Multiple Object Detection and Tracking

Francesco Vidaich
francesco.vidaich@studenti.unipd.it

Stefano Mancone
stefano.mancone@studenti.unipd.it

Abstract

In this work we implement a YOLOv3 architecture, trained on the COCO dataset, to solve the problem of Multiple Object Detection and the SORT (Simple Online and realtime Tracking) algorithm that, combined with the previous model, will solve the problem of Multiple Object Tracking. We will also introduce some metrics to evaluate the how well the model performs.

1. Introduction

Multiple object tracking (MOT) is the process of locating multiple objects over a sequence of frames (video). The MOT problem can be viewed as a data association problem where the goal is to associate detections across frames in a video sequence. The targets can be people, animals, cars and many other objects, depending on which ones the detection model was trained. In our case, the detection model YOLOv3 [13] was trained with the COCO [10] dataset, which is contains 80 different types of objects. The obtained MOT system is tested over many videos provided by MOTChallenge [9], a Multiple Object Tracking Benchmark which contains also the results of state-of-the-art tracking methods over the same videos. In order to compare them with our model, we implemented the *py-motmetrics* [7] library, which provides a Python implementation of metrics for benchmarking multiple object trackers.

1.1. Related Work

We first review the related work on MOT. Multi-object tracking can be classified into online methods [4] [5] [1] and batch methods [22] [2] based on whether they rely on future frames. Online methods can only use current and previous frames while batch methods use the whole sequence. These models work with already detected object, where the only task is to link the boxes over time. The rapid development of deep learning has motivated researchers to explore modern object detectors [21] [11]. Many of them treat object detection and re-identification as two separate task. They apply CNN-based object detectors such as Faster R-CNN [15] and YOLOv3 [13] to localize all objects of interest in input images, and then the results are handled by a Multi-object tracking method like the one described before. However, they are usually very slow because the two tasks need to be done separately without sharing. So it is hard to achieve video rate inference which is required in many applications. For this reason, one-shot MOT has begun to attract more research attention [17] [18]. The core idea is to simultaneously accomplish object detection and identity embedding (re-IDfeatures) in a single network in order to reduce inference time. Moreover, the field Multi-object tracking has been expanded also to the 3D domain [20], in which the system obtains 3D detections from a LiDAR point cloud to build boxes that identify the physical object.

1.2. Datasets

1.2.1 COCO Dataset

The COCO [10] dataset, meaning "Common Objects In Context", is a set of challenging, high quality datasets for computer vision, mostly state-of-the-art neural networks. This large-scale dataset can be used to tackle object detection, segmentation, captioning and other tasks. It it is composed by 330K images (more than 200K of them are labeled), which contain a total of 1.5 million object instances. The dataset contains 80 object classes; considering that it was used to train the YOLOv3 detection model, these classes are the ones that the model can detect (the *coco.names* file provides the list of objects that can be detected).

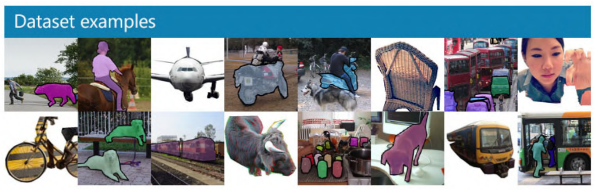


Figure 1: Examples from the COCO dataset

1.2.2 MOTChallenge

MOTChallenge [9] is a framework for the fair evaluation of multiple people tracking algorithms. The framework provides a large collection of datasets containing challenging sequences and the detections for all of them. Moreover, it functions also as a benchmark to compare the performance of state-of-the-art MOT architectures using various metrics. Each video is presented as a sequence of frames, and the ground truth detections is provided in a file, in which each line describes the position and the size of a specific bounding box in a specific frame. These files allowed us to measure how well our model performed over the different sequences.

2. Theory

2.1. Multiple Object Detection model

The model that tackles the Multiple Object Detection problem is YOLOv3 [13], which improves the system introduced with YOLO [14] (You Only Look Once).

