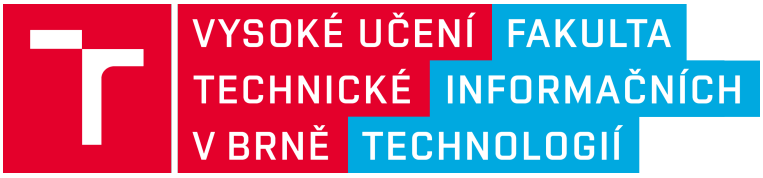
Project report for Pedestriant tracking

Alexandra Fernandes <242963@vut.cz>
Fco Javier Bolívar Expósito <xboliv00@stud.fit.vutbr.cz>

December 29, 2021



Contents

1 Introduction

Nowadays, there are several useful applications where it is necessary to know and follow people's whereabouts, for instance, having into account the pandemic we live in, following someone's contacts to retrace a contagious chain is useful.

The aim of this project is implementing a computer vision system for tracking pedestrians. Multiple Object Tracking is composed of two tasks: Detection, which consists in identifying and locating all known objects in a scene. Association, which consists in associating each detection on a timestep to a "track"/assigning an ID, so we end up with a list of detections over time of each object. "Object detection and tracking; Detection - A detection algorithm asks the question: is something there? Tracking - A tracking algorithm wants to know where something is headed." - citing from [BS19]

Our final result is able to take input from a camera, video or sequence of images, then output a video with tracks drawn over it, displayin it on the screen or saving a file and output a txt file with a list of detections associated with track for each frame in the MOT Challenge format ([MOT]). Desired input/output and all hyperparameters can be set with command line options.

2 Theory

In this section, the methods we used and why we have chosen them will be explained.

In general Pedestrian Tracking is a real-time task that ideally should be able to be performed on embedded systems. For this reason we use "online" methods (the solution is immediately available with each incoming frame and cannot be change at any later time) that are fast.

For the object detection, in this case people, YOLOv4-tiny (*You Only Look Once*) will be used and for tracking SORT algorithm. In this section, these two methods will be briefly explained, keep in mind that they are just not used "as-is" and there are some improvements that will be explained on the implementation and experimentation sections (fine-tuning on pedestrians / hyper-parameter tuning).

2.1 YOLOv4-tiny - detection

Being a Single-state detector, YOLO is one of the fastest algorithms out there to detect objects. Although it is no longer the most accurate algorithm for object detection, when you need real-time detection without losing too much precision, it is close in accuracy to state-of the art two-stage detectors [PER]. We can see that the family of YOLOv4 models has the best performance/accuracy trade-off. YOLO uses a single convolutional network to predict several bounding boxes and category probabilities for these boxes at the same time (quoting from [BS19])

YOLO-tiny has the following architecture:

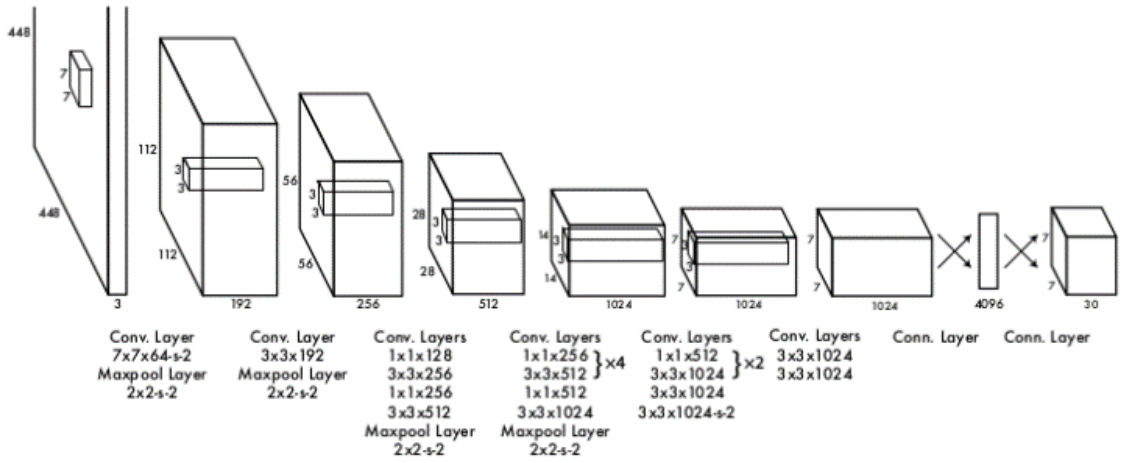


Figure 1: YOLO-tiny architecture, source:[img]

Even though Yolov4 and Scaled-Yolov4 offer great accuracy, both training and inference need a great amount of computation power. YOLO-tiny architecture is approximately 442% faster than other versions, although it is less accurate (according to [tin]). Yolov4-tiny is the only one from the family of Yolov4 models that can consistently run inference over 30fps on NVIDIA embedded boards [SPD].

