



Real-time multiple object tracking using deep learning methods

Dimitrios Meimelis¹ · Ioannis Daramouskas¹ · Isidoros Perikos¹ · Ioannis Hatzilygeroudis¹

Received: 21 October 2020 / Accepted: 26 July 2021 / Published online: 24 August 2021
© The Author(s), under exclusive licence to Springer-Verlag London Ltd, part of Springer Nature 2021

Abstract

Multiple-object tracking is a fundamental computer vision task which is gaining increasing attention due to its academic and commercial potential. Multiple-object detection, recognition and tracking are quite desired in many domains and applications. However, accurate object tracking is very challenging, and things are even more challenging when multiple objects are involved. The main challenges that multiple-object tracking is facing include the similarity and the high density of detected objects, while also occlusions and viewpoint changes can occur as the objects move. In this article, we introduce a real-time multiple-object tracking framework that is based on a modified version of the Deep SORT algorithm. The modification concerns the process of the initialization of the objects, and its rationale is to consider an object as tracked if it is detected in a set of previous frames. The modified Deep SORT is coupled with YOLO detection methods, and a concrete and multi-dimensional analysis of the performance of the framework is performed in the context of real-time multiple tracking of vehicles and pedestrians in various traffic videos from datasets and various real-world footage. The results are quite interesting and highlight that our framework has very good performance and that the improvements on Deep SORT algorithm are functional. Lastly, we show improved detection and execution performance by custom training YOLO on the UA-DETRAC dataset and provide a new vehicle dataset consisting of 7 scenes, 11.025 frames and 25.193 bounding boxes.

Keywords Computer vision · Multiple-object tracking · Deep learning · Deep SORT · YOLO

1 Introduction

Computer vision is a fundamental domain that aims to allow computer systems to analyze images, extract knowledge and interpret them as humans do. Multi-object tracking (MOT) is also called multi-target tracking (MTT) and has a very important part in computer vision [27]. In general, the task of MOT is largely partitioned to locating multiple objects, maintaining their identities and yielding

their individual trajectories given an input video [1]. MOT aims to process videos with the purpose to identify and track objects that belong to one or more categories, like cars, pedestrians, objects, animals without any prior knowledge concerning the appearance, the movement and the number of targets [2, 26]. Many computer vision problems depend to a large extent on multiple-object tracking systems [18, 28]. There are two important steps involved in designing such systems. The first step involves the detection of the objects [23]. So, the desired objects are detected in each frame of the video. The quality of the detection directly affects the performance of the overall monitoring procedure. The second step involves the matching of the identified objects to the previous ones to get their trajectories. High accuracy in the object detection system results in a smaller number of missing detections and ultimately produces smooth and accurate trajectories. Multiple-object tracking is also considered the process of locating multiple moving objects over time [27, 31, 41]. Multiple-object tracking systems have a variety of uses in a wide range of topics like security, video communication

✉ Isidoros Perikos
perikos@ceid.upatras.gr

Dimitrios Meimelis
dmeimelis@ceid.upatras.gr

Ioannis Daramouskas
daramousk@ceid.upatras.gr

Ioannis Hatzilygeroudis
ihatz@ceid.upatras.gr

¹ Computer Engineering and Informatics Department,
University of Patras, Patras, Greece

and compression, traffic control, medical imaging, self-driving cars and robotics [24, 29, 30].

Although object tracking is quite useful and desired, accurate, real-time object tracking is quite challenging and things are even more challenging when multiple objects are involved [37, 38]. One major issue that MOT is facing is the bounding box level tracking performance and saturation; therefore, most research is focused on handling these aspects [3, 4]. For tracking to perform sufficiently, object detection is necessary to work flawlessly across all frames and not having to use interpolation [39]. However, this is almost impossible due to occlusion, the variety in viewpoints and the noise that may be introduced in a video. Also, real-time tracking requires great computational resources and also needs to face challenges like the identity switches and various detection failures [33, 34, 40].

In this paper, first we explore the performance of various deep learning methods on the task of multiple-object tracking. We examine how widespread deep learning architectures are performing under various contexts in a wide range of scene scenarios. Also, the paper introduces a modification of the Deep SORT [5, 25] algorithm, which greatly improves the performance of object tracking methods, using different object detection models, such as YOLOv3-608 [6], YOLOv3-Tiny [6] and YOLOv4 [7]. The Deep SORT implementation is an extension to the simple online and real-time tracking (SORT) [8] algorithm and the SORT framework is utilized too as a mean to measure bounding boxes overlaps. While this is a high performance method, the number of identity switches due to occlusions from poor camera angles are too many [9–12]. By incorporating convolutional neural networks, to additionally include appearance information, Deep SORT can substantially reduce identity switches. Our modification on Deep SORT is based on the process of the initialization of the tracked object IDs and the way they are assigned and passed through, to be shown during the visualization process. The results indicate that our modified Deep SORT now properly displays the track IDs and it is closer to the ground truth in all the examined cases, a problem that exists on all YOLO & Deep SORT implementations we have found on the Internet. In addition, we present a way to improve the real-time operation of the deep learning methods by identifying and facing a bottleneck in the MOT framework. We tested and provide a way that can greatly improve the execution time of the tracking procedure. The results show that we have an increase of frames per second (FPS) in all examined deep learning networks which is up to 22%.

The novelty and the contribution of this paper can be summarized as follows. First, we explore the performance of various deep learning methods on the task of real-time multiple-object tracking. Our focus is road traffic, but we

also include pedestrian tests and we compare a grand total of 7 YOLO derivatives ranging from YOLOv3-Tiny all the way to a fully fledged YOLOv4 implementation. For the tracking mechanism, we use the DEEP Sort framework, which we modified to properly display the correct track ID in the real-time video feedback, a problem that occurred in all YOLO and Deep Sort implementations we could find on the Internet. For each implementation, we provide the optimal detection and tracking parameters which could be useful for fellow researchers and hobbyists. Moreover, we perform custom transfer learning training of the YOLO detector using a slightly modified version of the UA-DETRAC dataset. The UA-DETRAC trained YOLOv4 provided state-of-the-art performance when compared to the publicly available MS-COCO trained YOLOv4 in our test scenes. In addition to that, we also performed performance characterization on this framework and found a bottleneck in the execution pipeline, which we resolved, and we saw an execution performance increase of up to 22%. For the evaluation process, we provide a wide variety of metrics and we test nine different scenes, six of which are our own. For all test scenes, we also provide the ground truth files, which we generated either from the ground up or from existing data. Finally, through the creation of the ground truth files, we also provide a new vehicle multiple-object dataset consisting of 7 scenes, 11,025 frames and 25,193 bounding boxes.

The rest of the paper is structured as follows. Section 2 examines the literature and presents related works and methods on multiple-object tracking. Section 3 presents our implementation that is based on modified Deep SORT algorithm and the YOLO detection networks. The modification made on Deep SORT algorithm is presented and the way it affects the visualization of tracking of moving objects is illustrated. Section 3 describes the way that some bottlenecks in the multiple objects tracking procedure is faced and presents the way that improves the real-time performance in terms of frames per second. Section 4 deals with the experimental study, explains the datasets used for the training phases and for the testing phase and presents the results collected. Finally, Sect. 5 concludes the article and draws directions for future work.

2 Related work

Multiple-object tracking is attracting the increasing attention of researchers in computer vision and artificial intelligence. Several works in the literature study the performance of methods and systems and a detailed description of approaches and techniques can be found in [1, 2, 12].

In the work presented in [3], authors propose an online multi-target tracker that exploits both high and low-confidence target detections in a probability hypothesis density particle filter framework. Authors formulate an early association strategy between trajectories and detections after the prediction stage, which allows performing target estimation and state labeling without any additional mechanisms. The authors' solution has a peak multiple-object tracking accuracy (MOTA) score of 53 on MOT15 and 52.5 on MOT16.

Authors in [8] present an approach to multi-object tracking where the main focus is to associate objects efficiently for online and real-time applications. To this end, detection quality is identified as a key factor influencing tracking performance, where changing the detector can improve tracking by up to 18.9%. Despite only using a rudimentary combination of familiar techniques such as the Kalman filter and Hungarian algorithm for the tracking components, the approach achieves accuracy that is comparable to state-of-the-art online trackers. Additionally, emphasis is placed on efficiency for facilitating real-time tracking and to promote greater uptake in applications such as pedestrian tracking for autonomous vehicles. While being an overall good framework at the time, the identity switches are rather high with a value of 1001 in the MOT benchmark. Their solution has a peak MOTA score of 33.4 on MOT15.

In the work presented in [4], authors present an online method that encodes long-term temporal dependencies across multiple cues. One key challenge of tracking methods is to accurately track occluded targets or those which share similar appearance properties with surrounding objects. To address this challenge, authors present a structure of recurrent neural networks (RNN) that jointly reasons on multiple cues over a temporal window. Their motion and interaction models leverage two separate long short-term memory (LSTM) networks that track the motion and interactions of targets for a longer period—suitable for the presence of long-term occlusions. Their solution has a peak MOTA score of 37.6 on MOT15.

In the work presented in [2], authors present a comprehensive survey on works that employ deep learning models to solve the task of MOT on single-camera videos. Four main steps in MOT algorithms are identified, and an in-depth review of how deep learning was employed in each one of these stages is presented. A complete experimental comparison of the presented works on the three MOTChallenge datasets is also provided, identifying a number of similarities among the top-performing methods and presenting some possible future research directions.

In the work presented in [13], authors build on a neural class-agnostic single-object tracker named HART and introduce a multi-object tracking method MOHART

capable of relational reasoning. Authors explore a number of relational reasoning architectures and show that multi-headed, self-attention outperforms the provided baselines and better accounts for complex physical interactions in a toy experiment. Authors find that it leads to consistent performance gains in tracking as well as future trajectory prediction on three real-world datasets (MOTChallenge, UA-DETRAC and Stanford Drone dataset), particularly in the presence of ego-motion, occlusions, crowded scenes and faulty sensor inputs. On the MOTChallenge dataset, HART achieves 66.6% IOU, which itself is impressive given the small amount of training data of only 5225 training frames and no pre-training.

In the work presented in [14], authors present an end-to-end model, named FAMNet, where feature extraction, affinity estimation and multi-dimensional assignment are refined in a single network. All layers in FAMNet are designed differentiable and thus can be optimized jointly to learn the discriminative features and higher-order affinity model. Authors also integrate single-object tracking technique and a dedicated target management scheme into the FAMNet-based tracking system to further recover false negatives and inhibit noisy target candidates generated by the external detector. The proposed method is evaluated on a diverse set of benchmarks including MOT2015, MOT2017, KITTI-Car and UA-DETRAC and achieves promising performance on all of them in comparison with state-of-the-art. The authors' method has a MOTA score of 40.6 on MOT15.

In the work presented in [15], authors introduce a focal loss-based RetinaNet, which works as one-stage object detector, is utilized to be able to well match the speed of regular one-stage detectors and also defeats two-stage detectors in accuracy, for vehicle detection. State-of-the-art performance result has been shown on the DETRAC vehicle dataset. This is important because one-stage object detectors and two-stage object detector are regarded as the most important two groups of convolutional neural network-based object detection methods. One-stage object detector could usually outperform two-stage object detector in speed; however, it normally trails in detection accuracy, compared with two-stage object detectors.

In the work presented in [16], the authors introduce deep motion modeling network (DMM-Net) that can estimate multiple objects' motion parameters to perform joint detection and association in an end-to-end manner. DMM-Net models object features over multiple frames and simultaneously infer object classes, visibility and their motion parameters. These outputs are readily used to update the tracklets for efficient MOT. DMM-Net achieves PR-MOTA score of 12.80 @ 120+ fps for the popular UA-DETRAC challenge. Authors also introduce a synthetic large-scale public dataset Omni-MOT for vehicle tracking

that provides precise ground-truth annotations to eliminate the detector influence in MOT evaluation.

In the work presented in [17], authors present a CNN-based framework for online MOT. This framework utilizes the merits of single-object trackers in adapting appearance models and searching for target in the next frame. Simply applying a single-object tracker for MOT will encounter the problem in computational efficiency and drifted results caused by occlusion. Their framework achieves computational efficiency by sharing features and using ROI-Pooling to obtain individual features for each target. In the framework, they introduce spatial-temporal attention mechanism (STAM) to handle the drift caused by occlusion and interaction among targets. Besides, the occlusion status can be estimated from the visibility map, which controls the online updating process via weighted loss on training samples with different occlusion statuses in different frames. It can be considered as temporal attention mechanism. The proposed algorithm achieves 34.3% and 46.0% in MOTA on challenging MOT15 and MOT16 benchmark datasets, respectively.

3 Methodology

In this section, we present our framework for object tracking that relies on the modification of the Deep SORT algorithm. We describe the main methods for the object detection and tracking architecture of this implementation. More specifically, for the object detection procedure, YOLO models are utilized to detect desired objects in a frame, and after that, a modified version of Deep SORT algorithm is introduced to perform object tracking in the sequences of the frames. Our modification on the Deep SORT algorithm concerns the process of the initialization of the object IDs, and its rationale is to consider an object as “tracked” if it is detected in a set of previous frames. We assess the performance of the YOLO object detection models trained on the MS COCO and UA-DETRAC datasets using transfer learning in order to assess their performance and identify suitable and optimal synergies for multi-object tracking. The modified Deep SORT algorithm is tested using the YOLO models in tracking cars and pedestrians, a variety of datasets and scenes, and its performance is compared to the original Deep SORT algorithm. The results indicate that our modified Deep SORT algorithm now properly displays the assigned track IDs, while also providing good tracking performance. In the following subsections, we present the modification made on the Deep SORT algorithm, the implementation of the YOLO models as well as the optimization of our framework.