2.1.1 YOLO and the Anchor Boxes system

YOLO [14] is a real-time object detection algorithm, in which a convolutional neural network returns for each grid cell many possible detections (one for each anchor box) and their confidence. This is done to handle cases of overlap, i.e. whenever one cell contains the centre points of more objects. Anchor boxes are a set of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in your training datasets. During detection, the predefined anchor boxes are tiled across the image. Since most of the predicted anchor boxes will have a very low confidence (probability of object being present in it) value, we use NMS (Non-Maximal Suppression) to get rid of boxes with a confidence lower than a given threshold (called `conf_thres` in the code). After this first step, the NMS also eliminates all the boxes whose Intersection Over Union (IOU) value, which represents the portion of the box overlapping the object divided by the non-overlap portion, is higher than another threshold (called `nms_thres` in the code). It will repeat this until every non-maximal bounding box is removed for each object.

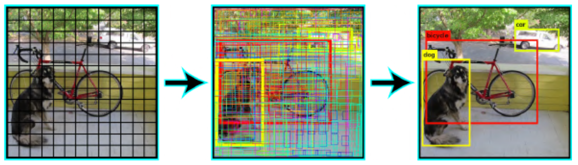


Figure 2: *The YOLO system (1) resizes the input image to the desired size, (2) runs a convolutional network on the image that returns many possible detections and their confidence, and (3) thresholds the resulting detections.*

2.1.2 YOLOv3 Architecture

YOLO is one of the fastest object detection algorithms, but its accuracy is very poor compared against more recent models. While YOLOv2 [12] introduced some improvements in terms of accuracy and speed, YOLOv3 [13] traded off that speed for further boosts in accuracy. Like its previous versions, the whole system can be divided into two major components: Feature Extractor and Detector. The most important new feature that characterizes v3 is that both components are multi-scale, meaning that we can obtain feature embeddings at three scales and feed them into three different branches of the Detector. The Feature Extractor of YOLOv3 uses Darknet-53, a 53-layer Convolutional Neural Network that features Residual layers, which allow to skip connections to help the activations to propagate through deeper layers without gradient diminishing. Since YOLOv3 is designed to be a multi-scaled detector, detection is done by taking feature maps of three different sizes from three different places in the network (see Figure 3) and by applying a $1 \times 1 \times (B * (5 + C))$ detection kernel over them, where B is the number of possible bounding boxes for each cell, C is the number of classes and the ”5” accounts for the 4 bounding box attributes (width, height and the position of its top-left corner) and one object confidence

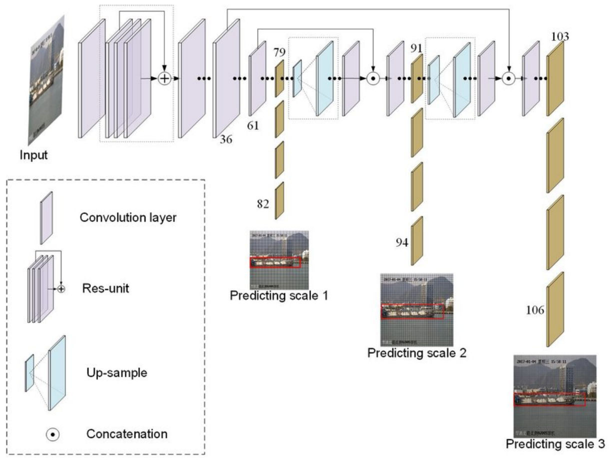


Figure 3: *YOLOv3 network architecture for feature extraction.*

In our case we used $B = 3$ anchors for each scale (for a total of 9 different anchor boxes) and, since YOLOv3 was trained using the COCO dataset with $C = 80$ classes of objects, our kernel had size $1 \times 1 \times 255$. Considering that we always resized the input image to 416×416, the three feature maps on which we applied the kernel have size 13×13, 26×36 and 52×52 (scales are given by downsampling the image 32, 16 and 8 times respectively). By choosing correct sizes for the anchor boxes at the three scales, we can make the detection feature map with shape $13 \times 13 \times 255$ responsible for detecting large objects, the one with shape $26 \times 26 \times 255$ responsible for medium objects and the $52 \times 52 \times 255$ one responsible for small objects. In total, since we consider the image at three different scales (for which there are 3 anchor boxes each), the total number of predicted boxes is

$$3 \times (13 \times 13 + 26 \times 26 + 52 \times 52) = 10647$$

Since YOLOv2 does not have a multi scale detection (only the 13×13 layer is used), the number of predicted boxes of YOLOv3 is more than 10 times bigger, resulting in a slower but much more accurate model.