2.2 SORT algorithm - tracking

SORT is an online tracker which works on the principle of tracking by detection. This method uses a strong detector to detect objects and the Hungarian

algorithm and Kalman filter are used to track them. It is designed for online tracking applications where only past and current frames are available and the method produces object identities on the fly. It tracks each detection by assigning a unique id to each bounding box, as soon an object is lost due to occlusion, wrong identification, etc. tracker assigns a new ID and start tracking the newfound object ([BS19] and [BGO+16]).

In 2016, SORT was ranked the best open source multiple object tracker on the MOT benchmark. Although since then trackers with more accuracy have been develop, SORT still has good use cases. It is the fastest method in the MOT20 challenge with a HOTA metric of 36.1, compared to the best tracker with a HOTA metric of 61.5 it has 63 more times FPS.

A significant proportion of SORT accuracy depends on the detection accuracy, so we hope even with a method this simple if we get good detections we can have a good tracker. We also would have liked to implement DeepSORT [WBP17] [WB18], a new and improved version of SORT with a Deep Association Metric, but we ran out of time.

To create this model three hyperparameters are necessary: max_age min_hits, iou_threshold, which will be further explained in section 4.1.2 of this report.

3 Detection Implementation

A great portion of our tracking results is going to depend on how good our detection is, so we need to improve as much as we can while still using the small model.

We are going to start with the first 29 layers pre-trained on 80 classes on the COCO Dataset, so the model already has learnt some features, and then we are gonna Fine-Tune training the whole model for just detecting people using Darknet.

3.1 Dataset

We have chosen the CrowdHuman Dataset [SZL+18] for it's quantity and variety of samples in respect to other datasets, so it has pretty representative samples. These two qualities are really important so the model can learn to

detect different types of persons in different contexts. In figure 2, it is possible to see the volume, density and diversity of different human detection datasets.

	Caltech	KITTI	CityPersons	COCOPersons	CrowdHuman
# images	42,782	3,712	2,975	**64,115**	15,000
# persons	13,674	2,322	19,238	257,252	**339,565**
# ignore regions	50,363	45	6,768	5,206	**99,227**
# person/image	0.32	0.63	6.47	4.01	**22.64**
# unique persons	1,273	$< 2,322$	19,238	257,252	**339,565**

Figure 2: Volume, density and diversity of different human detection datasets. Train set.

We used CrowdHuman and CityPersons datasets since they have different ranges of occlusion. This way the model can learn to detect partially occluded persons. On image 3 you can check the comparison of the visible ratio for these two datasets.

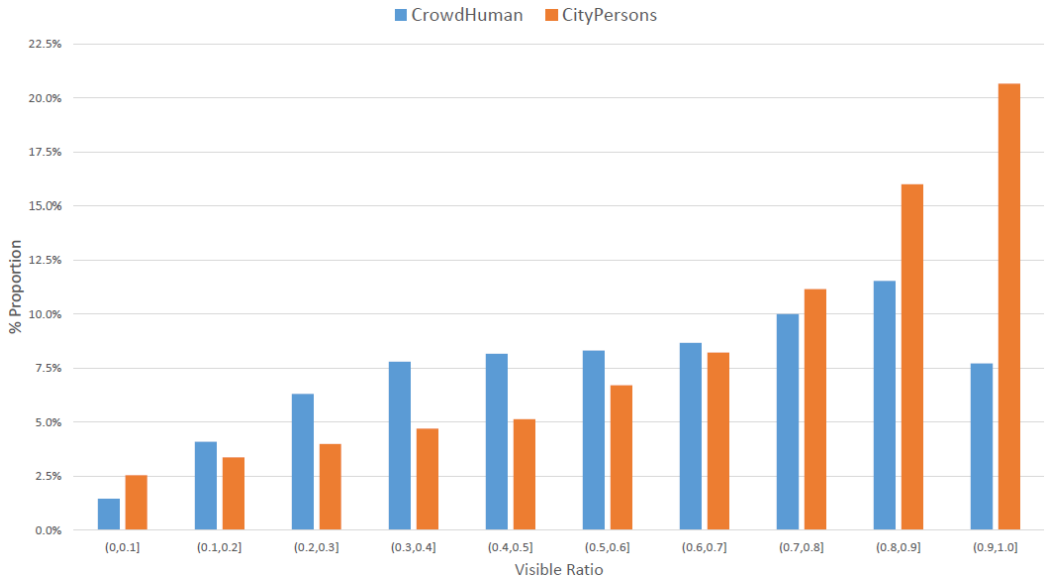


Figure 3: Comparison of the visible ratio between CrowdHuman and CityPersons dataset. Train set.

