintaining ByteTrack's impressive accuracy. This significant advancement paves the way for faster, more efficient object tracking in diverse real-world scenarios.

II. RELATED WORKS

When we searched the literature, we could not find any project group developing on ByteTrack. Instead, there are other methods in the field of MOT based on different ideas. For example, some methods use Particle Filter and Ensemble Random Forest Filter to predict and compare new positions of objects, while ByteTrack uses Kalman Filter.

Multi-object tracking methods can be categorized into detection-based, appearance-based, and feature-based methods. DB methods first detect objects in each frame of the video and then associate the detections across frames to form tracks such as FairMOT [1], and ByteTrack. AB methods track objects by modelling their appearance features, such as color and texture like SiamMOT [2]. FB methods track objects by modeling their motion features, such as velocity and acceleration such as DeepSORT [3] and TransMOT [4].

III. DATA

We worked on 4 different datasets. We downloaded them all from Kaggle. The first one was from Bundesliga matches, consisting of 20 clips which are 30 seconds. The other was a 10 second traffic video taken from daily life. Another one was the MOT17 dataset reserved for performance testing within the source codes of the project. This includes footage taken from a street camera in China. The last one was a 10 second video of people in the Tokio airport.

Since YOLO was used for the initial detection of objects and it is a pretrained model, we used the data we had only for performance testing.

Even though we did not make any changes in the preprocessing stages of the project's source codes, the following are the things that were done:

Otsu's Thresholding: This step increases the contrast of the image, making it easier to distinguish objects from the background.

Binarization: This step converts the image to black and white. This makes it easier for the model to detect objects.

Sampling: This step reduces the size of the image.

Methods performed as postprocessing: image resizing, coloring, framing, information highlighting. As mentioned in the 5. part, with the latest supervision library, it was possible to process much more understandable box annotators on output clips and frames.

IV. METHODS

In this project, we aimed to improve the processing time of ByteTrack by focusing on the most time-consuming stages. We considered a variety of approaches, including:

Particle Filter: Particle filters are a well-established method for tracking objects in video. They are typically more accurate than Kalman filters however the performance gains would not be worth the additional computational cost in this situation.

Ensemble Random Forest Filter: ERF filters [5] are a type of Bayesian filter that can be used for tracking

objects in video. They are typically more accurate than Kalman filters, but they can also be slower when the background is overly complex.

Optimizing Kalman Filter: In ByteTrack's current Kalman filter, in the predict() and update() methods, the _motion_mat and _update_mat matrices are recalculated each time. This causes unnecessary calculations. Optimization could be done by calculating and storing these matrices once. Calculation of the covariance matrix in predict() and up-date() methods; It is performed using the np.linalg.multi_dot() function. This function is designed to efficiently calculate matrix multiplications. However, this function may still involve some unnecessary calculations. The np.linalg.block_diag() function could be used to optimize these calculations.

The Gaussian Kalman Filter is a popular filter for linear systems. However, for nonlinear systems it has to perform linearization. This process may introduce linearization error and reduce prediction accuracy.

Unscented Kalman Filter does not perform linearization for nonlinear systems. Instead, it uses a set of points that accurately represent the behavior of nonlinear systems. These spots are created using a process called unscented conversion. The unscented transformation distributes random points in the state space, and these points consider the dynamics and measurements of the system.

The matching algorithms used by ByteTrack are responsible for associating object detections from different frames. The current implementation of ByteTrack uses the Lucas-Kanade algorithm. We considered using the Hungarian algorithm as an alternative. We also considered using the Delaunay triangulation algorithm. It is a geometric algorithm that can be used to find the best possible matches between object detections.

V. EXPERIMENTS

When we tested the methods that determined respectively, we achieved the best results with the Hungarian matching algorithm and the improved Kalman filter we created. We used RoboFlow's new library, supervision, to visualize the detection, tracking and counting the objects in the videos we worked on.

We used MOTA for accuracy loss while observing the change in speed as a performance metric. Multiple Object Tracking Accuracy is a key metric in computer vision projects that evaluate the performance of algorithms tracking multiple objects in a video sequence. It essentially tells us how accurate our tracker is at identifying and following individual objects throughout the video.