3.1 Modified deep SORT tracking algorithm

One of the most widely used object tracking frameworks is Deep SORT, which is an extension to SORT (simple real-time tracker) [5]. Deep SORT achieves better tracking and less identity switches by including an appearance feature vector for the tracks which is derived, in this case, by a pre-trained CNN that runs on the YOLO detected bounding boxes. Since simple detection models are very likely to fail at detecting numerous objects consecutively as the frames go by, we need to add new methods to keep track of them and properly identify them. This is where Deep SORT comes in to make a proper MOT framework.

The Kalman filter is a crucial component in Deep SORT. Each state contains 8 variables (u, v, a, h, u', v', a' and h') where (u, v) are the coordinates of the bound box, a is the aspect ratio, and h is the height of it. The respective velocities are given by u', v', a', h' . The state contains only absolute position and velocity factors, since we assume a simple linear velocity model. The Kalman filter helps us face the problems that may arise from non-perfect detection and uses prior states to predict a good fit for future bounding boxes. Now that we have the new bounding boxes tracked from the Kalman filter, the next problem lies in associating new detections with the predictions that have been created. Since they are processed independently, a method is needed to associate $track_i$ with incoming $detection_k$. To solve this, Deep SORT implements 2 things: a distance metric to quantify the association and an efficient algorithm to associate the data. The authors decided to use the squared Mahalanobis distance (effective metric when dealing with distributions) to incorporate the uncertainties from the Kalman filter. Thresholding this distance can give us a very good idea on the actual associations. This metric is more accurate than, say, Euclidean distance, as we are effectively measuring the distance between 2 distributions. For the data association part, the Hungarian algorithm is used. Lastly, the feature vector becomes our “appearance descriptor” of the object. The authors have added this vector as part of the distance metric. Now, the updated distance metric is:

$$D = \text{Lambda} * D_k + (1 - \text{Lambda}) * D_a$$

where D_k is the Mahalanobis distance and D_a is the cosine distance between the appearance feature vectors and Lambda is the weighting factor. The importance of D_a is so high that the authors make a claim saying, they were able to achieve state of the art even with Lambda = 0, meaning that they only used the appearance descriptor for the calculation. We provide the pseudocode for the Deep SORT-enabled framework below.

1. For every frame:
2. Perform prediction for the tracks using Kalman filter.
3. For every yolo detection:
4. Get detection features
5. Run non-maxima suppression
6. Calculate squared Mahalanobis distance based on predicted Kalman states
7. Find smallest feature cosine distance for every existing track
8. Update tracker with the use of IOUs and the Hungarian algorithm.
9. If initiated track has been consecutively detected for n_{init} frames then confirm track
10. Else:
11. Delete track.
12. If confirmed track has been consecutively not detected for MaxAge frames then delete track.

The variables that cause the biggest change in performance are score and IOU of each respective model, and n_{init} , $max_cosine_distance$ and $max_iou_distance$ from the Deep SORT framework. We will present the optimal values for each implementation in the videos used. We keep $max_cosine_distance$ and max_iou distance at the same values, of 0.4 and 0.7, respectively, for all our tests. We also set our n_{init} at 7, unless stated otherwise. The variable n_{init} dictates how many successful detections a track must have before it goes from its initial tentative state to its confirmed state.

A main aspect we noticed on the functionality and the utilization of Deep SORT concerns the fact that shows the initiated track IDs on the bounding boxes and not the confirmed tracks. This operation may cause problems in tracking and numbering correctly the detected objects in the sequences of frames. To address this problem, a main modification that we implemented to the Deep SORT

algorithm relates to the proper display and count of the confirmed detections. Specifically, each track has three states: the *initial tentative state*, the *confirmed state* and the *deleted state*. Every new track is classified as tentative for the first n_{init} frames. If the n_{init} frames pass and the track is still identified, it will become confirmed and feature similarity will also be employed. If a track fails to be identified properly using IOU similarity for every frame in the n_{init} phase, then it will be classified as deleted. We made sure that every bounding box shown on screen has the proper confirmed state ID on it.

Below we provide some example cases of the comparison results between the Deep SORT and the modified version. In Figs. 1 and 2, we present a case from the MOT15 dataset, where the ground truth for that part of the scene is 34 people. Our framework of the modified deep SORT measured 32 people, while the original Deep SORT resulted in measuring 79 people as it is illustrated in the

Fig. 1 Non-modified Deep SORT+YOLO framework. Frame ground truth is 34



Fig. 2 Results after our modifications. Frame ground truth is 34



Fig. 3 Non-modified Deep SORT+YOLO framework in the “Racetrack” scene. Frame ground truth is 19

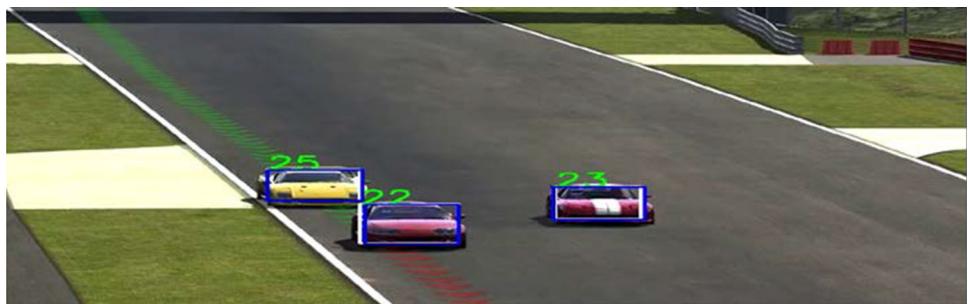


Fig. 4 Results after our modifications in the “Racetrack” scene. Frame ground truth is 19

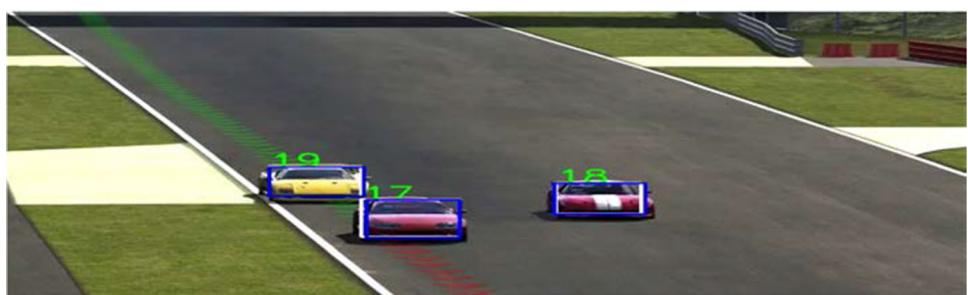


Fig. 5 Non-modified Deep SORT+YOLO framework in the “Rural road dusk” scene. Frame ground truth is 18



Fig. 6 Results after our modifications in the “Rural road dusk” scene. Frame ground truth is 18



bounding box IDs in Fig. 1. The difference is massive, and the main reason for this is the large number of identity switches either from occlusion or poor viewing angle as the detector struggles to maintain accurate detections across all frames. This can result in tracking numbers that, when the original Deep SORT algorithm is used, are quite higher than the ground truth. The modified version correctly displays the confirmed tracks on the video output.

An additional example case is illustrated in Figs. 3 and 4. The example case is in the frame of the “Roadtrack” test scene, where cars are detected and tracked. The grand truth of the example case is 19. In Fig. 3, the performance of the original Deep SORT is off by 6 tracking and giving IDs to 25 different cars. Also, it is worth noting that even in the same frame the non-modified code had already failed to properly ID this stack of cars, as illustrated by the fact that there is no car numbered with ID 24. The modified Deep SORT has a quite better performance which matches the ground truth.

Finally, another example case is presented in Figs. 5 and 6 in our own, real-world test scene named “Rural road dusk.” The ground truth at that part of the scenes is 18. In Fig. 5, the results of the original Deep SORT are off by 20. Although there were 18 different cars in the scenes, the original Deep SORT and YOLO framework resulted in tracking and giving IDs up to 38. In Fig. 6, the results of the modified Deep SORT are presented and we can see that the resulting IDs are very close to the ground and are off by just 1. The YOLO detector works the same way in both cases and the modified Deep SORT performs quite better compared to the original version reporting almost a perfect performance.

As illustrated in the above three example cases, the modified version of the Deep SORT has a quite good performance and resulted in better tracking and consistent annotations of IDs. This is crucial when we create real-time online MOT systems since we can even feed in real-time a live video from a camera and have it display proper IDs and tracking results.

3.2 Detection models

The Deep SORT tracking algorithm needs to be integrated with a multiple-object detection model that will perform the detection of the desired objects in a frame and after that, the Deep SORT will perform the tracking procedure. In the context of our study, we examine the performance of the Deep SORT in the pipeline with YOLO (You Only Look Once) [19]. YOLO has been proven to offer high performance and detection accuracy [35] and the Yolo models used and examined here are (i) the YOLOv3-Tiny, (ii) the YOLOv3-416 and 608 and (iii) the YOLOv4-608.

These models are trained on the MS-COCO and DETRAC datasets, and we use the weights that have been generated by the training on these datasets. The first implementation works with the YOLOv3-Tiny model and weights. In the second implementation, the YOLOv3-416 and 608 model and the corresponding weights are used and lastly, we use YOLOv4-608 with 608-by-608 tensor input to test our framework with state-of-the-art models. These YOLO detection models have been formulated into Keras along with their weights that were generated from their perspective Darknet projects. Darknet [32] is an open-source neural network framework written in C and CUDA, and it supports CPU and GPU computation.

The developers of YOLO reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection. First, YOLO is extremely fast. The frame detection process is looked at as a regression problem which enables YOLO to have a simplified pipeline. They simply run this neural network on a new image at test time to predict detections. This model achieves high throughput, which makes it suitable for process streaming video in real time. Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time, so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects, because it cannot see the larger context. Third, YOLO learns generalizable representations of objects. This network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means that YOLO reasons globally about the full image and all the objects in the image.

3.2.1 YOLOv3-Tiny integration

For the tiny YOLOv3 model, we are using these anchors: [10, 13], [23, 27], [37, 58], [81, 82], [135, 169], [344, 319], which correspond to the size of the bounding boxes and are fundamental for the correct training and detection of our CNN. Moreover, we set anchor mask values of [[3, 4, 5], [0, 1, 2]]. We configure them based on the design and the dimension of the objects we want to detect. To begin with, for this framework we set a score = 0.3 and IOU = 0.2 for our tests in section 7.1 and score = 0.6 and IOU = 0.3 for 7.2 and 7.3. Score is the confidence percentage for the

detection coming out of our CNN. Do keep in mind that YOLOv3-Tiny has only 21 layers and it needs only 5.5 billion flops per frame. For the study, we use a tensor input of (416, 416). YOLOv3-Tiny has a mean average precision (mAP) of 23.7% on the MS COCO dataset.

3.2.2 YOLOv3 integration

For YOLOv3-416 and 608, we are using these anchors: [10, 13], [16, 30], [33, 23], [30, 61], [62, 45], [59, 119], [116, 90], [156, 198] and [373, 326], which correspond to the size of the bounding boxes and are fundamental for the correct training of our CNN. Moreover, we set anchor mask values of [[6, 7, 8], [3, 4, 5], [0, 1, 2]]. They were configured and fine-tune based on the design and the dimension of the objects we want to detect. For this framework, we set a score = 0.4 and IOU = 0.2 for our tests in Sect. 6.1 and score = 0.6 and IOU = 0.3 for 6.2 and 6.3. YOLOv3-608 has only 106 layers and it needs 140 billion flops per frame when having a tensor input of (608, 608) and 65.86 billion flops per frame when at (416, 416). The complexity is much higher, and the increased computation requirements cause a considerable drop in average frame. That being said it achieves significantly better results in the MS COCO dataset having a mAP of 55.3 for tensor input of 416 and 57.9 for tensor input of 608. In our study, we use it with tensor input of 416 and 608.

3.2.3 YOLOv4-608 integration

For YOLOv4-608, we are using the same anchors and mask that we used on YOLOv3-608. For this framework, we set a score = 0.6 and IOU = 0.3 in all of our tests, where the score is the confidence percentage for the detection coming out of our CNN. Notice that we gradually increase our detection thresholds as we continue testing

more complex and higher performing models. With YOLOv4, we now set our tensor input at 608 instead of 416, which we previously did for YOLOv3, and that is because we want to see how the framework will behave when aiming at the best possible detection and feature extraction. YOLOv4 achieves an mAP of 65.7% with an input tensor of (608, 608), which is significantly higher than the previous models.

3.3 Framework optimization

An important part of the proper real-time operation of our framework concerns a set of optimization procedures that were performed. We monitor the functionality of the framework in our systems with help from Intel's VTune software stack. We launch our application, and then we hook VTune to the corresponding process ID of our framework. We detected a bottleneck in the CPU section of our systems indicating that the CPU cannot feed fast enough our GPU, while also being able to perform the necessary calculations for the tracking algorithm provided by Deep SORT. In Fig. 7, we can see that the single-threaded nature of the software on crucial functions causes issues and, if we were to multithread and batch our functions for video pre-processing, it would not meet the criteria for a real-time tracking algorithm, since it does not process every frame as it is created. Looking closer at the graph shown in Fig. 7, we see our primary thread failing to hold steady at 100% CPU time, which is caused by our GPU having to run our CNN on a per-frame basis which causes the CPU to become idle.