The dataset labels are not compatible with Darknet format, so we converted them using `https://github.com/alaksana96/darknet-crowdhuman` for both the training and validation sets.

3.2 Training

We are not going to enter in depth on the training process, because it is long and you have to organize the data and config files, but we will highlight some of the most important parts.

We need to slightly modify the model architecture and training hyperparameters to train for just one class, we do so mostly following the Darknet Recommendations.

To get some GPU acceleration (even if very slow) the model was trained on Google Colab. Darknet was built with CUDA support, conf files, datasets and pre-trained weights were uploaded to drive.

We trained for 6000 iterations, but the Colab Environment crashed multiple times through the 15 hours of training, losing detailed statistics of the training improvement. Still, the training was set up to save weights every 1000 iterations to google drive, allowing resuming the training and calculating accuracy metrics for each one on the validation dataset.

You can fin the explanation of all metrics used here.

With a long training we could end up overfitting on the training data, and wasting time without getting any improvement. For this reason we calculated the AP@50 metric for weights every 1000 iterations on the validation data. Then we can use this metric to choose the weights with best results on the validation data.

3.3 Parameter tuning and Results

The model was trained for a 416x416 resolution, however, we can still increase the resolution it uses for inference and get better accuracy at a performance cost. Also we can change the confidence threshold (minimum probability assigned to a detection to report it).

Multiple configurations were tried to choose the best one. Relevant configurations are shown in table 1. All results can be found on iteration_improvements.txt file.

Iterations	Resolution	AP@50	Detection Time
Orig. Pretrained	416x416	34.82%	262s
1000	416x416	31.37%	233s
2000	416x416	48.02%	210s
...	...	...	...
5000	416x416	52.06%	201s
6000	416x416	52.41%	205s
6000	608x608	61.00%	236s
6000	832x832	61.80%	314s

Table 1: Accuracy and inference time for relevant cases

We ended up choosing the 6000 iterations with 608x608 resolution for a good trade-off between accuracy and speed. Then, to choose the confidence threshold, let's look at table 2:

conf_thresh	precision	recall	F1-score	avg. IoU
0.20	0.65	0.62	0.63	46.98%
0.25	0.69	0.59	0.59	50.44%
0.30	0.73	0.56	0.64	53.33%
0.35	0.76	0.54	0.63	55.84%
0.40	0.79	0.51	0.62	58.14%

Table 2: Confidence threshold comparison for 6000its 608x608res

We have the intuition that False Positives can have a higher negative impact on the tracking algorithm than False Negatives. The reason for this is that a false detection can create a new track that can then steal detections from real tracks, or put a false detection on a real track. And in the other case, if you lack a true detection for one frame but then get good ones in other frames you can still have enough detections on the tracks to see the real path of a person.

This is just an intuition and needs further testing, but we chose the 0.35 for conf_thresh, having into account the precision and the average IoU covered.

4 Tracking Implementation

For the final implementation with the working python program, we used OpenCV to handle the input/output and to run the detection model. The reason for this was that we already were familiar with OpenCV from the POVa course assignments and because the Deep Learning implementation for the CPU is usually faster than in other frameworks and we don't have GPUs with proper drivers to use.

All code is original from what we already knew or using documentation, except the SORT algorithm, which is from the original implementation [BGO+16]. The code for this project is released on github with a compatible license.

4.1 Experiments and Evaluation

4.1.1 Evaluation Metrics

For evaluation we looked at three metrics of Multi Object Tracking, we computed HOTA[LOD+20], MOTA[BS08], IDF1[RSZ+16] and submetrics.

We gave HOTA the most importance, for the following reasoning: "In comparison it can be seen that the MOTA score is heavily biased towards measuring detection at the expense of ignoring association. In contrast the IDF1 score is heavily biased towards measuring association, at the expense of ignoring detection. HOTA finds a perfect balance between these two extremes by equally weighting both detection and association, while allowing analysis of each component separately with the DetA and AssA sub-scores." (quote from the author of HOTA)

4.1.2 Hyperparameter Tuning

The SORT algorithm doesn't need training, but it has some parameters that can be tuned.