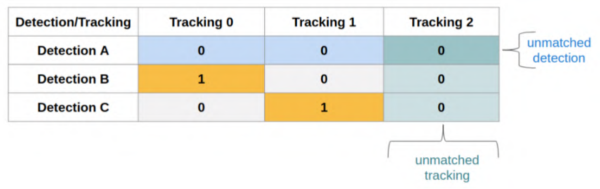
After that, YOLOv3 performs multilabel classification for objects detected in each box. Object confidence and class predictions in YOLOv3 are predicted through logistic regression and then a process similar to the one described in Section 2.1.1 occurs in order to discard every box with low confidence.

2.2. Multiple Object Tracking algorithm: SORT

The Simple Online and Realtime Tracking SORT [4] is one of the most popular and one of the simplest algorithms for tracking multiple objects. In few words, it associates already detected objects across different frames based on the coordinates of detection results. SORT does not need to learn anything or to know what are the tracked objects because it is based on a heuristic method that combines the Kalman filter [19] and the Hungarian algorithm [8].

The SORT algorithm associates an objects from one frame to another based on a score: in this work we use IOU (Intersection Over Union), which measures how much two boxes are overlapped. Consider an example with two following frames at times $t-1$ and t. For each iteration of the algorithm, we have to compute the IOU between every detection obtained from frame t and every target's prediction obtained from detections belonging to frame $t-1$. The target's identities and its boxes are propagated through time using the Kalman filter, which can predict the positions of each centre of the box and their scales at the following timestep using a linear velocity model. After storing all the IOU scores in a matrix, we use the Hungarian algorithm (implemented in the function `scipy.optimize.linear_sum_assignment`) in order to assign each new detection with the target's prediction by minimizing the cumulative IOU score.

When objects enter and leave the image, unique identities need to be created or destroyed accordingly (see Figure 4). For creating these identities, we consider any detection with an overlap less than IOU$_{min}$ to signify the existence of an untracked object. The tracker is initialised using the geometry of the bounding box with the velocity set to zero. Tracks are terminated if they are not detected for T_{Lost} frames. Since the linear velocity model is a poor predictor of the true dynamics and we are primarily concerned with frame-to-frame tracking, the baseline SORT algorithm uses $T_{Lost} = 1$. If an object reappears after few frames, tracking will implicitly resume under a new identity.



Detection/Tracking	Tracking 0	Tracking 1	Tracking 2
Detection A	0	0	0
Detection B	1	0	0
Detection C	0	1	0

unmatched detection
unmatched tracking

Figure 4: *An "Unmatched detection" implies that a new object has been identified, so we need to create a new ID. An "Unmatched tracking" implies that the previously tracked object is not detected anymore, therefore its ID must be deleted. A cell value is "1" if IOU > IOU$_{min}$.*

2.3. Metrics

In order to evaluate this kind of task we need to define adequate metrics. This is not a trivial step because while in other machine learning is usual to have many more literature and the evaluation tactics and metrics are already a standard, multi object tracking is a relatively new task which could not really be addressed except in recent years. The following questions therefore arises: Which errors do we want to minimize? Which kind of correct guesses do we want to maximize? Depending on the application, the answer can be very different.

For example, in the case of an autonomous driving system, it is important to identify all the objects that are on the road, i.e. to maximize the True Positives (TP) and minimize the False Negatives (FN) (which would lead to an accident) so perhaps the recall (TP / TP + FN) could be an adequate metric.

A different example could be if we want to count the number of people that go through a particular street to analyze the city flux. In this case it is crucial to count each person one and only one time. We introduce then the concepts of mostly tracked targets (MT), mostly lost targets (ML) and partially tracked (PT). The first means that a real target was tracked for more than 80% of frames, the second means that the target was tracked for less than (20%) of frames and the last one identifies all the other targets. Then, an appropriate metric that we want to maximize in this case could be a ratio between this quantities such as naively MT/ML or a smoother version as $\frac{MT+PT}{ML+PT+1}$. In the second case the perfect outcome would be the number of trackable entities (we can always normalize and obtain a metric in $[0, 1]$).