We got these results when we took any frame from our road_traffic_clip video and ran the model on it before the changes were made:

384x640 5 persons, 7 cars, 4 motorcycles, 1 bus, 1 traffic light, 1 bench, 1 backpack, 253.7ms Speed: 4.5ms preprocess, 253.7ms inference, 928.0ms postprocess per image at shape (1, 3, 384, 640)

Figure 1

The output after the improvements were made:

384x640 5 persons, 7 cars, 4 motorcycles, 1 bus, 1 traffic light, 1 bench, 1 backpack, 63.1ms Speed: 2.6ms preprocess, 63.1ms inference, 2.7ms postprocess per image at shape (1, 3, 384, 640)

Figure 2

The output with airport_people_crowd_clip before the changes were made:

384x640 12 persons, 6 handbags, 64.8ms Speed: 14.4ms preprocess, 64.8ms inference, 7.2ms postprocess per image at shape (1, 3, 384, 640)

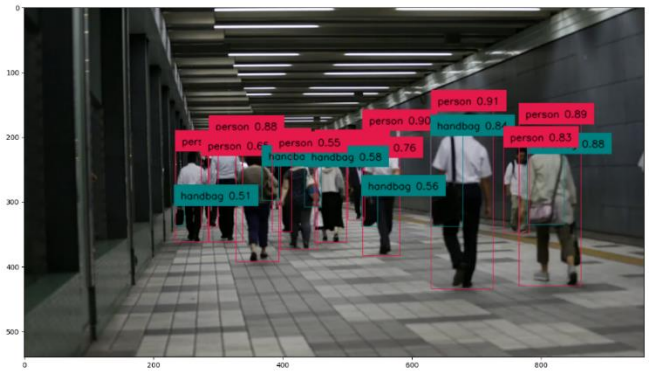

Figure 3

The output after the improvements were made:

384x640 12 persons, 6 handbags, 64.5ms Speed: 7.4ms preprocess, 64.5ms inference, 2.2ms postprocess per image at shape (1, 3, 384, 640)

Figure 4

Before the changes, when we draw a line from point (250,250) to (400,200) and count the number of vehicles passing towards the +x direction we determined:

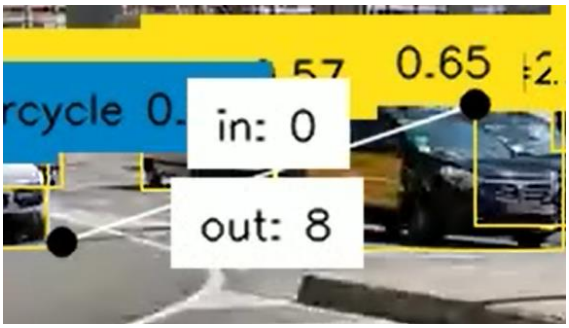

Figure 5

After the changes, when we draw a line from point (250,250) to (400,200) and count the number of vehicles passing towards the +x direction we determined:

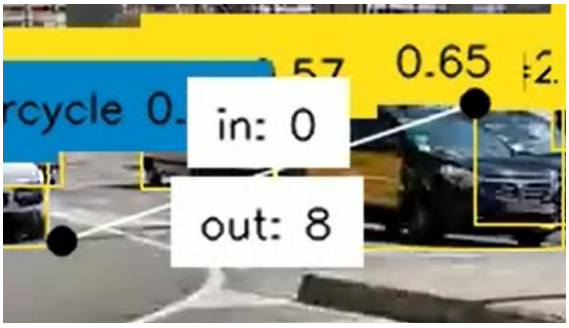

Figure 6

We can conclude that there was no huge of accuracy decrease in tracking and counting. When we work on other data sets and take the gain averages, there is speeding success almost 100%.

for road_traffic_clip	without improvements	with improvements	gain in speed (%)
Speed (ms)	253,7	63,1	402.06
Preprocessing (ms)	4,5	2,6	
Postprocessing (ms)	928,0	2,7	

Figure 7

for MOT17	without improvements	with improvements
FP	25491	25480
FN	83721	83683
IDs	2196	2237
MOTA	80,3	78,89

Figure 8

VI. CONCLUSION

The ByteTrack aims to improve the performance of multi-object tracking by associating every detection box instead of only the high score ones. We achieved consistent improvement on the mentioned metrics. By working on this project, we have learnt: the principles of MOT, implementing different MOT algorithms, evaluating and improving the performance of them.

REFERENCES

[1] Zhang, Z., et al. : FairMOT: Towards a Fair Detection and Tracking Benchmark. arXiv:2004.07755 (2020).

[2] Peng, Z., et al. : SiamMOT: Siamese Multiple Object Tracking. Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. No. 7. 2020.

[3] Bewley, A., et al. : Simple Online and Real-Time Multiple Object Tracking Using Kalman Filter and Hungarian Algorithm. Proceedings of the Asian Conference on Computer Vision. Springer, Cham.

[4] He, T., et al. : TransMOT: A Transformer-Based Method for Multiple Object Tracking. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, 2021.

[5] Godoy, V., Napa-Garcia, G., Gomez-Hernandez, J. : Ensemble Random Forest Filter: An Alternative To The Kalman Filter For Inverse Modeling. 10.48550/arXiv.2207.03909 (2022).