The infrastructure used for our experiments is equipped with an NVIDIA GTX 1070 8 GB VRAM paired with 16 gigabytes of RAM and an Intel i7 6700 K. We used the NVIDIA CUDA toolkit version 10.0 and Tensorflow-gpu version 1.14. Keras version 2.2.4 and Python Anaconda

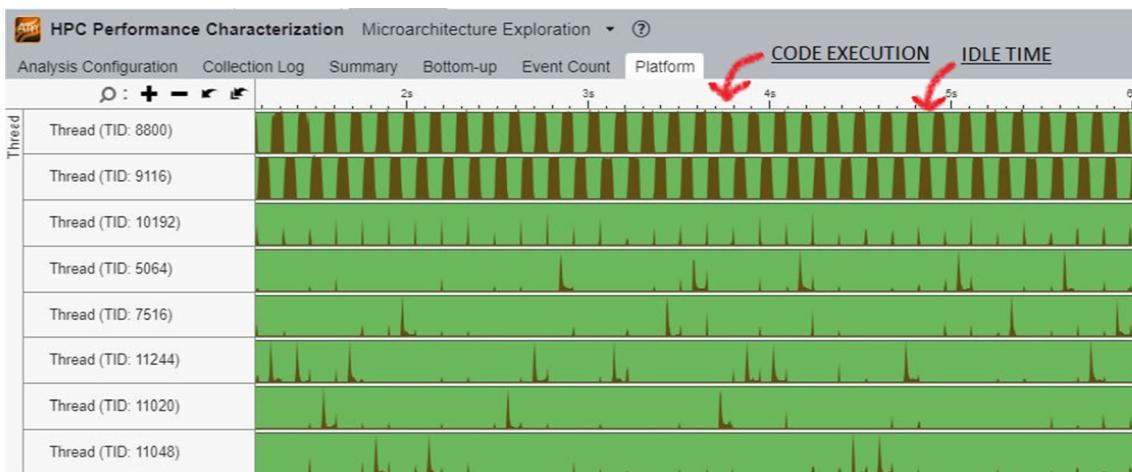


Fig. 7 Spawning threads during execution of our framework using Intel VTune

distribution 2019.03 were the frameworks for the implementations. For the experiments, the processor was set at 4.5 GHz and the graphics card was also locked at 2.1 GHz, while the training and validation data were kept on an NVME drive to alleviate any potential storage bottlenecks.

Each row represents a thread spawned by python for this framework. The CPU first has to preprocess a frame from the video input and then send it to the GPU for the object detection part of the framework. When the GPU is doing calculations, the CPU is idling while waiting for the GPU to send back the result. When the results get sent back to the CPU, it is time for the Deep SORT algorithm to take over and match each bounding box with the correct IDs. This is also executed on the CPU. After this process is completed, the CPU writes back to the frame the output from the model along with the correct IDs. This process is getting repeated until there are no more frames to process.

The rest of the threads remain mostly idle and that is expected behavior, since we have not multithreaded the video pre-processing task or the CPU side tasks of our Deep SORT framework.

Knowing that, by default, python and tensor flow installations are not compiled to make use of more advanced SSE4.1/SSE4.2 and AVX instructions, we try to find ways to improve the performance by using optimized libraries for our system. Initially, Intel's python packages for Numpy, Scipy and others were installed. These packages have improvements mainly from the use of SSE4.2, AVX, AVX2 instructions. However, the performance did not improve so much, because these libraries were not hotspots in our code. After this, we started timing every part of our code and found that the video processing tasks, which were powered by the Pillow library, were taking a big part of our execution time. With the use of VTune to perform HPC profiling of our framework, as shown in Fig. 8, we can see that our primary thread can grow in terms of vectorization. Since we are not bandwidth bound

with just 1.2% of time spent waiting for data from DRAM, we know that our memory subsystem is ready to handle an increase in the data flow. We installed and used the Pillow 6.0.0 SIMD AVX2 package, and we got an improvement that ranged from 10 to 22%. The percentage of the improvement depends on the number of cars detected per frame, the detector used and video input resolution. This improved performance is presented in detail in the results of the experimental study.

In Table 1, we present some results from the tests, where we compare the generic SSE Pillow 6.0 version versus the AVX2 enabled, Pillow version 6.0.

The results indicate that, after the optimization procedures, we have a substantial improvement in the real-time operation and performance of the detection procedure. As illustrated in Table 1, the SIMD optimizations that were introduced on the framework have a substantial impact on the frames per second. We recorded the highest improvement (21.87% in the frames per second) in the crossroad video when running the YOLOv3-Tiny framework. As the input resolution goes up and the time spent for object detection goes down, we will experience more and more performance improvements from this.

4 Experimental study

In this section, we present the experimental study and the results collected. The experiments focus on the examination of the performance of the modified version of Deep SORT tracking algorithm, and its performance is assessed toward the performance of the original version of the algorithm. We examine its performance in a wide range of datasets and integrations with YOLO detectors and assess the performance of the YOLO multiple-object detection models trained on the MS-COCO and the UA-DETRAC datasets using transfer learning on the testing datasets.

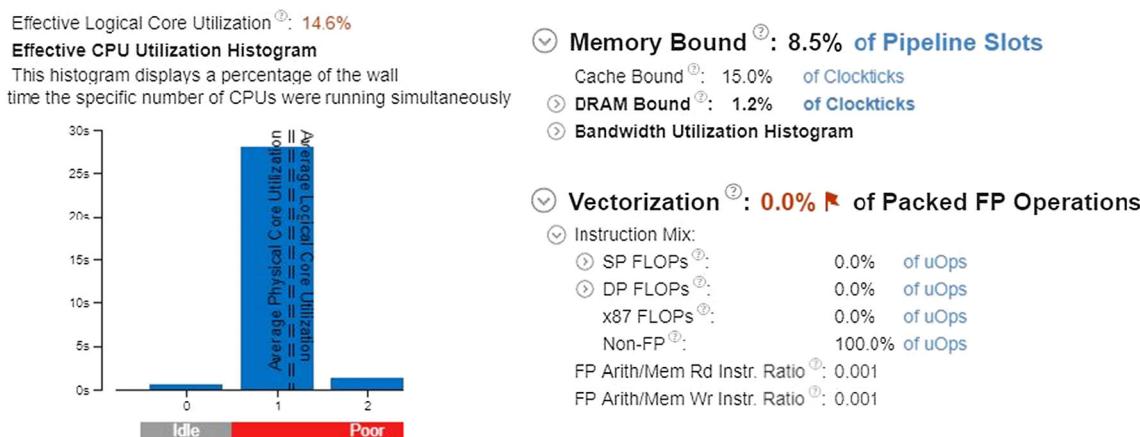


Fig. 8 Gathering performance metrics for our framework using Intel VTune

Table 1 MOT performance comparison

Frames per second	Video resolution	Before SIMD	After SIMD	Improvement %
YOLOv3-Tiny on the crossroad video	1080p	12.8	15.6	21.87
YOLOv3-416 on the crossroad video	1080p	9.2	10.4	13.04
YOLOv4-608 on the crossroad video	1080p	6.7	7.8	16.41
YOLOv3-Tiny on the Racetrack video	1080p	12.8	15	17.18
YOLOv3-416 on the Racetrack video	1080p	9.2	10.5	14.13
YOLOv4-608 on the Racetrack video	1080p	6.7	7.9	17.91
YOLOv3-Tiny on the Straight road 480p video	480p	29.4	33.1	12.58
YOLOv3-416 on the Straight road 480p video	480p	15	16.6	10.66
YOLOv4-608 on the Straight road 480p video	480p	9.9	10.9	10.1

Based on our tests, we found that the UA-DETRAC trained YOLOv3-Tiny and YOLOv4-608 models were able to outperform the MS-COCO ones on average, in terms of execution speed and detection accuracy.

4.1 Datasets used for the training procedure

4.1.1 MS-COCO

The Microsoft Common Objects in COntext (MS-COCO) dataset contains 91 common object categories with 82 of them having more than 5000 labeled instances. In total the dataset has 2,500,000 labeled instances in 328,000 images. Additionally, a critical distinction between this dataset and others is the number of labeled instances per image, which may aid in learning contextual information. MS COCO contains considerably more object instances per image (7.7) as compared to ImageNet (3.0) and PASCAL (2.3). Utilizing over 70,000 working hours, a vast collection of object instances was gathered, annotated and organized to drive the advancement of object detection and segmentation algorithms. Emphasis was placed on finding non-iconic images of objects in natural environments and varied viewpoints. Dataset statistics indicate that the images contain rich contextual information with many objects present per image. We only shortly mention this dataset, because we used the weights created by the YOLO authors that were trained on the MS COCO dataset. We used the pre-trained weights of all the YOLO models trained on the MS-COCO dataset as described in [6]. The models are trained to detect 80 classes, and in the context of our experiments, we used the model's detections for the "car" class.

4.1.2 UA-DETRAC dataset

The UA-DETRAC dataset [12] was created by the University at Albany for comprehensive performance evaluation of MOT systems. The UA-DETRAC dataset

consists of 100 videos, selected from over 10 h of image sequences acquired by a Canon EOS 550D camera at 24 different locations, which represent various traffic patterns and conditions including urban highway, traffic crossings and T-junctions. Notably, to ensure diversity, the creators capture the data at different locations with various illumination conditions and shooting angles. The videos are recorded at 25 frames per second (fps) with the JPEG image resolution of 960×540 pixels. More than 140,000 frames in the UA-DETRAC dataset are annotated with 8250 vehicles, and a total of 1.21 million bounding boxes of vehicles are labeled. Creators asked over 10 domain experts to annotate the collected data for more than two months. They also carried out several rounds of cross-check to ensure high-quality annotations. The UA-DETRAC dataset is divided into training (UA-DETRAC-train) and testing (UA-DETRAC-test) sets, with 60 and 40 sequences, respectively. The creators selected training videos that are taken at different locations from the testing videos, but ensure the training and testing videos have similar traffic conditions and attributes. This setting reduces the chances of detection or tracking methods to over-fit to particular scenarios. The classes four classes are "car," "bus," "van" and "others." The vast majority of the dataset is labeled as "car." In Fig. 9, example cases from the datasets are presented as well as an example case with the corresponding bounding boxes of the objects too.

4.2 Datasets used for testing

In the context of the study, we employ nine scenes for assessing the performance of the methods and our modified version of the Deep SORT. The nine scenes that were used are different from the datasets used for the training procedure of the models. Seven datasets out of the nine concern the tracking of vehicles, and two (MOT16 and MOT20) concern the tracking of pedestrians. For all test scenes used, we have generated ground truth files either from scratch or through existing data. We have created a



Fig. 9 Example training instances from the datasets. On the right diverse example cases with cars are illustrated, while on the right an example case with the corresponding bounding boxes is illustrated

new vehicle dataset from the videos we captured consisting of 7 scenes, 11.025 frames and 25.193 bounding boxes. Lastly, all the bounding boxes coordinates are given in top left, width, height format and we also provide ground truth files in MOT16 format for the “Rural road dusk” scene.

The Crossroad [20] is a publicly available car traffic video, which is 3 min and 31 s long. The base resolution is 1080p with 16:9 aspect ratio, and it is just running at 10 frames per second. This video has been captured from a road traffic camera.

The “Straight road” and “Racetrack” datasets are captured from a racing simulator called Assetto Corsa. The datasets were created by us with a resolution of 1080p and at 60 frames per second. We encode the same file down to 480p and 60 frames per second for further testing. These captures from a video game are utilized to take full control over the test scene and avoid recording artifacts, while simultaneously being able to capture a lossless and high-resolution file. Also, it is worth noting that the “Racetrack” video has higher rates of identity switches, due to the higher occlusion rate from having more cars close to each other on a per-frame basis.

Furthermore, we created two new testing scenes captured by a drone of our team, which we name “Rural road” and “Rural road dusk.” They were created using real-world traffic from a public road. The Rural road video scene concerns a public road on a sunny day. We have filmed approximately 15 min of public road traffic using the DJI Phantom 3 drone at 1080p and 25fps. We cut down the video to approximately 2 min, by taking out the parts where there was no traffic. The video incorporates a balance between cars, large vans and pickup trucks. The “Rural road dusk” video concerns the same public road in dusk under different lighting conditions. Specifically, we recorded the traffic of the same road one hour before dusk. The camera was facing the sun, and the cars were generating shadows on the road. So, the Rural road dusk dataset

scene is of quite higher difficulty. All the datasets are publicly available via the GitHub account of our team.

The MOT16 [21] is a widely used dataset for object tracking procedures. We used the “MOT16-09” scene, which is captured outdoors, facing a sidewalk from a low angle. It is a 30-frames-per-second video at 1080p resolution and has a duration of 18 s. The ground truth for the tracks in this video is 25.

The MOT20 [22] is another widely used dataset for object tracking procedures, and we used the “MOT20-01” scene. The scene is captured indoors in a crowded train station. It is quite challenging scene and comes at a 25-frames-per-second video at 1080p resolution. Its duration is 17 s, and its ground truth for the tracks in this video is 90.

4.3 Results

In this subsection, we present the results of the experimental study. We present the performance results of the methods examined and the modified version of Deep SORT. The experimental results are structured in two parts, the first concerns the performance of the methods, when the Yolo detectors are trained on MS-COCO, and the second when they are trained on the DETRAC dataset.

We rank these frameworks based on the results we get from a wide variety of metrics. First one is the Deep SORT Tracks Initiated metric. The closer this metric is to the ground truth the better the performance of the tracking is. A number greatly higher than ground truth usually shows that the detector struggles to keep track of a certain object across the scene. Every time the detector fails, there is a chance that a new initiated track is created if the object gets detected on future frames. The second one is the modified Deep SORT Count metric, which is the amount of the confirmed tracks for the scene based on the modification performed to take into consideration a set of previous frame detection. Moreover, we provide recall, precision, F1

score and a confusion matrix for the TP, TN, FN metrics to evaluate detector performance along with tracking performance. Lastly, we also provide MOTA and MOTP scores when available. The MOTP metric is the total position error for matched object hypothesis pairs over all frames, averaged by the total number of matches made. It shows the ability of the tracker to estimate precise object positions, independent of its skill at recognizing object configurations, keeping consistent trajectories and more. MOTA accounts for all object configuration errors made by the tracker, false positives, misses, mismatches, over all frames. It gives a very intuitive measure of the tracker’s performance at keeping accurate trajectories, independent of its precision in estimating object positions. To evaluate detector performance, we use a slightly modified version (to include all the metrics we wanted) of the tool used and created by [36], an open-source evaluator.