Rather than seeking the best metric for all the different cases in MOT, MOTChallenge [9] choose to aggregate the most important ones and evaluate each sequence by using all of these metrics for each submitted algorithm. In this way they can be compared by following the metric that best suits the particular problem. Some are very standard, although they are not less relevant for any reason, such as false positive, false negative, recall and precision. We then have the already mentioned MT, ML and PT. The left metrics that we are going to talk about come mainly from [3] and [16].

2.3.1 MOTA and MOTP

The first two are called multi object tracking accuracy and precision ($MOTA$ and $MOTP$) and are from [3]. In order to work with this metrics, we need to introduce the concept of distance between a tracked object and a ground truth object from labeled data. The distance can be of course arbitrarily chosen but a common choice is the IOU (Intersection Over Union) norm distance, which is a similarity measure where we see two box that identify an object as sets. The IOU measure for sets is also known as Jaccard distance in a more abstract context, and it is defined as

$$d_J(A,B) = 1 - J(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

where the intersection of two boxes is the number of common pixels while the union is the number of pixels contained in both boxes together. This similarity distance become 1 if the boxes does not have common pixels and 0 if all the pixels are the same. Having this notion of distance we can define the $MOTA$ metric as

$$MOTA = 1 - \frac{\sum_t (m_t + \text{fp}_t + \text{mme}_t)}{\sum_t g_t}$$

where m_t is the number of misses at a given frame t, i.e. real objects that weren't identified by the algorithm (meaning that it has $d_J = 1$ with each identified object), which also represents the false negatives. fp_t are the false positives at frame t i.e. identified objects that are not associated to any ground truth entity. mme_t count the mismatch error at frame t, this error is the number of trajectories (object identified with the same id among different frames) that switched followed tracked subject w.r.t. the ground truth, see Fig 5 for a graphic example of this eventuality. Lastly g_t is just the number of objects (ground truth) present at time t in the frame.

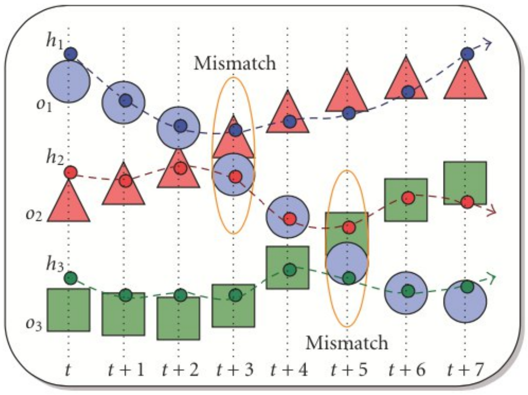


Figure 5: Mismatch error, o_1 cross the path first with o_2 and then with o_3. The model wrongly switch the hypothesis ids two times corresponding to a $\sum_t \text{mme}_t = 2$

The second metric introduced in [3] $MOTP$ is defined as

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}$$

Where d_t^i is the distance between an object and his associated hypothesis from the model at time t and c_t is the number of matches found at time t i.e. the number of associations object/hypothesis.

2.3.2 ID metrics

The other relevant metrics are [16]. Here the authors redefine standard statistical metrics such as precision recall and F1 score by changing the definition of true positive, true negative, false positive and false negative.

We start with the set of the true trajectories $\{\tau\}$ and the set of the computed trajectories $\{\gamma\}$, we then have to build a bipartite graph $G = (V_T, V_C, E)$. The vertex set V_T has one "regular" node for each true trajectory τ and one "false positive" node for each computer trajectory γ. The vertex set V_C then has one "regular" node for each computer trajectory γ and one "false negative" node for each true trajectory τ. We then define the $m(\tau, \gamma, t, \Delta)$ function that, for a given frame t, a true trajectory, a computed trajectory and a threshold Δ, calculate d_J and return 1 (miss) if the distance of the two boxes (predicted and true) associated to the two trajectories at that time is more than the threshold, i.e. if $d_J > \Delta$, otherwise $m(\tau, \gamma, t, \Delta) = 0$. Then the set of the edges in the G graph is computed by pairing the nodes from the two sets of the bipartite graph in a one-to-one fashion. This is done by minimizing the cumulative cost function of a particular configuration where each edge has a cost defined by

$$c(\tau, \gamma, \Delta) = \sum_{t \in \mathrm{T}_\tau} m(\tau, \gamma, t, \Delta) + \sum_{t \in \mathrm{T}_\gamma} m(\tau, \gamma, t, \Delta)$$

where the two sets T_τ and T_γ are the time sets where the two trajectories τ and γ are defined.

