Object Tracking Web Application to Compare Model Performances

Sherly Hartono   
Khouy College of Computer Sciences

Northeastern UniverityBoston, USA

hartono.s@northeastern.edu

*Abstract*—This project devises a web application to perform real-time multiple objects tracking and display the performances of three YOLO version models and Faster RCNN using React framework for front end and Flask for backend. Deep SORT algorithm is used for the tracking stage.

***Keywords - Multiple Object Tracking (MOT); YOLOv7; YOLOv5; YOLOv4; Faster RCNN; Web Application; Deep SORT***

# Introduction

Object tracking is one of the most important tasks in computer vision. Its real-life application includes, but not limited to, traffic monitoring, autonomous driving, and autonomous robots in manufacturing. While object detection simply produces bounding boxes and their corresponding classes as output, multiple object tracking requires the algorithm to also assign a target ID to each bounding box. So instead of identifying all cars in a frame as just a “car”, MOT algorithm will attempt to identify each car as separate objects.

Although there are many projects available on combining YOLO with tracking algorithms to create an object tracker, it was difficult to find one where these models are deployed on a full web application that uses client-server architecture. Furthermore, in real-life, there will be multiple models that are available for different use cases. So in addition to making the model run on the browser, this project also evaluates the models using three metrics that are displayed on the dashboard at the end of the session. The metrics calculated are standard deviation of the confidences, standard deviation of the box sizes, frame per seconds, and the number of objects detected. These results are saved to a csv file after each session thus allowing users to compare different performances under different environments. The models that will be used are YOLO version 4, 5, 7 and Faster RCNN.

# Related Work

Faster RCNN [1] is one of the first algorithms that reaches near real-time object detection as it uses Region Proposal Network (RPN) which fixes the problem of selective search that was faced by its predecessor, Fast RCNN. Faster RCNN with ResNet backbone has accuracy of AP 36.2, which is better than YOLO early versions with AP=33.0 for YOLOv2, and AP=21.2 for YOLOv3 when trained on MS COCO dataset according to [2]. Due to this reason, even though it is much slower than YOLO, it is still worth evaluating its overall performance for use in object tracking. We will also look at YOLOv4 [3], YOLOv5 [4], and YOLOv7 [5] as they are renowned for their speed and thus are more suitable for real-time MOT tasks. This is because with YOLO architecture, unlike Faster RCNN, it does not need to look twice to get the categories and bounding boxes of the objects. We should also take note that YOLO has a limitation on detecting small objects that appear in groups [6].

For tracking we will be using Deep SORT algorithm [7] which is an extension of the original SORT algorithm. In addition to performing Kalman filtering in image space and frame-by-frame data association of using Hungarian method, Deep SORT integrates appearance information to improve the performance of SORT. This reduces the number of identity switches during periods of occlusion.

# Methods

The project is split into two main parts. The first part is the backend that contains the deep learning models and other modules that evaluate the model performances, save these results in database and csv file then send them back to the client browser. The second part is the front end that grabs users input and displays the video frames, and evaluation results. It allows users to create different sessions and test the four models under different environments. The program also allows users to choose different inputs either from live webcams or from a given recorded video.

All the models are pre trained using MS COCO dataset so that meaningful comparison can be made between them. The detection and tracking is performed using Apple M1 CPU with 3.2 GHz speed and ARM instruction set. Thus, we expect lower performance compared to those observed in the official papers that executed tasks with GPU engines. For this reason, the smallest versions of each of the models are used.

For Faster RCNN the backbone used is ResNet-50-FPN that is available from Pytorch’s torchvision library. This is the backbone that is used in the original paper [1]. Note that Feature Pyramid Network is used which generates multi-scale feature maps with better quality than the regular feature pyramid for object detection.

YOLOv4-tiny, YOLOv5-s, YOLOv7-tiny that this project is using have 6.1 million, 7.2 million, and 6.2 million number of parameters respectively [5]. To run YOLOv4 and v7 we can simply obtain their weights and config files from [8] which is provided by Darknet, an open-source Neural Network framework. We use OpenCV’s dnn module to read these weights and config files and initialize the model. Non maxima suppression threshold to select the best bounding box and confidence threshold is set 0.4 and 0.3 respectively. Using the given detect method from the dnn module, it will take OpenCV’s image frame as an input and output the class ids, scores, and bounding boxes’ x and y’s minimum and coordinates and their width and height as a NumPy array. For YOLOv5 we can use the code from [9] by loading it from Pytorch Hub [10]. For Faster RCNN and YOLOv5 the bounding box outputs are coordinates only so we first need to compute the weight and height of the bounding boxes before passing it to our Deep SORT tracker.

We can then grab these frames and their corresponding bounding boxes information and pass it to Deep SORT to extract features and generate detections that can be recognized by Kalman filter update step. We propagate the state distribution to the current time step, then given new detection, we perform Kalman filter update step to update the prediction. The official code for the Deep SORT algorithm can be obtained from [11].