4.3.1 Results with optimized detection models trained on MS-COCO

Here, we present the results of our framework when the YOLO detectors are trained on MS-COCO. Initially, we present the performance when YOLO3-Tiny is used as detector and after that the performance when YOLO3 and YOLO4 are used.

4.3.1.1 YOLOv3-Tiny In Table 2, the results of the Deep SORT and the modified Deep SORT using YOLOv3-Tiny trained on MS-COCO as detector are presented. A first point concerns the tracking performance in the “Racetrack” video, which is also poor due to the subpar detection performance of YOLOv3-Tiny. Looking at the initiated tracks metric of the Deep SORT (85), we can tell that this detection model consistently failed to hold track of the objects it detected, which is also shown by the high number

of false negatives. The closer this metric is to the ground truth, the better the performance of the tracking is. A number greatly higher than ground truth usually shows that the detector struggles to keep tracking of a certain object across the scenes. Having said that, the Deep SORT algorithm did a decent job at mitigating this issue as seen from the Deep SORT count metric.

It is worth noting that trying to track on a 1080p source only gets us 15fps using this setup. This may not be viable for real-time tracking. Looking at the results for the Straight road at 480p, the average frame rate is 33.1 FPS, and for Racetrack at 480p, it is 35.2 FPS. This indicates that the framework is quite suitable for real-time tracking. In the Crossroad video, massive amounts of identity switches were experienced due to the extremely low frame rate of the source which, in turn, caused big gaps from frame to frame for the bounding boxes. This makes the trajectory estimation algorithm often to fail.

Lastly, in the “Rural road” and “Rural road dusk” videos the results indicate a poor tracking performance due to the poor detection exhibited by YOLOv3-Tiny, as shown by the very poor recall and F1 score. There are many detection failures as pointed out by the significantly higher initiated tracks metric compared to the Deep SORT count for every scene except for the “Straight road” 1080p and 480p. On the Racetrack dataset, where the ground truth was 23 cars, we measured 31 on both resolutions and the tracks initiated for both tests were close to 86 for the 480p video and to 85 when tested at 1080p, which indicates a large amount of detection failures. The tracking of the Modified Deep SORT in the “Straight road” is lower than the ground truth (6 vs 9). This is because the cars in the back are not detected by this YOLO model in time. The tracking for the cars that were detected is excellent.

In Fig. 10, we provide the precision-recall curve for all scenes tested and in Figs. 11, 12, 13 and 14, example

Table 2 MOT results on the YOLOv3-Tiny-enabled framework

YOLOv3-Tiny	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
Crossroad-Init = 4	15.6	263	104	92	0.944	0.375	0.537	4211	246	6990
Crossroad-Init = 7	15.6	269	80	92	0.944	0.375	0.537	4211	246	6990
Straight road 1080p	15.1	10	6	9	1	0.436	0.608	187	0	241
Straight road 480p	33.1	12	6	9	1	0.483	0.651	207	0	221
Racetrack	15	85	31	23	0.997	0.801	0.889	3486	8	862
Racetrack 480p	35.2	86	31	23	0.996	0.828	0.905	3604	12	744
Rural road	15.1	220	56	44	0.99	0.447	0.616	1321	13	1632
Rural road dusk	16	102	18	24	0.965	0.4	0.565	595	21	892

Fig. 10 Precision–recall curve for YOLOv3-Tiny

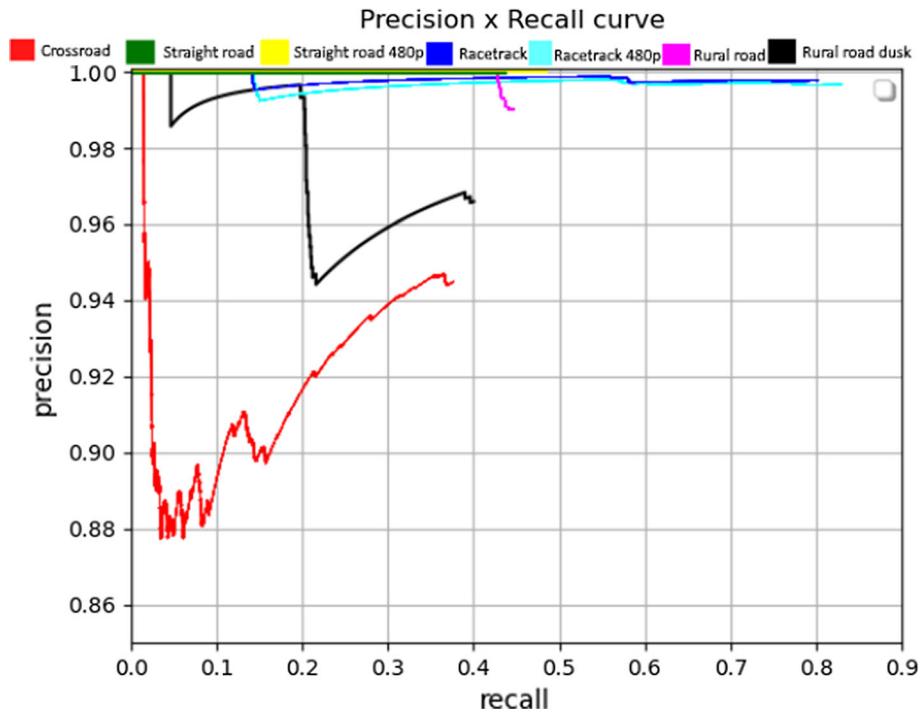
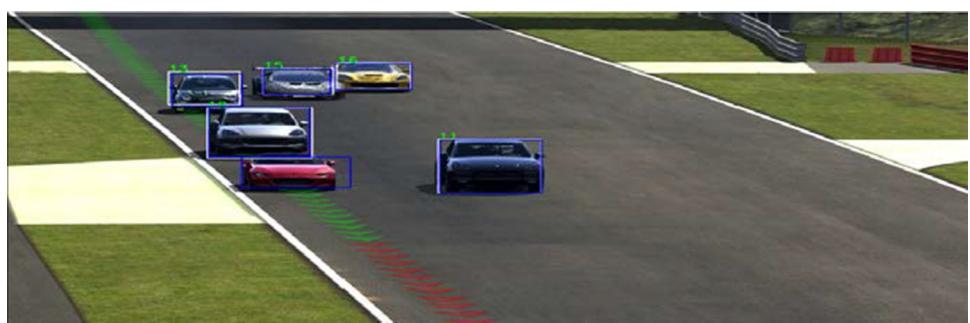


Fig. 11 YOLOv3-Tiny-enabled framework on the crossroad video



Fig. 12 YOLOv3-Tiny-enabled framework on the Racetrack video



detection frames from the YOLOv3-Tiny framework are illustrated. As seen in Fig. 11, YOLOv3-Tiny has trouble detecting the cars in the distance, something that results in numbering errors by a considerable amount. In Fig. 12, even with the modified Deep SORT algorithm now properly displaying the tracks, we still see skipped Ids, which is

caused by the red car in the front which YOLOv3-Tiny has trouble detecting consistently. This causes issues to the Deep SORT to mark it as a confirmed track. In Figs. 13 and 14, the same problem is illustrated. The cars in the distance at the back cannot be properly detected and the car numbered “2” has been eliminated, because of the excessive

Fig. 13 YOLOv3-Tiny-enabled framework on the Straight road 1080p video

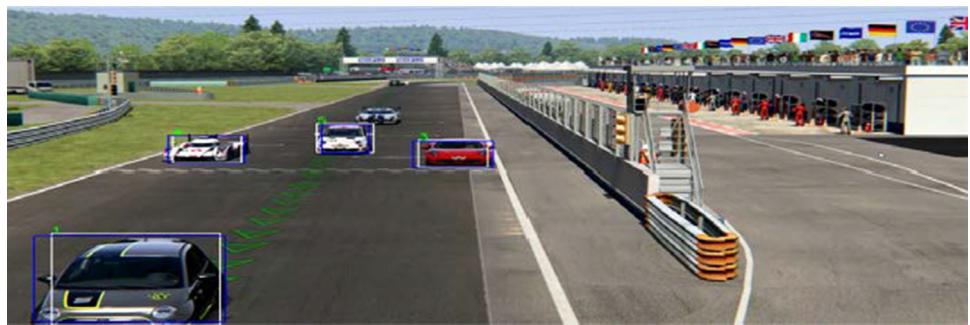


Fig. 14 YOLOv3-Tiny-enabled framework on the Straight road 480p video

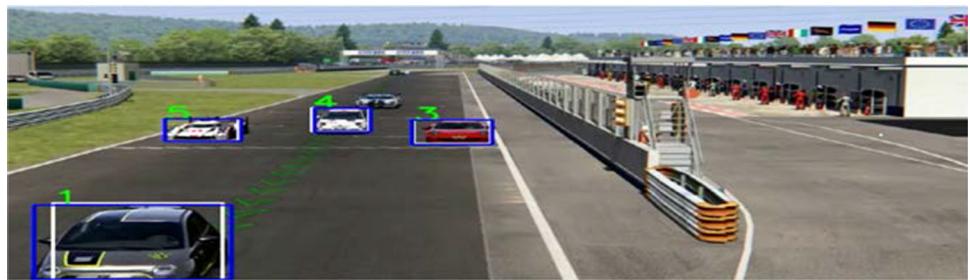


Table 3 MOT results on the YOLOv3-416-enabled framework

YOLOv3-416	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
Crossroad-Init = 4	10.3	411	150	92	0.892	0.823	0.856	9226	1110	1975
Crossroad-Init = 7	10.4	451	103	92	0.892	0.823	0.856	9226	1110	1975
Straight road 1080p	10.3	10	8	9	0.876	0.759	0.813	325	46	103
Straight road 480p	16.6	10	7	9	0.829	0.626	0.713	268	55	160
Racetrack	10.5	35	24	23	0.985	0.919	0.951	3999	58	349
Racetrack 480p	16.9	46	24	23	0.988	0.893	0.938	3887	47	461
Rural road	10.6	83	45	44	0.877	0.894	0.885	2640	368	313
Rural road dusk	10.9	58	23	24	0.937	0.802	0.865	1194	79	293

identity switches. We can also notice that the lower video input resolution did not affect the detection process.

4.3.1.2 YOLOv3-416 In Table 3, the results of the Deep SORT and the modified Deep SORT using YOLOv3-416 as detector are presented. The results show that the detection performance on the “Straight road” video and on the more complex “Racetrack” video is significantly better than YOLOv3-Tiny. The increased detection performance as seen by the recall and F1 score metrics, allows the Deep SORT framework to perform even better. The performance in the “Crossroad” video is low, mainly because of the low-resolution and frame rate video captured by the CCTV. The results also point out that now we experience more

identity switches, as seen from the tracks initiated by Deep SORT (411 and 451, respectively). The reason for this is the fact that YOLOv3-416 is better at detecting hard-to-see cars compared to YOLOv3-Tiny. The results also show that the modified Deep SORT algorithm performed quite well and made a quite good tracking, counting 150 (vs 411) and 103 (vs 451) cars, respectively. On the Racetrack scene, we noticed significantly less initiated objects, because this scene has a clear view of the cars. This allowed the much-improved YOLOv3-416 to keep track of the initiated objects. We now also notice near perfect performance in the Rural road videos, which is attributed to the much better detection performance of YOLOv3-416 over YOLOv3-Tiny. The Deep SORT count is off by 1

compared to ground truth and the Initiated tracks are significantly lower compared to YOLOv3-Tiny.

In Fig. 15, we provide the precision–recall curve for all scenes tested and in Figs. 16, 17, 18 and 19, example detection frames from the YOLOv3-416 framework are illustrated. In Fig. 16, we can see that YOLOv3-416 can now detect cars that are far at the back distance. In Figs. 17 and 18, we see the same; the cars at the back distance are now detected and tracked properly. However, we still experience an identity switch even with this improved detection performance on both video inputs. We can also notice that the lower video input resolution did not greatly

affect the detection process, since we only saw a tiny increase in detection performance for the cars that were furthest away. Finally, in Fig. 19, we can see proper detection and tracking performance for this part of the test. The increased accuracy of YOLOv3-416 over YOLO-Tiny is noticeable and provided better tracking performance as illustrated above.