The first sum can be then identified with the false negatives associated while the second with the false positive. While the meaning of this cost function is easy to understand for two regular nodes, the interpretation of a regular node and a false node (positive/negative) must be intended in the sense that if the cost of associating a trajectory with itself is globally less than associating it with another regular trajectory then we can conclude that the trajectory is an error, a false positive or false negative depending on whether we are looking at a τ or γ trajectory.

Once this optimization is done we are left with (τ, γ) matches which are true positives ID $(IDTP)$, $(\bar{\gamma},\ \gamma)$ matches which are false positive ID $(IDFP)$, $(\tau, \bar{\tau})$ matches which are false negative ID $(IDFN)$ and finally $(\bar{\gamma}, \bar{\tau})$ that are true negative matches ID $(IDTN)$.

We can then finally redefine three standard statistics metrics declined in our framework, precision, recall and F_1 score

$$IDP = \frac{IDTP}{IDTP + IDFP}$$

$$IDR = \frac{IDTP}{IDTP + IDFN}$$

$$IDF1 = \frac{2IDTP}{2IDTP + IDFP + IDFN}$$

Which gives an adequate measure of these three quantities in this context. We will see in the results section how the different metrics we presented behave.

3. Code Development

The starting point of the project was a repository [6] that provided us the tools that form the backbone of this project: the already trained YOLOv3 and two python notebooks that illustrate how it could be implemented to perform object detection and tracking. We reorganized the code in order to have two high level scripts: *PyTorch_Object_Detection.py* and *PyTorch_Object_Tracking.py*.

Let's talk about how these two scripts actually work, starting from the first one. After loading the YOLOv3 model, the script loads the desired image and uses the `detect_image()` function, which uses YOLOv3 to perform the multilabel classification at the three different scales (as described in section 2.1.2) and then uses the `non_max_suppression()` function to filter out the low confidence detections (as described in section 2.1.1). Then, each detected bounding box is plotted over the image and the final result is stored.

In the second script an external program is called, *FFmpeg*, which, in conjunction with the `matplotlib.animation` library, allows us to build and store the tracked video. In particular, the script takes in input the frames of the video that we want to analyze and, after loading the YOLOv3 model, it also initialize the SORT object. In this case, the `detect_image()` function is called for each frame but, before plotting the resulting boxes over their corresponding image, we use the SORT object to build IDs for each box and to keep them identified through frames (as described in section 2.2). The output video containing the tracked objects is built iteratively by adding one analyzed frame at a time to the mp4 file. In addition to the video, the script also builds the file */data/out.txt*, which contains the boxes attributes and IDs of the tracked objects for each frame following the conventional MOTchallenge format[1].

In order to obtain the metrics we referred to the *py-motmetrics* [7] library, which implemented most of the used metrics in MOTchallenge. Then, we give this file and the one corresponding to the ground truth (*gt.txt*) provided by the dataset to the *py-motmetrics* library in order to obtain the different metrics that we described in the previous section.

In the root directory there is a bash script that let us to run the whole inference and evaluation process with a single command, for more details read the README file in the main directory of the project.

4. Results

4.1. Detection on single images

First, let's consider only the YOLOv3 detection model. If the MOTChallenge datasets contains mainly sequences from security cameras in which there are almost only moving people, the model can detect and classify many other objects. This can be seen in Figure 6, in which we show output results over images that contain different types of objects.