After the tracking process, some object ids can disappear. That is, the tracker does not think the object that was in the previous frame is still there. We can see the example in “Fig. 1”. Deep SORT failed to track the chair after detection. So we will only keep scores and sizes where the number of objects detected before and after tracking are the same, which means the scores are correctly associated with the same objects in the order of accuracy.

Graphical user interface, text

Description automatically generated

1. Object id 31 refers to the person at index 0. The chairs are not tracked after Kalman update.

For each frame, we then collect the scores and normalized box sizes ­­so we can calculate the standard deviation after every 3 seconds to measure the model’s stability. We will also compute the number of frames per second (FPS) and the number of objects detected per frame. These results will be saved in SQLite database so that the data can persist at every session. At the end of every session, a csv file is outputted. A real time chart is also displayed at every 3 seconds update in the browser so the user can track the model performance in real time. At the end of the session a summary of these results are computed to give the average standard deviations, number of objects detected, and FPS of every model used during a session.

# Experiments And Results

The experiments are divided into three parts. The first part is to compare the models in quiet versus busy scenes. The second part is to test the impact of illumination on model performance while the last part is to check how Deep SORT performs during occlusion.

## Comparing Busy Traffic Versus Quiet Library Scene

In a single session the models are alternated after 8-10 data points. This is done twice with the second half of the session done in random order. This is visualized in “Fig 4”. After every session, a result summary is displayed as shown in “Fig5” and “Fig 6” for the first two session summary results. In both sessions, Faster RCNN has the most stable confidence score with standard deviation of 1.91 followed by YOLOv5 with standard deviation of 2.59. YOLOv4 performs the worst in Scene 1 and YOLOv7 performs the worst in Scene 2. In terms of size, again Faster RCNN and YOLOv5 performs the best and YOLOv4 performs the worst with standard deviation of 0.012 in the traffic scene. Note that the standard deviation of size increases after every model swap.

Faster RCNN seems to be paying the cost of stability with speed as it barely reaches 1 FPS in both sessions and is therefore the slowest model. YOLOv5 is the fastest model with 9.5 FPS in session one whereas in session two YOLOv7 is the fastest with 4.23 FPS. In the traffic scene, the FPS of all the three YOLOs do not differ by much as they fall between the range of 4.10 to 4.24. In the library scene, the range is much wider - between 5.14 - 9.53 FPS.

All the models apart from YOLOv7 capture a similar number of objects, around 4 to 5 objects in a frame. YOLOv7 captured the least number of objects of 1.97. We can see how this is represented in “Fig 2”. In session two however, the number of objects detected by Faster RCNN doubled or even quadrupled that of YOLO. This can be clearly seen in “Fig 3”. The most visible difference can be seen by how the group of people on the left around the stairs are not detected by any of the YOLO models.

A collage of a person

Description automatically generated with medium confidence

1. Scene 1 of the experiments that shows different bounding boxes detected depending on the models

A picture containing text, indoor, cluttered, several

Description automatically generated

1. Scene 2 of the experiments and the bounding boxes deteced by the different models

Chart, line chart

Description automatically generated

1. Scene 2 of the experiments and the bounding boxes deteced by the different models

Table

Description automatically generated

1. Scene 1 session result

Table

Description automatically generated

1. Scene 2 session result

## Comparing Low vs High Illumination

The second part of the experiments involves the same scene in normal lighting for scene 3 and in low lighting for scene 4. There are only two objects that can be detected, a person and a chair. “Fig 8” and “Fig 9” illustrates the score summary of these experiments. For normal illumination, YOLOv7 has the most stable confidence score with the lowest standard deviation of just below 1.9 while YOLOv4 performs the worst at 4%. In low illumination however, the rank is almost flipped as YOLOv7 standard deviation shot up to 10.03% while YOLOv4 deviation remains low at 3.3%.

In both scenes the FPS of all the models is higher than in the first part of the experiments due to lower number of objects detected. YOLOv5 and v7 in low lighting condition are the fastest performing models with over 8 FPS.

With regards to the number of objects, all the models can detect the chair apart from YOLOv7 under bright light. In low illumination scenes however, only Faster RCNN can detect the chair as seen in “Fig 7”. Faster RCNN also displays unusual behavior in that its standard deviation on confidence score is among the highest in low-light mode compared to other models. Whereas in the other three scenes Faster RCNN is almost always the most stable in confidence score.