4.3.1.3 YOLOv4 In Table 4, the results of the Deep SORT and the modified Deep SORT using YOLOv4-608 detector are presented. We again see that the detection performance on the “Straight road” video is good and the performance

Fig. 15 Precision–recall curve for YOLOv3-416

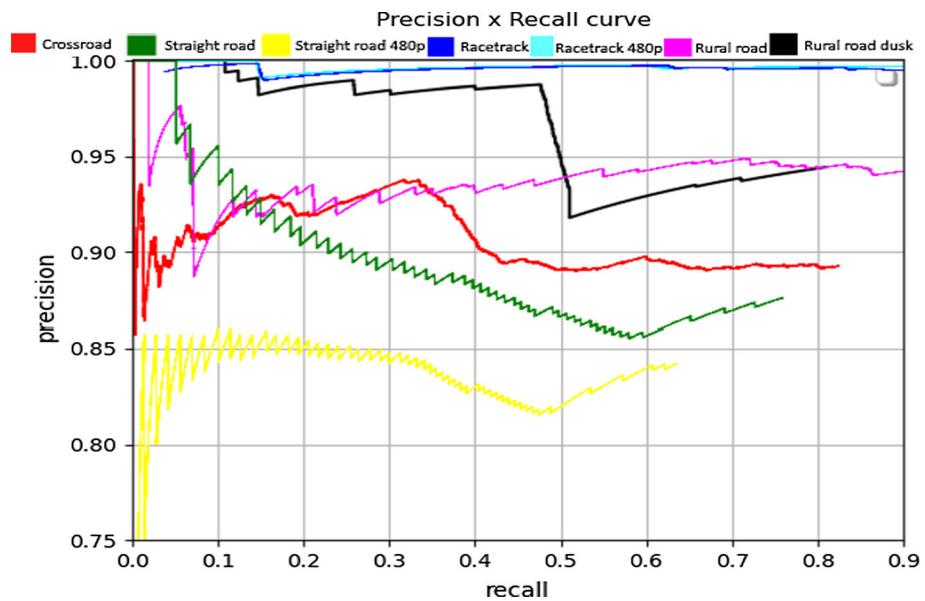


Fig. 16 YOLOv3-416-enabled framework on the crossroad video



Fig. 17 YOLOv3-416-enabled framework on the Straight road 1080p video

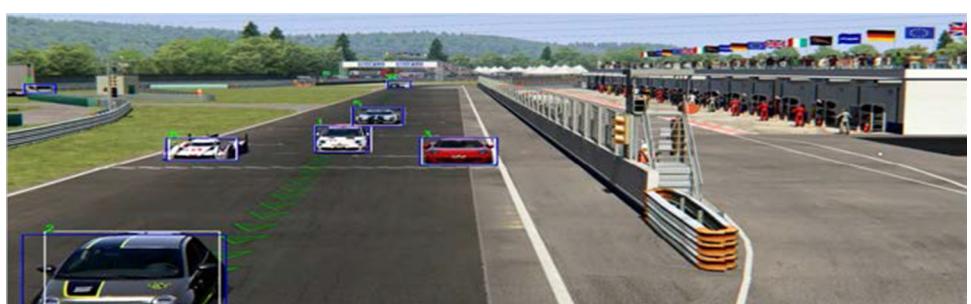


Fig. 18 YOLOv3-416-enabled framework on the Straight road 480p video

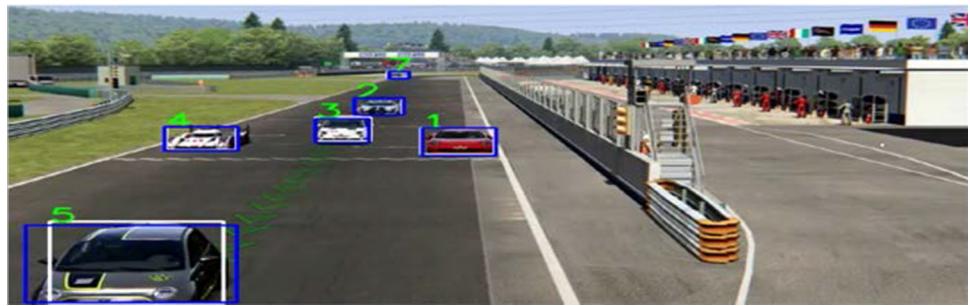


Fig. 19 YOLOv3-416-enabled framework on the Racetrack video

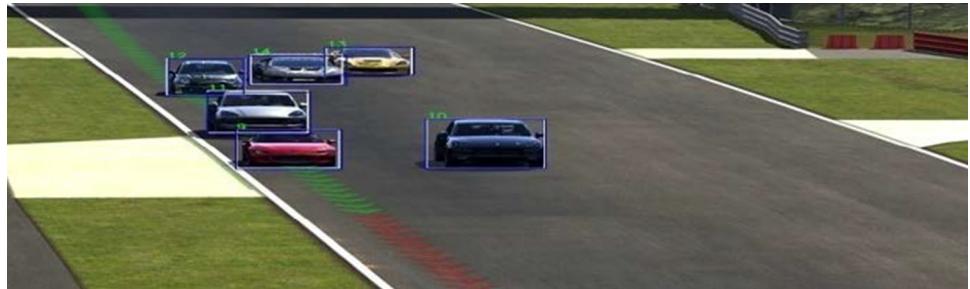


Table 4 MOT results on the YOLOv4-608-enabled framework

YOLOv4-608	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
Crossroad-Init = 4	7.8	396	91	92	0.939	0.8	0.864	8966	578	2235
Crossroad-Init = 7	7.8	434	66	92	0.939	0.8	0.864	8966	578	2235
Straight road 1080p	7.7	20	10	9	1	0.899	0.947	385	0	43
Straight road 480p	10.9	12	7	9	0.996	0.719	0.835	308	1	120
Racetrack	7.9	44	24	23	0.997	0.973	0.985	4234	10	114
Racetrack 480p	11.3	37	25	23	0.994	0.965	0.98	4200	22	148
Rural road	7.8	90	48	44	0.959	0.92	0.939	2718	115	235
Rural road dusk	8.1	59	24	24	0.952	0.841	0.893	1252	63	235

on the more complex “Racetrack” video is significantly better than using YOLOv3-Tiny and roughly equal to YOLOv3-416. The performance in the “Crossroad” video is good enough, when we use an *n_init* value of 4, something that is necessary because of the low-resolution and the frame rate of the video captured by the CCTV. It’s the only way to track most of the cars, since a lot of them are only visible for less than 7 frames. The increased performance of YOLOv4-608 is now visible in this instance. Lastly, it is worth noting that YOLOv4-608 is noticeable slower than YOLOv3-416, but not by a large amount. The increased detection performance is good and worth the cost of a few FPS, as we witness an uplift in all detection performance metrics compared to YOLOv3-416. Looking at the tracks initiated by the Deep SORT, we can once

more see a lot more initiated tracks than expected on the crossroad video, which is caused mainly by the poor video quality. A small problem we noticed with YOLOv4 is that it sometimes detected cars at places where there were none. That happened for only one frame, so the Deep SORT algorithm was able to exclude that result without trouble. It is worth noting that now we have a tensor input resolution of (608, 608) so, drops in the detection accuracy are more noticeable on the 480p videos. The results on the “Straight road 480p” and “Racetrack 480p” indicate a drop in the detection performance in both cases. Performance in the Rural road videos remains good and significantly better than YOLOv3-Tiny.

In Fig. 20, we provide the precision–recall curve for all scenes tested and in Figs. 21, 22, 23 and 24, we present

Fig. 20 Precision–recall curve for YOLOv4-608

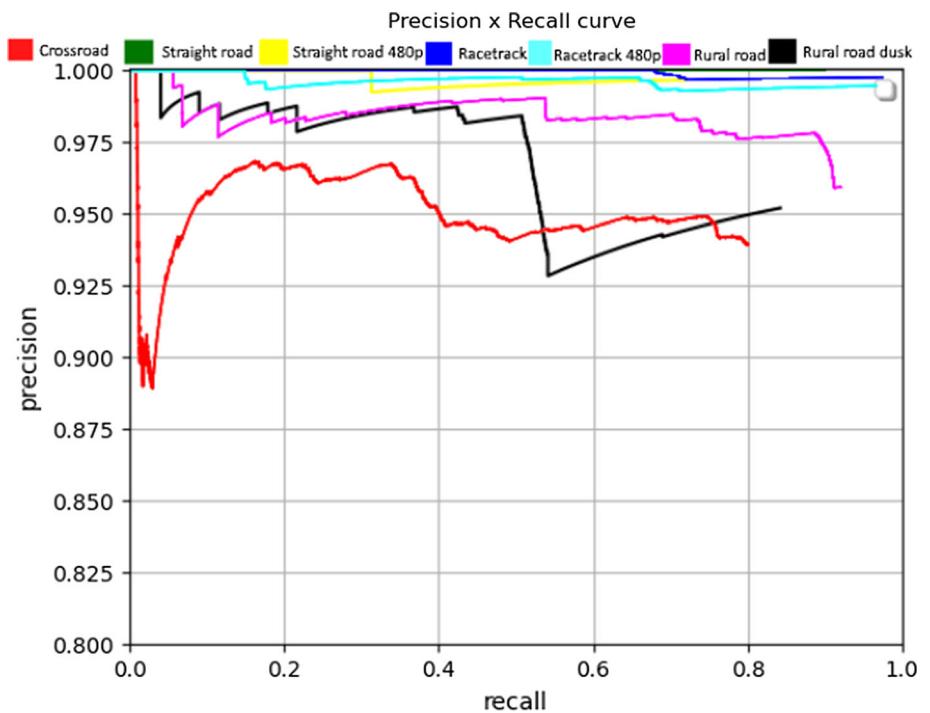


Fig. 21 YOLOv4-608-enabled framework on the crossroad video



Fig. 22 YOLOv4-608-enabled framework on the Straight road 1080p video



Fig. 23 YOLOv4-608-enabled framework on the Straight road 480p video



Fig. 24 YOLOv4-608-enabled framework on the Racetrack video



Table 5 MOT comparison on the YOLOv4-608-enabled framework using the “Racetrack 480p” video

YOLOv4	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	11.3	37	25	23	0.994	0.965	0.98	4200	22	148
UA-DETRAC	12.4	31	23	23	0.999	0.953	0.975	4144	3	204

example cases from the detection frames of YOLOv4-608. As seen in Fig. 21, YOLOv4-608 can now detect cars that are far away and we can keep tracking them without having to be close to the camera. In Figs. 22 and 23, we do see good tracking. However, at that point of the video, the cars further away cannot be properly identified. Once the cars get closer to the camera, the detection and the tracking are excellent. We can also note that this time, the lower video input resolution affected the detection process a bit more than previously and this is mainly caused by the fact that we have a tensor input of (616, 616) and the vertical resolution of the video was 480 pixels. Small cars were hard to be properly detected from the reduced vector information. Finally, in Fig. 24, we once more see good tracking and detection performance by YOLOv4-608.

4.3.2 Results with optimized detection models trained on UA-DETRAC

In this part of the experimental study, we examine the performance of the modified Deep SORT when integrated with YOLO detectors that are trained on UA-DETRAC dataset. In the Racetrack and Rural road videos we use an $n_{init} = 7$ and on the crossroads video $n_{init} = 4$ due to the lower frame rates of the video.

4.3.2.1 Results on YOLOv4 In Table 5, the results of the Deep SORT and the modified Deep SORT using YOLOv4 trained on UA-DETRAC as detector are presented. For the training procedure, we measured an average loss of 1.583 and a mAP of 98.68%. The results are quite good and can facilitate the good performance of the Deep SORT framework.

Starting with the “Racetrack 480p” scene, we achieve perfect numbering and tracking across the whole test scene

with our YOLOv4 detector trained on the UA-DETRAC dataset. Also, we notice an increase in the execution performance which is approximately 10% as seen in Table 5. This is due to the simplification of the YOLOv4 network, since we train it on just one class.

We now pay attention to the “Tracks Initiated” metric and now see that YOLOv4-608 works best on this test scene when trained on the UA-DETRAC dataset with a perfect Deep SORT count and much lower initiated tracks compared to the MS-COCO one, which failed to track well during a mild occlusion phase and had trouble detecting some of the vehicles.

In Fig. 25, we provide the precision–recall curve and in Fig. 26 we now see perfect numbering and tracking of all cars in that particular frame. All other detectors we tested failed to achieve this performance.

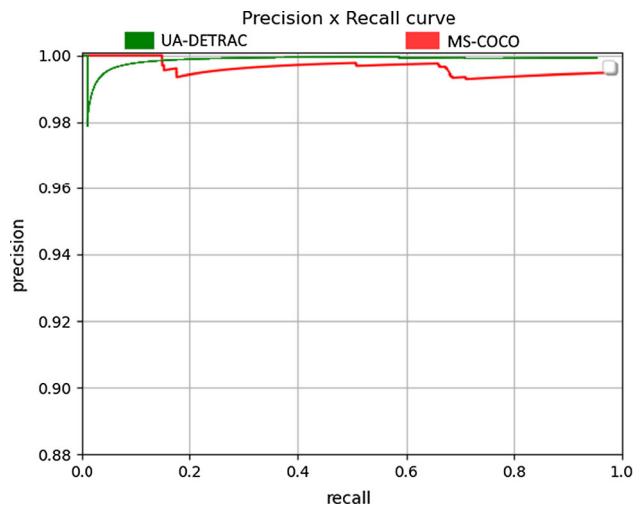


Fig. 25 Precision–recall curve for YOLOv4-608 comparison on the Racetrack 480p video

Fig. 26 YOLOv4-608 UA-DETRAC-enabled framework on the Racetrack 480p video

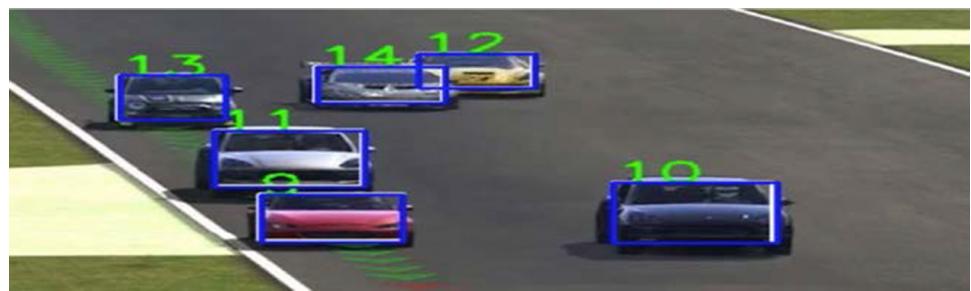


Table 6 MOT comparison on the YOLOv4-608-enabled framework using the “Crossroad” video

YOLOv4	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	7.8	396	91	92	0.939	0.8	0.864	8966	578	2235
UA-DETRAC	8.5	291	87	32	0.997	0.694	0.818	7779	23	3422