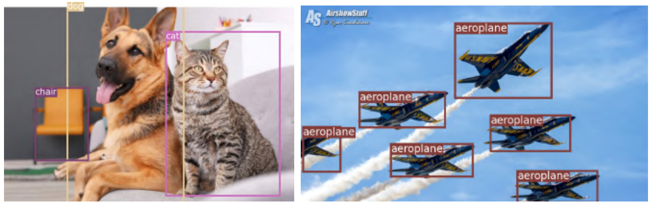


Figure 6: Object detection results over different images

In addition to the detected image, the *PyTorch_Object_Detection.py* script returns also the inference time (i.e. the time needed to complete the object detection task over the image). This quantity can be really important, because having a small enough inference time implies

[1]See https://motchallenge.net/instructions/

that the detection model could also be used for a realtime object tracking system. If running the model on the CPU did not return promising results in this regard, as this value fluctuated around 1 second, the situation changed drastically when we started performing the same task enabling CUDA to run the model on the GPU. In these case the inference time dropped below 0.1 seconds. To put it in perspective, knowing that the time needed to run the SORT algorithm is negligible when compared to the inference time, the model could have been used to perform realtime object tracking on a video sequence with a framerate around 10 fps, especially if we consider that many industrial security cameras operate at framerates a lot lower than the standard 30 fps (the average is around 10-15 fps) in order to minimize the storage costs.

However, the realtime object tracking problem was never tackled during this work since, as described in Section 3, each analyzed frame of the video was iteratively stored on disk, which is a task that takes a time much longer than inference time, thus reducing significantly the number of frames that are analyzed in a second.

4.2. Detection and tracking on video sequences

Here we present two showcases in Fig 7 from the MOT datasets (the eleventh and the tenth dataset of MOT16) that can help us to understand the algorithm pitfalls and strengths and if they are reflected in the metrics.

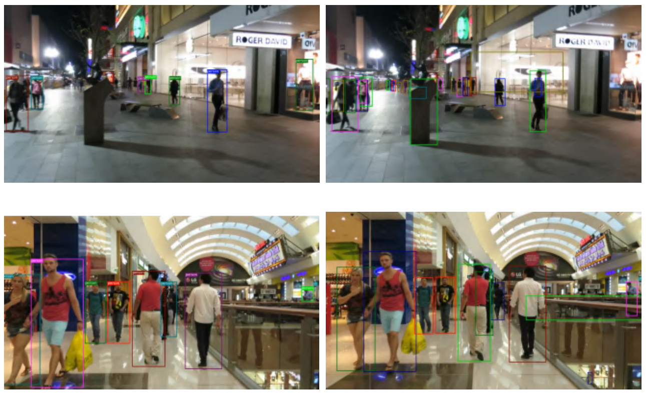


Figure 7: Single frame comparison between the tracking model (left) and ground truth (right) for MOT datasets 10 (top) and 11 (bottom)

We want then to compare the results with the obtained metrics that are reported in Tab 1 and in Tab 2. In both scenes, most of the people are tracked but, while in the bottom one there is a lot of lighting and the people occupy a good portion of the image, in the top one the lighting is poor and a lot of people is in the background. Also the top image is blurry because the camera is moving. In both cases

Set	IDF1	IDP	IDR	Rcll	Prcn
10	33.0%	57.9%	23.0%	34.2%	85.9%
11	52.5%	67.9%	42.8%	57.8%	91.8%

Table 1: First group of metrics, the first three are from [16], then we have plain recall and precision over all the boxes.

Set	GT	MT	PT	ML	MOTA	MOTP
10	54	7	15	32	27.8%	0.239
11	69	15	21	33	52.3%	0.177

Table 2: Second group of metrics, GT is the ground truth number of trajectory then we have the number of which are mostly tracked, partially tracked and mostly loss. The last two metrics are the ones defined in [3].

we have that a very high precision, meaning that the identified boxes are effectively real people so, if it is important to have few false positives, it would seem that the performance is adequate. On the other hand, the recall metric in both its versions is very low for the set 10, a lot of people in the background could not have been detected and, given the reasons already discussed (blur and lighting), in the set 11 a much higher percentage of objects were detected and thus we obtained an higher recall. F_1 score summarize precision and recall and its values supports this, but the metric to look in a specific case depends on whether we are interested in generic performance or in minimizing a particular type of error.