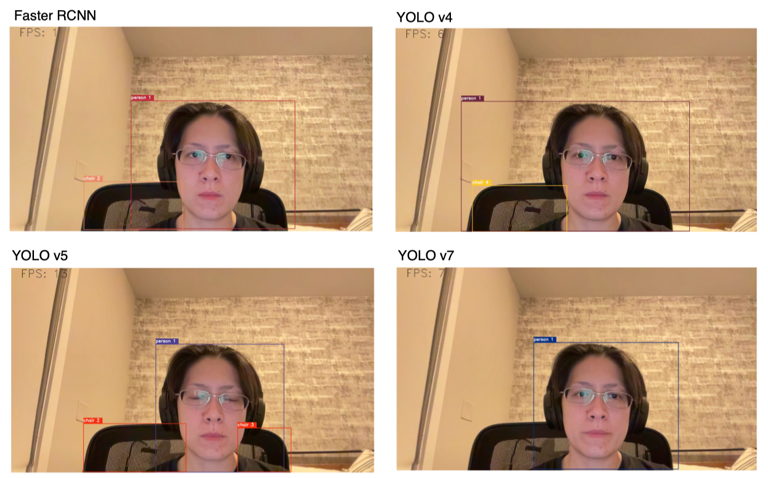


Fig 7. Scene 3 of  the experiment showing a person in a normal lighting environment

A collage of a person

Description automatically generated with low confidence

Fig 8. Scene 4 of  the experiment showing a person in low lighting environment

## Testing Occlusion

For testing occlusion, we will look at inter-object occlusion. That is when one of two objects being tracked occludes the other object. We will be testing two scenarios. The first scene involves two different objects, a book and a person. The experiment is to check if the person is still identified under the same ID after she has been occluded by a book. The second scene involves a particular area of a football match scene. One player will occlude the other player and we will check if there is an identity switch after one object occludes the other.

The book and the person scene can be seen in “Fig 10”. Out of the four models, only YOLOv4 assigned a new ID to the same person after the person has been occluded by a book. The rest of the models successfully assigned the same ID to the person and when the person is completely covered by the book, the same three models assign the person ID to the book and detect it as a book. None of the models has successfully tracked the book. It either has not identified the book or misidentified it as a refrigerator or laptop or lost track of the book.

The result of the football occlusion scene can be seen in “Fig 11”. None of the models correctly tracked the person after occlusion. Faster RCNN and YOLOv5 flipped the players’ ID after occlusion and YOLOv5 even assigned a new ID to one of the players.

A collage of a person's face

Description automatically generated with low confidence

Fig 10. Testing occlusion problems of each model. A book is dragged across a person. If a person with the same ID is identified in the subsequent frame, the text will be highlighted in green.

A picture containing website

Description automatically generated

Fig 11. Scene 6. A scene where two players, one with black shirt and another with blue shirt is tracked after the blue shirt player occludes the black shirt player.

# Discussion and Summary

The FPS results of all the experiments align with the fact that YOLO runs faster than Faster RCNN due to its simpler architecture. All the YOLO models run at least 4 times faster than Faster RCNN. This is because unlike faster RCNN, it’s trained to do “objectness”, classification, and regression of the bounding boxes in a single shot.

This speed however is a trade off with accuracy as Faster RCNN is able to detect more objects correctly. This was illustrated in the first experiments where Faster RCNN can detect many more objects both in the library, in traffic, and in low illumination. This also aligns with the original YOLO paper [1] that acknowledged the limitation of YOLO in detecting small objects that cluster together. In three of our scenes, the objects that cluster together; chairs in the library, people in the traffic, football players, were all detected poorly by all the YOLO models.

YOLOv7 surprisingly always performed poorly compared to YOLOv5. It has higher standard deviation in confidence score and box sizes in three out of four scenes. YOLOv5 is the fastest model among all and Faster RCNN is the most sensitive in that it can detect the most objects. Both models performs the best in terms of confidence score stability.

##### References

1. R. Shaoqing *et al.*,“Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *arXiv:1506.01497* *[cs.CV],* 2016.
2. L. Jiao *et al* “A Survey of Deep Learning-based Object Detection,” arXiv:1907.09408v2 [cs.CV], Oct. 2019.
3. [A. Bochkovskiy, C.Y. Wang, H.Y. Mark Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv:2004.10934*](https://arxiv.org/abs/2004.10934) *[cs.CV],* 2020.
4. G. Yang *et al*., "Face Mask Recognition System with YOLOV5 Based on Image Recognition," 2020 IEEE 6th International Conference on Computer and Communications (ICCC), 2020, pp. 1398-1404, doi: 10.1109/ICCC51575.2020.9345042.
5. [A. Bochkovskiy *et al*., “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,“](https://arxiv.org/abs/2004.10934) *arXiv:2207.02696 [cs.CV],* 2022.
6. J. Redmon *et al*., “You Only Look Once: Unified, Real-Time Object Detection,” *arXiv:1506.02640 [cs.CV],* 2016.
7. N. Wojke et al., “Simple Online and Realtime Tracking with a Deep Association Metric,” *arXiv:1703.07402 [cs.CV],* 2017.
8. A. Bochkovskiy (2021) darknet [Source code]. https://github.com/AlexeyAB/darknet/releases/.
9. G. Jocher (2020) YOLOv5 [Source code]. https://github.com/ultralytics/yolov5
10. N. Wojke (2019) Deeo SORT [Source code]. https://github.com/nwojke/deep\_sort