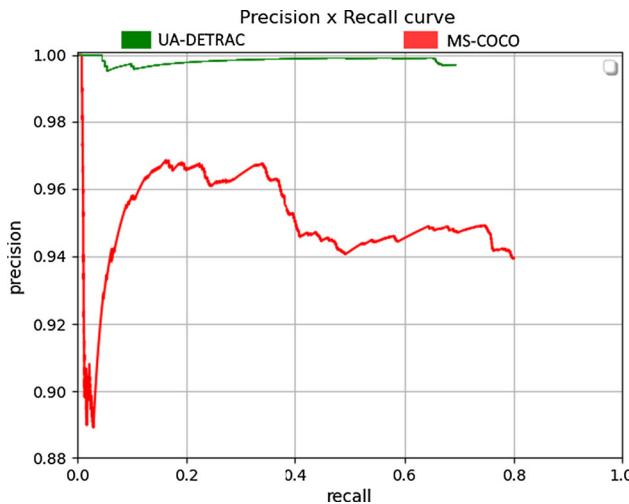


Fig. 27 Precision–recall curve for YOLOv4-608 comparison on the crossroad video

In Table 6, we see that in the “crossroad” video we have an increase in the performance of approximately 10%. The results show that the UA-DETRAC trained YOLOv4 detector reports better performance compared to the MS-COCO one. While the initiated tracks are significantly closer to the real counts, we do see a worse Deep SORT

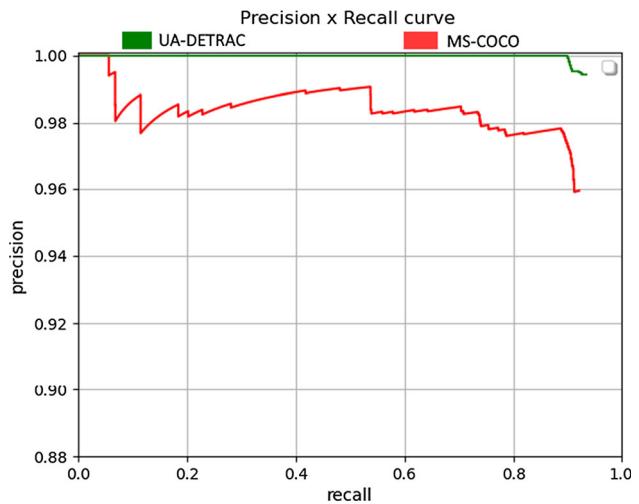


Fig. 28 Precision–recall curve for YOLOv4-608 comparison on the Rural road video

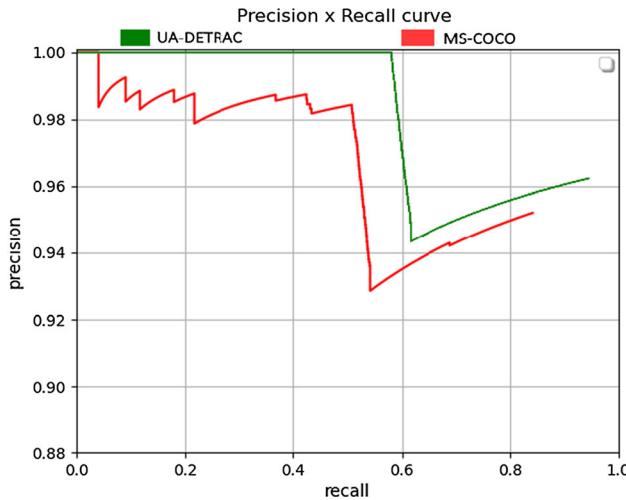
count for the UA-DETRAC trained YOLOv4. We notice a much better tracking process and much higher detection accuracy of big trucks and vans compared to the MS-COCO trained YOLOv4. A major benefit of the UA-DETRAC dataset is the wide variety of vehicles that is included. The MS-COCO dataset is lacking in that

Table 7 MOT comparison on the YOLOv4-608-enabled framework using the “Rural road” video

YOLOv4	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	7.8	90	48	44	0.959	0.92	0.939	2718	115	235
UA-DETRAC	8.4	82	46	44	0.994	0.935	0.963	2762	16	191

Table 8 MOT comparison on the YOLOv4-608-enabled framework using the “Rural road dusk” video

YOLOv4	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	8.1	59	24	24	0.952	0.841	0.893	1252	63	235
UA-DETRAC	8.7	48	25	24	0.962	0.944	0.953	1405	55	82

**Fig. 29** Precision–recall curve for YOLOv4-608 comparison on the Rural road dusk video

department and its inability to properly train our models in detecting big vehicles became apparent in this test scene. Having said that, we did notice that the UA-DETRAC trained YOLOv4 performed worse at detecting cars that are further away, and this is also shown in the false negative metric. Lastly, in Fig. 27 the precision–recall curve is provided.

In Table 7, we notice an increase in execution performance of approximately 10% on the “Rural road” video. The UA-DETRAC-trained YOLOv4 once more showed better performance compared to the MS-COCO one, as seen by all performance metrics we offer including Fig. 28.

**Fig. 30** UA-DETRAC on the Rural road dusk video**Fig. 31** MS-COCO on the Rural road dusk video**Table 9** MOT metrics on the YOLOv4-608-enabled framework using the “Rural road dusk” video

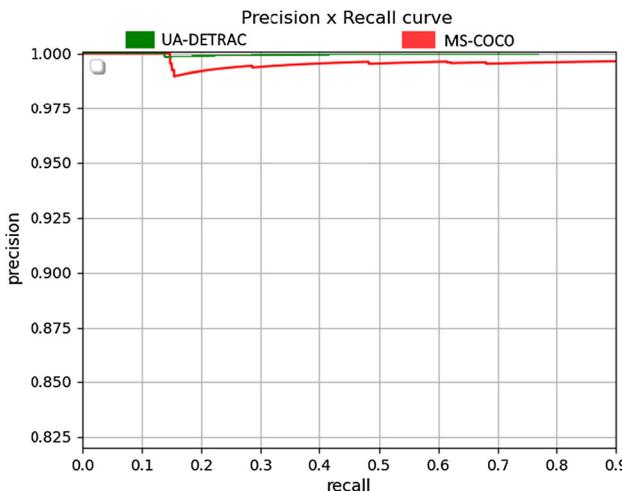
YOLOv4	MOTA	MOTP
MS-COCO	58.4	81.1
UA-DETRAC	71.1	85.7

While the initiated tracks are significantly closer to the real numbers, we also see better results in the modified Deep SORT count metric. This test scene is considered of medium difficulty. We did notice much better tracking and much higher detection accuracy of big trucks and vans compared to the MS-COCO trained YOLOv4. Also, this test has few frames where heavy occlusions take place. The MS-COCO model exhibited a bit worse detection performance, which caused a few more identity switches during easy parts of the scene.

In Table 8, the results show an increase in execution performance of 7%. The UA-DETRAC-trained YOLOv4 showed once more better performance compared to the MS-COCO one. While the initiated tracks are significantly closer to the real numbers, we see that the Deep SORT count is off by just one for our custom framework, while the MS-COCO one is excellent. The results show better tracking and much higher detection accuracy compared to the MS-COCO-trained YOLOv4. The MS-COCO model exhibited a bit worse detection performance, which caused a few more identity switches during easy parts of the scene (Fig. 29).

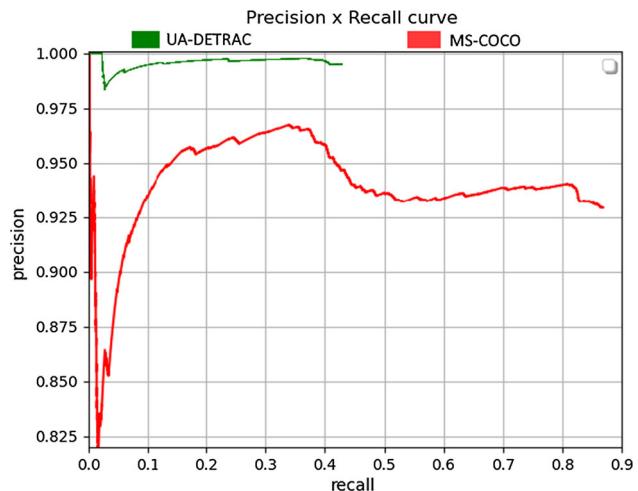
Table 10 MOT comparison on the YOLOv3-608-enabled framework using the “Racetrack 480p” video

YOLOv3	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	12.4	55	24	23	0.996	0.921	0.957	4005	14	343
UA-DETRAC	14.1	58	23	23	0.999	0.77	0.869	3348	1	1000

**Fig. 32** Precision–recall curve for YOLOv3-608 comparison on the Racetrack 480p video

The UA-DETRAC framework measured one above ground truth, because once more there was a car carrying a trailer in the video, which both models detected as a vehicle. The reason why MS-COCO managed to match ground truth is shown in Figs. 30 and 31. The MS-COCO framework completely failed to detect the truck shown in the pictures. As seen in Table 9, the UA-DETRAC-enabled YOLOv4 scored significantly better in the MOTA and MOTP metrics.

4.3.2.2 Results on YOLOv3 In Table 10, the results of the Deep SORT and the modified Deep SORT using YOLOv3 trained on UA-DETRAC as detector are presented. For the training procedure on the UA-DETRAC dataset, the tensor input was set at 416 for 8000 batches and we measured an average loss of 0.823 and a mAP of 96.31%. The results show an increase in the execution performance, which is

**Fig. 33** Precision–recall curve for YOLOv3-608 comparison on the crossroad video

approximately 15% on the “Racetrack 480p” video. This is due to the same reasons we described in YOLOv4. We again notice that the UA-DETRAC-enabled YOLO is better in this test scene compared to the MS-COCO one, which had a small mishap. Deep SORT count is now perfect, since we once more achieved perfect tracking. However, YOLOv3, due to its worse overall mAP performance, exhibits more initiated tracks compared to YOLOv4. Looking at the detection metrics, we can tell that the UA-DETRAC trained YOLO had trouble in detecting the cars at times, but what made it score better in tracking was the detection consistency, once detection occurred. We also provide the precision–recall curve in Fig. 32.

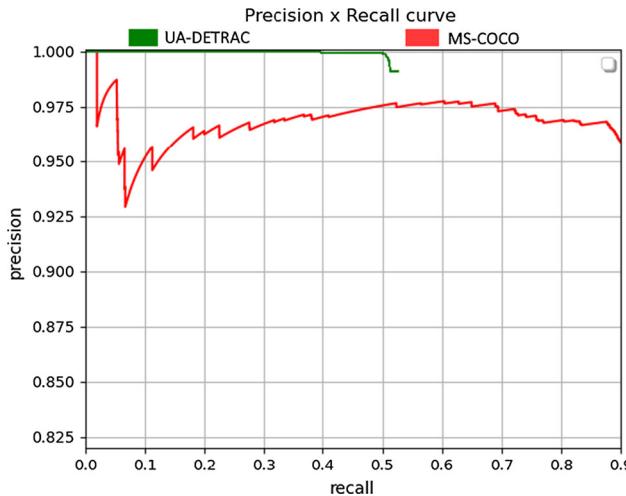
In Table 11, we notice an increase in the execution performance, which is approximately 15% on the “Crossroad” video. The UA-DETRAC-trained YOLOv3, while providing a Deep SORT count closer to ground truth

Table 11 MOT comparison on the YOLOv3-608-enabled framework using the “Crossroad” video

YOLOv3	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	8.2	312	108	92	0.929	0.868	0.897	9729	740	1472
UA-DETRAC	9.5	217	89	92	0.995	0.429	0.599	4807	24	6394

Table 12 MOT comparison on the YOLOv3-608-enabled framework using the “Rural road” video

YOLOv3	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	8.4	93	48	44	0.952	0.911	0.931	2692	133	261
UA-DETRAC	9.3	120	46	44	0.991	0.527	0.688	1557	14	1396

**Fig. 34** Precision–recall curve for YOLOv3-608 comparison on the Rural road video**Fig. 35** MS-COCO framework on the Rural road video**Fig. 36** UA-DETRAC framework on the Rural road video

compared to the MS-COCO one, does have significantly worse detection performance during this test, as seen by the precision, recall and F1 score metrics (Fig. 33). This test scene is considered of high difficulty due to the extremely low frame rate and video noise.

Moving on to the “Rural road” video which is now tested using YOLOv3, in Table 12, we now notice an increase in the execution performance of roughly 10%. Also, we notice once again that the UA-DETRAC-enabled YOLO detector has a worse performance in this test scene compared to the MS-COCO trained one, but was able consecutively detect vehicles better, once detection started to occur. Deep SORT tracking counts are closer to ground truth, but we again see more initiated tracks compared to YOLOv4, which in this case confirms that misses a lot of detections, which is also shown in Fig. 34. Lastly, we notice slightly higher frame rates compared to the YOLOv4 detectors.

In both UA-DETRAC-trained models, we noticed much more consistent detections of trucks, buses and vans. To prove our point, we attach two screenshots from the detection output. In Fig. 35, the MS-COCO trained model fails to detect the vans, while in Fig. 36, the UA-DETRAC model provides consistent tracking of them.

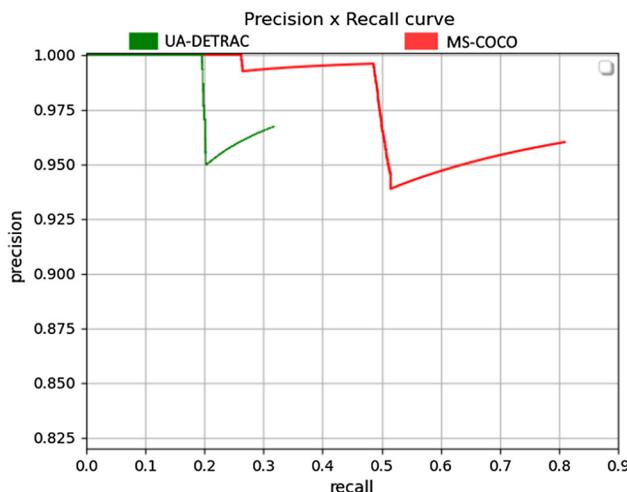
Lastly, in Table 13, the results in the Rural road dusk video are presented. We notice an increase in execution performance of roughly 15%. The UA-DETRAC-trained YOLOv3 detector showed worse performance compared to the MS-COCO one. YOLOv3 exhibited more initiated tracks and counted significantly less cars compared to YOLOv4. This time we notice a better overall tracking when using the MS-COCO trained YOLO even though the UA-DETRAC YOLO could detect trucks better and all metrics show this. The UA-DETRAC model had worse detection performance which caused more identity switches even during easy parts of the scene. The lightning conditions of this scene proved troublesome for the UA-DETRAC dataset as also seen by the precision-recall curve (Fig. 37). In Table 14 we can also see the significantly better MOTA and MOTP scores the MS-COCO-trained YOLOv3 had compared to the UA-DETRAC one.