- **max_age** = Maximum number of frames to keep alive a track without associated detections
- **min_hits** = Minimum number of associated detections before track is initialised
- **iou_thres** = Minimum IoU (Intersection over Union) for match

We used the training set from MOT15 ([MOT]) as a validation set to tune the parameters. And tested all possible combinations with the following lists of parameters: max_age[1, 5, 20], min_hits[3, 5, 7], iou_thres[0.2, 0.3, 0.4] for a total of 27 combinations.

Evaluation was done with the official evaluation code from TrackEval [JL20]. But for that, we needed to run our program for each combination, for each sample on the dataset, creating a txt file with results in MOT Format in a certain folder structure. So we made a bash script to automate this process and get all folders ready to just move to TrackEval and evaluate.

In table 3, you can find a compilation of the best combinations of the three hyperpameters and the respective metrics results for each combination.

max_age	**min_hits**	**iou_thres**	**HOTA**	**MOTA**	**IDF1**
1	7	0.3	38.28	43.01	48.836
1	5	0.4	38.687	43.383	48.563
20	3	0.2	41.069	43.097	53.744
20	5	0.2	40.542	43.413	53.585
5	5	0.3	40.555	43.521	52.617
20	5	0.3	40.76	43.446	53.519
20	3	0.3	41.306	43.175	53.72
5	3	0.3	41.164	43.22	52.844

Table 3: Some results of grid search

Taking into account the highest HOTA metric, and second best IDF1, we chose **max_age**=20, **min_hits**=3, **iou_thres**=0.3.

4.1.3 Final Evaluation

For the final testing we had a problem. To avoid bad intentions on the MOT Benchmarks, the test data ground truth is not available to the public. Instead, you can send your results after registration on the MOT webpage and you will get the accuracy metrics. We asked for it, but our registration has not been confirmed and we couldn't use the test set. As a workaround testing was done using a training set from a different dataset, which was MOT16.

Testing the final model on MOT16 we got 37.858 HOTA, 38.783 MOTA and 46.59 IDF1

For speed, on a Ryzen 5 2600 CPU the average FPS were 10 . Profiling one sample, 85% of execution time was spent on the detection step, so we expect great speedup using a GPU for detection.

5 Conclusion

Tracking is already a task within reach, the improvement in faster/better methods for detection, and for tracking in the last years made it better than ever before. It still needs a huge amount of computation, anything bigger than our small model is expensive to deploy, but with improvements in hardware we are not too far away from good enough results.

Our implementation still has a lot of room to improve, and may not be precise enough depending on the use case.

We also learned that fine tuning is powerful, we didn't expect the detection to improve so much. With more training time and a bigger Yolov4 model we could get amazing results.

Sort seems to have problems when people cross each other, not taking into account the velocity/direction of them. DeepSort or a different tracking algorithm that takes these things into account should improve the association score.

References

[BGO+16] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.
github repo: `https://github.com/abewley/sort`
link to original paper: `https://arxiv.org/abs/1602.00763`.

[BS08] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008, 01 2008.

[BS19] Akansha Bathija and Prof. Grishma Sharma. Visual object detection and tracking using yolo and sort, 2019.
`https://www.ijert.org/research/`

visual-object-detection-and-tracking-using-yolo-and-sort-IJERTV8IS110343
pdf

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[img] Tiny-yolo architecture image. https://www.researchgate.net/figure/Architecture-of-Tiny-YOLO-CNN_fig2_338542688

[JL20] Arne Hoffhues Jonathon Luiten. Trackeval. https://github.com/JonathonLuiten/TrackEval, 2020.

[LOD+20] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. Hota: A higher order metric for evaluating multi-object tracking. *International Journal of Computer Vision*, pages 1–31, 2020.

[MOT] Motchallenge - the multiple object tracking benchmark. https://motchallenge.net

[PER] Yolo-performance. https://blog.roboflow.com/scaled-yolov4-tops-efficientdet

[RSZ+16] Ergys Ristani, Francesco Solera, Roger S. Zou, Rita Cucchiara, and Carlo Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. *CoRR*, abs/1609.01775, 2016.

[SPD] Yolo-speed. https://alexeyab84.medium.com/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73

[SZL+18] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. Crowdhuman: A benchmark for detecting human in a crowd. *arXiv preprint arXiv:1805.00123*, 2018.

[tin] Tiny-yolo. https://www.pyimagesearch.com/2020/01/27/yolo-and-tiny-yolo-object-detection-on-the-raspberry-pi-and-movidius-nc

[WB18] Nicolai Wojke and Alex Bewley. Deep cosine metric learning for person re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 748–756. IEEE, 2018.

[WBP17] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.