For the second group of metrics, which are a little more specific about tracking, we have that the algorithm performs better in the second case as we saw before. Here however the performance seems to be not good enough for a real application since most of the object are either mostly loss or partially tracked and very few mostly tracked. An interesting point is that the MOTA metric is very different between the two shown cases: using only metric, we can't prove wether this is due to having more false positives, false negatives or mismatches we do not know, but this metric confirms again that the algorithm performed better in the second video.

5. Conclusions

In this project we used an already implemented and trained neural network architecture (YOLOv3) and a multi object tracking algorithm (SORT) to make inference over different videos from the MOTchallenge benchmark site. This work was mostly a proof of concept about how a multi object tracking pipeline could be implemented but, in order to evaluate the performance of our model, we also implemented different metrics. From here a future work could be a more fine search on which choices are better depending on the particular environment, using different and more advanced models, different tracking algorithms and metrics.

References

[1] Seung-Hwan Bae and Kuk-Jin Yoon. “Robust Online Multi-Object Tracking Based on Tracklet Confidence and Online Discriminative Appearance Learning”. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR ’14. USA: IEEE Computer Society, 2014, pp. 1218–1225. ISBN: 9781479951185. DOI: `10.1109/CVPR.2014.159`.

[2] Jerome Berclaz et al. “Multiple Object Tracking using K-Shortest Paths Optimization”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33 (2011), pp. 1806–1819. DOI: `10.1109/TPAMI.2011.21`.

[3] Keni Bernardin and Rainer Stiefelhagen. “Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics”. In: *EURASIP Journal on Image and Video Processing* (2008).

[4] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)* (Sept. 2016). DOI: `10.1109/icip.2016.7533003`.

[5] E. Bochinski, V. Eiselein, and T. Sikora. “High-Speed tracking-by-detection without using image information”. In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2017, pp. 1–6.

[6] Chris Fotache. *pytorch_objectdetecttrack*. `https://github.com/cfotache/pytorch_objectdetecttrack`.

[7] Christoph Heindl. *py-motmetrics*. `https://github.com/cheind/py-motmetrics`.

[8] H. W. Kuhn. “The Hungarian method for the assignment problem”. In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97. DOI: `10.1002/nav.3800020109`.

[9] L. Leal-Taixé et al. “MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking”. In: *arXiv:1504.01942 [cs]* (Apr. 2015). arXiv: 1504.01942. URL: `http://arxiv.org/abs/1504.01942`.

[10] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: (2014). arXiv: `1405.0312 [cs.CV]`.

[11] Nima Mahmoudi, Seyed Mohammad Ahadi, and Mohammad Rahmati. “Multi-Target Tracking Using CNN-Based Features: CNNMTT”. In: *Multimedia Tools Appl.* 78.6 (Mar. 2019), pp. 7077–7096. ISSN: 1380-7501. DOI: `10.1007/s11042-018-6467-6`.

[12] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: `1612.08242 [cs.CV]`.

[13] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: `1804.02767 [cs.CV]`.

[14] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: `1506.02640 [cs.CV]`.

[15] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. arXiv: `1506.01497 [cs.CV]`.

[16] Ergys Ristani et al. "Performance Measures and a Data Set for Multi-target, Multi-camera Tracking". In: *Computer Vision – ECCV 2016 Workshops* (2016), pp. 17–35.

[17] Paul Voigtlaender et al. *MOTS: Multi-Object Tracking and Segmentation*. 2019. arXiv: `1902.03604 [cs.CV]`.

[18] Zhongdao Wang et al. *Towards Real-Time Multi-Object Tracking*. 2019. arXiv: `1909.12605 [cs.CV]`.

[19] Greg Welch and Gary Bishop. "An Introduction to the Kalman Filter". In: *Proc. Siggraph Course* 8 (Jan. 2006).

[20] Xinshuo Weng et al. *3D Multi-Object Tracking: A Baseline and New Evaluation Metrics*. 2019. arXiv: `1907.03961 [cs.CV]`.

[21] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. *Simple Online and Realtime Tracking with a Deep Association Metric*. 2017. arXiv: `1703.07402 [cs.CV]`.

[22] Li Zhang, Yuan Li, and Ramakant Nevatia. "Global data association for multi-object tracking using network flows". In: June 2008. DOI: `10.1109/CVPR.2008.4587584`.