Table 13 MOT comparison on the YOLOv3-608-enabled framework using the “Rural road dusk” video

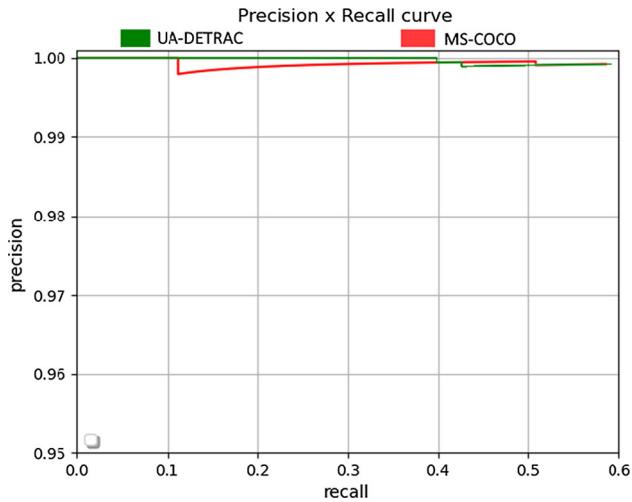
YOLOv3	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	8.7	57	23	24	0.96	0.809	0.878	1204	50	283
UA-DETRAC	9.7	64	18	24	0.967	0.317	0.477	472	16	1015

Table 14 MOT metrics on the YOLOv3-608-enabled framework using the “Rural road dusk” video

YOLOv3	MOTA	MOTP
MS-COCO	59.1	71
UA-DETRAC	19.4	67.6

**Fig. 37** Precision–recall curve for YOLOv3-608 comparison on the Rural road dusk video

4.3.2.3 Results on YOLOv3-Tiny In Table 15, the results of the Deep SORT and the modified Deep SORT using YOLOv3-trained on UA-DETRAC as detector are presented. During the training, we measured an average loss of 0.762 and a mAP of 96.32%. This time during testing we keep the model size at (416, 416) instead of (608, 608) to further increase throughput.

**Fig. 38** Precision–recall curve for YOLOv3-Tiny comparison on the Racetrack 480p video

The results show a great increase in the execution performance. The UA-DETRAC-powered YOLOv3-Tiny is approximately 50% faster when tested on the “Racetrack” video. We notice the UA-DETRAC-enabled YOLO is better in this test scene compared to the MS-COCO one, regarding the Track-initiated metric, but the modified Deep SORT count of the MS-COCO one is closer to the ground truth. Overall, the detection performance of the UA-DETRAC was slightly better as shown by the recall, F1 score and Fig. 38.

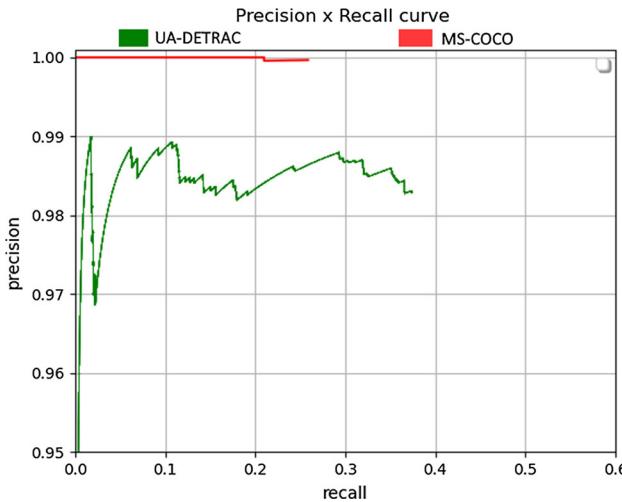
In Table 16, the results show that we have a great increase in the execution performance, which is approximately 15% on the “Crossroad” video. The UA-DETRAC-trained YOLOv3-Tiny is significantly better compared to the MS-COCO one and all metrics show this. The improved tracking performance is attributed to the

Table 15 MOT comparison on the YOLOv3-Tiny-enabled framework using the “Racetrack 480p” video

YOLOv3-Tiny	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	35.7	136	29	23	0.999	0.585	0.738	2544	2	1804
UA-DETRAC	54.3	99	31	23	0.999	0.591	0.743	2572	2	1776

Table 16 MOT comparison on the YOLOv3-Tiny-enabled framework using the “Crossroad” video

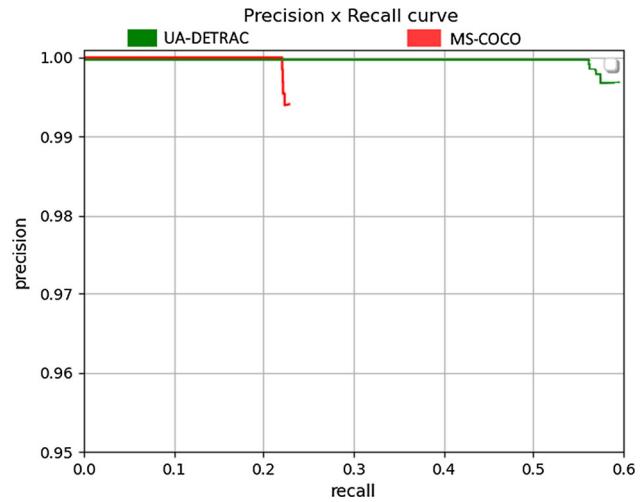
YOLOv3-Tiny	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	15.7	257	97	92	0.999	0.258	0.41	2895	1	8306
UA-DETRAC	18	225	90	92	0.982	0.374	0.542	4193	73	7008

**Fig. 39** Precision–recall curve for YOLOv3-Tiny comparison on the crossroad video

improved detection performance as seen by the significantly better F1 score and precision-recall curve (Fig. 39).

Moving on to the “Rural road” video, we again see an increase in execution performance of roughly 15%. We again notice that the UA-DETRAC-enabled YOLO is better in this test scene compared to the MS-COCO one, by a large margin, as indicated by the detection performance metrics. In Table 17, the results show that both models had a poor performance in this test, given how far they are from the ground truth. YOLOv3-Tiny failed numerous times and even its execution performance is nearly doubled, the loss in accuracy in this test is substantial (Fig. 40).

Lastly, the results in the Rural road dusk video are presented in Table 18. We notice an increase in execution performance of roughly 15%. This time the UA-DETRAC-trained YOLOv3-Tiny showed better results compared to

**Fig. 40** Precision–recall curve for YOLOv3-Tiny comparison on the Rural road video

the MS-COCO. The MS-COCO-trained YOLOv3-Tiny failed consistently to keep track of cars, and this is shown by the modified Deep SORT count metric. The UA-DETRAC model had much better detection performance, although still poor, but that helped the Deep SORT count metric to be closer to ground truth. Tracking performance remains significantly worse, when compared to YOLOv4, as seen from the high tracks-initiated metric at 73 and it is clear that YOLOv3-Tiny is not suitable for accurate tracking and numbering of cars. Here, we can also see the improvement in tracking performance through the MOTA and MOTP metrics in Table 19. While there is an improvement, it is once more obvious that tracking was poor as is also seen in Fig. 41.

In conclusion, the use of the UA-DETRAC dataset assisted in creating an overall better framework for car

Table 17 MOT comparison on the YOLOv3-Tiny-enabled framework using the “Rural road” video

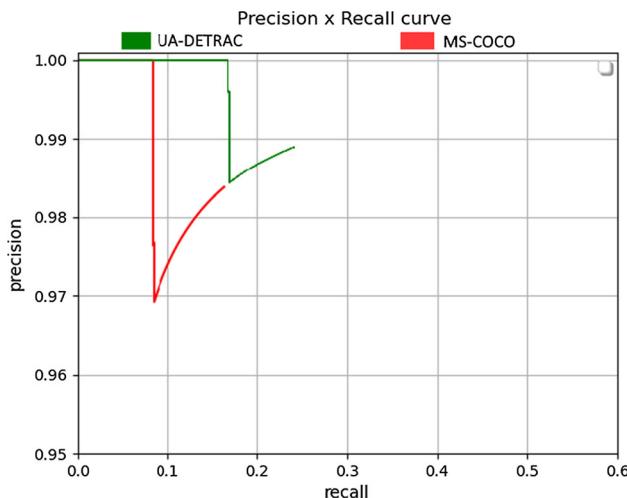
YOLOv3-Tiny	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	15.2	211	24	44	0.994	0.228	0.371	675	4	2278
UA-DETRAC	17.4	172	64	44	0.997	0.594	0.745	1757	5	1196

Table 18 MOT comparison on the YOLOv3-Tiny-enabled framework using the “Rural road dusk” video

YOLOv3-Tiny	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
MS-COCO	16.1	114	6	24	0.983	0.163	0.28	243	4	1244
UA-DETRAC	18.8	73	20	24	0.988	0.241	0.388	359	4	1128

Table 19 MOT metrics on the YOLOv3-Tiny-enabled framework using the “Rural road dusk” video

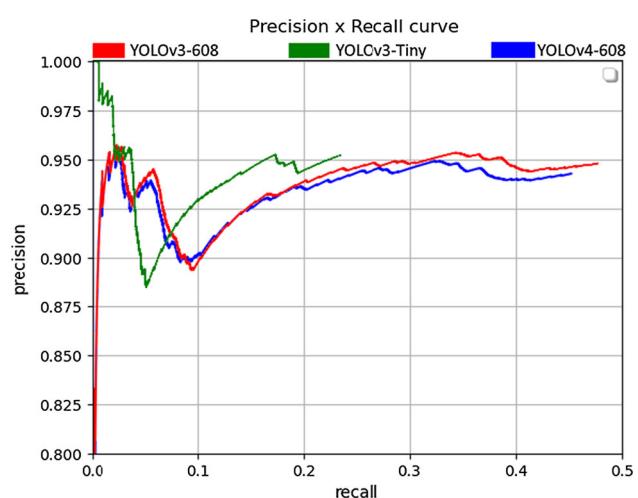
YOLOv3-Tiny	MOTA	MOTP
MS-COCO	3.7	67.6
UA-DETRAC	10.8	49.7

**Fig. 41** Precision–recall curve for YOLOv3-Tiny comparison on the Rural road dusk video

traffic, especially when YOLOv4 is used as detector. The framework reports the best tracking performance, while also being slightly faster compared to its MS-COCO counterpart.

Table 21 MOT metrics on the MOT16 scene

MOT16-09	MOTA	MOTP
YOLOv3-Tiny	37.2	76.1
YOLOv3-608	57.1	78.8
YOLOv4-608	42.3	61.9

**Fig. 42** Precision–recall curve comparison on the MOT16-09 video

4.3.3 Exploring the modified deep SORT on pedestrian videos

The second part of the experimental study concerned the evaluation of our framework and the modified Deep SORT in pedestrian videos and scenarios. In the context of this

Table 20 MOT results on the MOT16 scene

MOT16-09	Frames per second	Deep SORT Tracks Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
YOLOv3-Tiny	15	52	14	25	0.952	0.234	0.375	2025	101	6621
YOLOv3-608	8	81	22	25	0.948	0.476	0.634	4122	225	4524
YOLOv4-608	7.6	72	30	25	0.943	0.452	0.611	3909	236	4737

Fig. 43 YOLOv3-Tiny-enabled framework on the MOT16-09 scene



Fig. 44 YOLOv3-608-enabled framework on the MOT16-09 scene



experiment, two scenes from the MOT benchmark were used, the MOT16 benchmark and the MOT20, respectively. We also provide the MOTA and MOTP metrics as seen on the MOT benchmark.

4.3.3.1 Results on the MOT16 scene In Tables 20 and 21, the results of the MOT16 are illustrated. The results show that YOLOv3-Tiny achieves the best execution throughput with 15 frames per second using 1080p video as input. YOLOv3 and YOLOv4-608 perform at nearly half the performance with 8 and 7.6 FPS, respectively. In Fig. 42, we also provide the precision and recall curve.

However, YOLOv3-Tiny failed to keep track of most objects due to its poor detection rate, while it also had a MOTA score of 37.2 and MOTP score of 76.1. Many

pedestrians that were not close to the camera could not get tracked, as seen in Fig. 43. The YOLOv3-Tiny-powered Deep SORT could not perform well enough having a modified Deep SORT count of 14 and 52 initiated tracks.

YOLOv3-608 did much better compared to YOLOv3-Tiny, having a modified Deep SORT count of 22 and 81 initiated tracks with a MOTA score of 57.1 and MOTP score of 78.8. The high number of initiated tracks does show that tracking was still relatively poor, given that we only need to track 25 pedestrians. As you can see in Fig. 44, this detector did a much better job at detecting pedestrians that were far away from the camera. It did face a problem that we will describe right below during our YOLOv4-608 notes.

Fig. 45 YOLOv4-608-enabled framework on the MOT16-09 scene

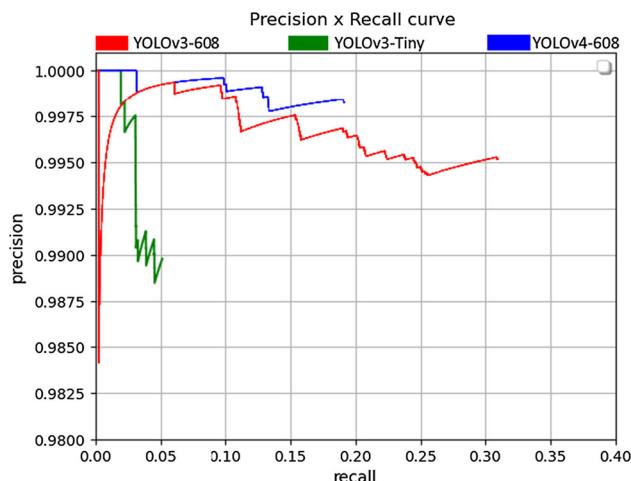


Table 22 MOT results on the MOT20 scene

MOT20-01	Frames per second	Deep SORT Initiated	Modified Deep SORT	Ground truth	Precision	Recall	F1 score	TP	FP	FN
YOLOv3-Tiny	15.5	114	20	90	0.989	0.051	0.097	1360	14	25,287
YOLOv3-608	6.9	188	48	90	0.995	0.309	0.472	8245	40	18,402
YOLOv4-608	7.3	135	41	90	0.998	0.191	0.320	5092	9	21,555

Table 23 MOT metrics on the MOT20 scene

MOT20-01	MOTA	MOTP
YOLOv3-Tiny	4.9	67.1
YOLOv3-608	31.8	73.3
YOLOv4-608	18.9	59.7

**Fig. 46** Precision–recall curve comparison on the MOT20-01 video

YOLOv4-608 achieved the best tracking performance, and our framework has a MOTA score of 42.3 and a MOTP score of 61.9. It was able to detect the people behind the glass entrance of the shop and it was able to keep track of people better than YOLOv3. It initiated less tracks when compared to YOLOv3-608. One major advantage of YOLOv4 was that it was much better at proper distinction and feature capture of the tracks. As an example, notice in Fig. 45 how the old man in the background is now properly labeled at track 25 and not at track 5, in contrast to Fig. 44, where YOLOv3 thought it was the same pedestrian that passed at the beginning of the video.

4.3.3.2 Results on the MOT20 scene In Tables 22 and 23, we see the results for MOT20 scene. The results show that YOLOv3-Tiny achieves the best execution throughput with 15.5 frames per second. YOLO3 and YOLO4-608 perform at nearly half the performance with 6.9 and 7.3 FPS, respectively. In Fig. 46, we also provide the precision and recall curve.

In this scene, the modified Deep SORT coupled with YOLOv4 was faster than v3 and this is attributed to the overall less detected tracks per frame that v4 had as also confirmed by the recall and F1 score. This reduced the number of CPU cycles needed to calculate trajectories and

Fig. 47 YOLOv3-Tiny-enabled framework on the MOT20-01 scene

Fig. 48 YOLOv3-608-enabled framework on the MOT20-01 scene



Fig. 49 YOLOv4-608-enabled framework on the MOT20-01 scene



Fig. 50 Ground truth of the MOT20-01 scene for the same frame tested



confirm tracks. The worse detector was again YOLOv3-Tiny, since it failed to keep track of most objects due to its poor detection rate, having a MOTA score of 4.9 and a MOTP score of 67.1. The framework using YOLO-Tiny detector had a count of 20 and 114 initiated tracks, which is far from the ground truth of 90 tracks. We provide a screenshot in Fig. 47 that shows its failure to detect many of the pedestrians.

YOLOv3-608 did much better compared to YOLOv3-Tiny, having a Deep SORT count of 48 and 188 initiated tracks with a MOTA score of 31.8 and a MOTP score of 73.3. The high number of initiated tracks does show that tracking was still relatively poor, given that we only need

to track 90 pedestrians. As shown in Fig. 48, this detector did a much better job at detecting pedestrians that were far away from the camera. Having said that, the people that were very far away could not be detected by any of our models. At that part of the scene, heavy occlusion occurs and there is a hefty amount of video noise and blur from the poor lighting conditions.

Finally, this time YOLOv4-608 achieved worse tracking performance, while also having a worse Deep SORT count compared to YOLOv3-608 having a MOTA score of 18.9 and a MOTP score of 59.7. It initiated 135 tracks, which is significantly lower when compared to YOLOv3-608, but that is just because it failed to track many of the pedestrians

as seen in Fig. 49. The population density of this scene paired with the increased video noise and the reflection from the sun at the back make this scene incredibly difficult to complete. We show the ground truth bounding boxes for this scene at the same frame in Fig. 50.

5 Conclusions

In this paper, first we explore the performance of various deep learning methods on the task of multiple-object tracking. We examine how widespread deep learning architectures are performing under various contexts in a wide range of scene scenarios. We introduced a modification of the Deep SORT algorithm, which aids at properly displaying the track IDs, a crucial aspect of real time object tracking. Our modification on the Deep SORT is based on the process of the initialization of the object IDs, and its rationale is to consider an object as “tracked” if it is detected in a set of previous frames, while properly passing the information to the framework, a problem that occurred in all Deep SORT and YOLO implementations we found. The results indicate that our Deep SORT modification is functional across all tests.

In addition, we present a way to improve the real-time operation of the deep learning methods by identifying and facing bottlenecks in the MOT framework. We tested and provide a way that can greatly improve the execution time of the tracking process. The results show that we have an increase of frames per second (FPS) in all examined deep learning networks, which is up to 22%. Through our experimental process and our results, we found out that through the use of a dataset specialized in car traffic, we can achieve better performance than using the models trained on the MS COCO dataset. As we saw, during testing, the YOLOv3-Tiny-enabled framework was only suitable for simple scenes, where small occlusions occur, and the field of view remains constrained. YOLOv4 offered the best performance, which was later enhanced by using the UA-DETRAC dataset during training and we also provide what we consider the optimal parameters for each framework tested. Finally, we have created and introduced a new vehicle dataset from the videos we captured consisting of 7 scenes, 11.025 frames and 25.193 bounding boxes. The dataset is suitable for testing and training multiple-object detectors and includes a variety of scenes, capture devices and daytime changes in efforts to cover weak points detectors may have.

A main direction that future work could examine concerns the addition of more features on the Deep SORT algorithm such as being able to track and label more than one classes at a time and also adjust the tracking process to take into account camera movement. Furthermore, we also

plan to compare and explore tracking performance using our own custom, fine-tuned and purpose built detectors.

6 Supplementary materials

All necessary materials, code and datasets can be found at <https://github.com/Jimmeimetis/Deepsort-Yolo-implementations>.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Luo W, Zhao X, Kim T-K (2014) Multiple object tracking: a review. CoRR. <http://arxiv.org/abs/1409.7618>
2. Ciaparrone G, Sánchez FL, Tabik S, Troiano L, Tagliaferri R, Herrera F (2020) Deep learning in video multi-object tracking: a survey. Neurocomputing 381:61–88
3. Sanchez-Matilla R, Poiesi F, Cavallaro A (2016) Online multi-target tracking with strong and weak detections. vol 9914, pp 84–99. https://doi.org/10.1007/978-3-319-48881-3_7
4. Sadeghian A, Alahi A, Savarese S (2017) Tracking the untrackable: learning to track multiple cues with long-term dependencies. CoRR. <http://arxiv.org/abs/1701.01909>
5. Wojke N, Bewley A, Paulus D (2017) Simple online and realtime tracking with a deep association metric. CoRR. <http://arxiv.org/abs/1703.07402>
6. Redmon J, Farhadi A (2018) YOLOv3: an incremental improvement. CoRR. <http://arxiv.org/abs/1804.02767>
7. Bochkovskiy A, Wang C-Y, Liao H-YM (2020) YOLOv4: optimal speed and accuracy of object detection
8. Bewley A, Ge Z, Ott L, Ramos F, Upcroft B (2016) Simple online and realtime tracking. CoRR. <http://arxiv.org/abs/1602.00763>
9. Bernardin K, Stiefelhagen R (2008) Evaluating multiple object tracking performance: the CLEAR MOT metrics. EURASIP J Image Video Process 2008(1):246309. <https://doi.org/10.1155/2008/246309>
10. Leal-Taixé L, Milan A, Reid ID, Roth S, Schindler K (2015) MOTChallenge 2015: towards a benchmark for multi-target tracking. CoRR. <http://arxiv.org/abs/1504.01942>
11. Voigtlaender P et al (2019) MOTS: multi-object tracking and segmentation. CoRR. <http://arxiv.org/abs/1902.03604>
12. Wen L et al (2015) DETRAC: a new benchmark and protocol for multi-object tracking. CoRR. <http://arxiv.org/abs/1511.04136>
13. Fuchs F, Kosiorek AR, Sun L, Jones OP, Posner I (2019) End-to-end recurrent multi-object tracking and trajectory prediction with relational reasoning. <http://arxiv.org/abs/1907.12887>
14. Chu P, Ling H (2019) FAMNet: joint learning of feature, affinity and multi-dimensional assignment for online multiple object tracking. CoRR. <http://arxiv.org/abs/1904.04989>
15. Wang X, Cheng P, Liu X, Uzochukwu B (2018) Focal loss dense detector for vehicle surveillance. <http://arxiv.org/abs/1803.01114>

16. Sun S, Akhtar N, Song X, Song H, Mian A, Shah M (2020) Simultaneous detection and tracking with motion modelling for multiple object tracking. <http://arxiv.org/abs/2008.08826>
17. Chu Q, Ouyang W, Li H, Wang X, Liu B, Yu N (2017) Online multi-object tracking using CNN-based single object tracker with spatial-temporal attention mechanism. <http://arxiv.org/abs/1708.02843>
18. Ristani E, Solera F, Zou RS, Cucchiara R, Tomasi C (2016) Performance measures and a data set for multi-target, multi-camera tracking. CoRR. <http://arxiv.org/abs/1609.01775>
19. Redmon J, Divvala SK, Girshick RB, Farhadi A (2015) You only look once: unified, real-time object detection. CoRR. <http://arxiv.org/abs/1506.02640>
20. Maiya SR (2020) abhyantika/nanonet_object_tracking. GitHub. https://github.com/abhyantika/nanonet_object_tracking
21. Milan A, Leal-Taixé L, Reid ID, Roth S, Schindler K (2016) MOT16: a benchmark for multi-object tracking. CoRR. <http://arxiv.org/abs/1603.00831>
22. Dendorfer P et al (2020) MOT20: a benchmark for multi object tracking in crowded scenes. <http://arxiv.org/abs/2003.09003>
23. Emami P, Pardalos PM, Elefteriadou L, Ranka S (2020) Machine learning methods for data association in multi-object tracking. ACM Comput Surv (CSUR) 53(4):1–34
24. Hou X, Wang Y, Chau LP (2019) Vehicle tracking using deep SORT with low confidence track filtering. In: 2019 16th IEEE international conference on advanced video and signal based surveillance (AVSS). IEEE, pp 1–6
25. Wojke N, Bewley A (2018) Deep cosine metric learning for person re-identification. In: 2018 IEEE winter conference on applications of computer vision (WACV). IEEE, pp 748–756
26. Karunasekera H, Wang H, Zhang H (2019) Multiple object tracking with attention to appearance, structure, motion and size. IEEE Access 7:104423–104434
27. Voigtlaender P, Krause M, Osep A, Luiten J, Sekar BBG, Geiger A, Leibe B (2019) MOTS: multi-object tracking and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7942–7951
28. Sun S, Akhtar N, Song H, Mian AS, Shah M (2019) Deep affinity network for multiple object tracking. IEEE Trans Pattern Anal Mach Intell 43:104–119
29. Liu G, Liu S, Muhammad K, Sangaiah AK, Doctor F (2018) Object tracking in vary lighting conditions for fog based intelligent surveillance of public spaces. IEEE Access 6:29283–29296
30. Xu S, Savvaris A, He S, Shin HS, Tsourdos A (2018) Real-time implementation of YOLO+ JPDA for small scale UAV multiple object tracking. In: 2018 international conference on unmanned aircraft systems (ICUAS). IEEE, pp 1336–1341
31. Yoon YC, Boragule A, Song YM, Yoon K, Jeon M (2018) Online multi-object tracking with historical appearance matching and scene adaptive detection filtering. In: 2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS). IEEE, pp 1–6
32. Darknet: open source neural networks in c. <https://github.com/AlexeyAB/darknet>. Accessed 25 Apr 2020
33. Hou X, Wang Y, Chau LP (2019) Vehicle tracking using deep SORT with low confidence track filtering. In: 2019 16th IEEE international conference on advanced video and signal based surveillance (AVSS). IEEE, pp 1–6
34. Nguyen HQ, Nguyen TB, Nguyen TA, Le TL, Vu TH, Noe A. Comparative evaluation of human detection and tracking approaches for online tracking applications
35. Lin JP, Sun MT (2018) A YOLO-based traffic counting system. In: 2018 conference on technologies and applications of artificial intelligence (TAAI). IEEE, pp 82–85
36. Padilla R, Netto SL, da Silva EA (2020) A survey on performance metrics for object-detection algorithms. In: 2020 international conference on systems, signals and image processing (IWSSIP)
37. Wang Z, Zheng L, Liu Y, Wang S (2019) Towards real-time multi-object tracking. arXiv preprint <http://arxiv.org/abs/1909.12605>
38. Lu HC, Li PX, Wang D (2018) Visual object tracking: a survey. Pattern Recognit Artif Intell 31(1):61–76
39. Yao R, Lin G, Xia S, Zhao J, Zhou Y (2020) Video object segmentation and tracking: a survey. ACM Trans Intell Syst Technol (TIST) 11(4):1–47
40. Llamazares Á, Molinos EJ, Ocaña M (2020) Detection and tracking of moving obstacles (DATMO): a review. Robotica 38(5):761–774
41. Sam JR, Augasta G (2021) Review of recent advances in visual tracking techniques. Multimed Tools Appl 80:24185–24203

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.