Vehicle Counting with YOLO and DeepSORT

John Xiang
Stanford University
jxiang16@stanford.edu

Abstract

Vehicle counting provides critical information to traffic analysis problems (like monitoring flow, peaks/jams, etc.) and other related traffic issues like highway management. Having an good and accurate understanding of traffic flow can help for urban planners to be able to make more informed decision to improve the general quality of life for the citizens. While there already exist numerous methods to accomplishing the task already, from manually counting to special censors (and many other alternatives), developing smart systems to automatically count in real time can be incredibly useful and save much time and effort. For this paper, a simple single camera system is assumed such that an approach can be easily be used by most people who own a phone camera, and is able to process recorded videos relatively quickly.

Code for the project can be found at https://github.com/jxiang16/cs231a-final-proj, but since the recordings are of areas nearby, and we didn't censor license plates or anything like that, for the sake of privacy (and the random cars filmed), it is in a private repository, so send an email to request access (it has already been shared with the TA who has shared feedback on past parts of the project).

1. Introduction

The problem at hand here is relatively simple - given a video of ongoing traffic, we want to be able to figure out how many vehicles pass by during it. Ideally, this can run in close to real time and could be used in a common traffic monitoring system with a single or multiple camera setup. However, due to the nature of the class project and personal equipment limitations, we took assumed a single camera output and only dealt with processing recordings quickly, rather than any input streams. Thus, this paper assumes a very simple setup of recording with any sort of single camera from any angle (a phone was used here.

For this paper, the problem of vehicle counting boiled down to being able to track the state of each vehicle throughout a video, and then just counting how many vehicles show up. This involves solving 2 subproblems: object (and in this case, vehicle) detection, and multi object tracking (MOT). There is a lot of past work in each of these subproblems that will be mentioned in the Background/Related Work section. Given the time limitations of the project (especially since we ended up switching project decisions pretty late, due to difficulties directly connecting my past project ideas to course material in a practically achievable manner), We aimed to see what could be used off the shelf, especially for the portions of the work not as directly relevant to coursework, and get a system working, and then attempt to improve portions of it in the remaining time (which there was a bit of). With that in mind, the portion we wanted to focus on the most was the tracking portion, since it was most relevant to class material (as in our optical flow lectures and homework) - we mainly tried to improve this portion of the project, as will be discussed later.

2. Background/Related Work

In past works that aim to count vehicles, there are numerous approaches to doing so. Some past works have attempted to integrate multicamera setups (which are relatively common in traffic surveillance systems), as done by Ciampi et al [2] [1]. There have even been attempts to use audio signals in order to count vehicles with decent success, as done by Djukanovic et al [3].

The approach we ended up taking is most similar to that taken by H. Song et al [5]. In their paper, they develop a system that uses image processing to extract and segment the road surfaces, parts of which are then plugged into their object detector (YOLOv3) to find the vehicles and their bounding boxes in the image. They then plug this into their own object tracking algorithm, which looks to match vehicle feature descriptors to existing detections limited to certain pixel ranges from frame to frame.

We took a similar approach by breaking down the problem into the object detector (also used YOLO) and object tracker, though we took a different approach for solving the latter). For simplicity, we skipped the road

surface segmentation portion, as it seemed mostly as an enhancement, and given the angle/point of view for most of the recordings used in this project (mostly eye-level, rather than having surveillance cameras at high elevations), it would not be of much use.

For object detection, as previously noted, we did end up using YOLO, specifically, YOLOv5 [1]. For a bit of context on it, it was built by Glenn Jocher, who had earlier implemented YOLOv3 in PyTorch - the main reason why we ended up using it over the other YOLO versions was just because it used PyTorch rather than Darknet [2] which was used as the net framework for the other traditional versions of YOLO (v1-v4), and I'm a bit more comfortable with the former in case modifications were necessary - there was no real strong reason, though we did want to particularly use some version of YOLO over other object detection options because it ran quickly and was relatively lightweight. It would be interesting to also test with other versions of YOLO (at least v3 and v4), or other completely different object detectors to see effects on accuracy and runtime.

For the object tracking side, again, there was a lot of options. For example, H. Song et al. took an approach that used ORB as a feature extractor for the vehicles (seemingly chosen for computational performance and matching costs) and calculated a descriptor using BRIEF (Binary Robust Independent Elementary Features) [5]. They use the feature descriptor to match to vehicles (i.e. existing detections), though they simply use a pixel distance threshold T, for which, if the center points of a vehicle object is greater than T between two adjacent frames, they are considered different vehicle. There were also options like the (Extended) Kalman filter we covered in class, and many options to use as feature descriptors for vehicle matching (SIFT, SURF, aforementioned ORB). e looked for something more robust than the pixel distance threshold approach, and eventually came across DeepSORT [10], which builds off the authors' earlier work in SORT [11]. While We did strongly consider doing something like the above (i.e. just looking for a match and using a pixel distance threshold) because it seemed relatively easy to implement, we ended up deciding to use DeepSORT as it seemed more robust and was more directly related to ideas we learned about in class (e.g. Kalman Filters). We think that an approach that uses a basic pixel distance threshold when finding matches comes with the large drawback of likely failing if there is heavy occlusion. For example, if there is a large tree blocking out a portion of the scene such that every car that passes by vanishes for a portion of the camera view would likely surpass these pixel distance thresholds, and fail to count a vehicle as the same vehicle before and after the tree. Of course, DeepSORT has its own parameters for similar ideas (i.e. how long until a track expires), but it seems to be more robust in these situations and probably be much more lenient and flexible to handle cases like that. More details on DeepSORT will be discussed in the following section.

[1] https://github.com/ultralytics/yolov5

[2] https://github.com/pjreddie/darknet

3. Approach

As mentioned in problem statement, the core of vehicle counting involves solving two subproblems: object (in our case, just vehicle) detection, and then object tracking throughout the video. We decided to use YoloV5 for the former, and DeepSORT for the latter, both with a few adjustments. We will go over what each of these does at a high level in the following sections.

3.1. Object Detection (YOLO)

The job of the object detector here is to take the in the input images (frames of the video, in our case), and give us the object classifications and their bounding boxes. A high level flow of what YOLO does is illustrated in Figure 1 [7]. It takes in the input image, divides it into a grid of cells. Each cell in the grid is responsible for predicting a potential object box and class as shown in the rightwards and downwards flows respectively. These predictions are then combined into proposed class bounding boxes, which are then filtered using non-maximum suppression (NMS) and threshold detection to come to the final. There are of course, much finer details, especially with the more recent YOLO versions - more specific details on the latest YOLO architecture and algorithms can be found on the official documentation for YOLOv3 [9] or the original YOLO paper [8].

In our case, since we only care about counting vehicles, we throw out all of the non-vehicle classifications out and only keep the vehicle ones (cars, buses, trucks, motorcycles, etc. (though most except cars and trucks were rare or nonexistent in our data)), and plug in the rest of the information (bounding box/location in original image) into our object tracker. There were not too many modifications made to YOLOv5 aside from excluding non-vehicle classifications and writing some necessary code to integrate it with the object tracker at each frame.

There was no strong reason for choosing YOLOv5 over other YOLO versions - the main motivating factor. It may be interesting to explore using alternative YOLO versions, or even alternative object detectors (e.g. (Fast(er)) RNN, SSDs, etc.) and see how accuracy and runtime can be affected, but that was out of the scope of this project.

3.2. Object Tracking (DeepSORT)

The job of the object tracker here is to take the detections that come from running the object detector on the incoming frames, and be able to match them to existing detected states

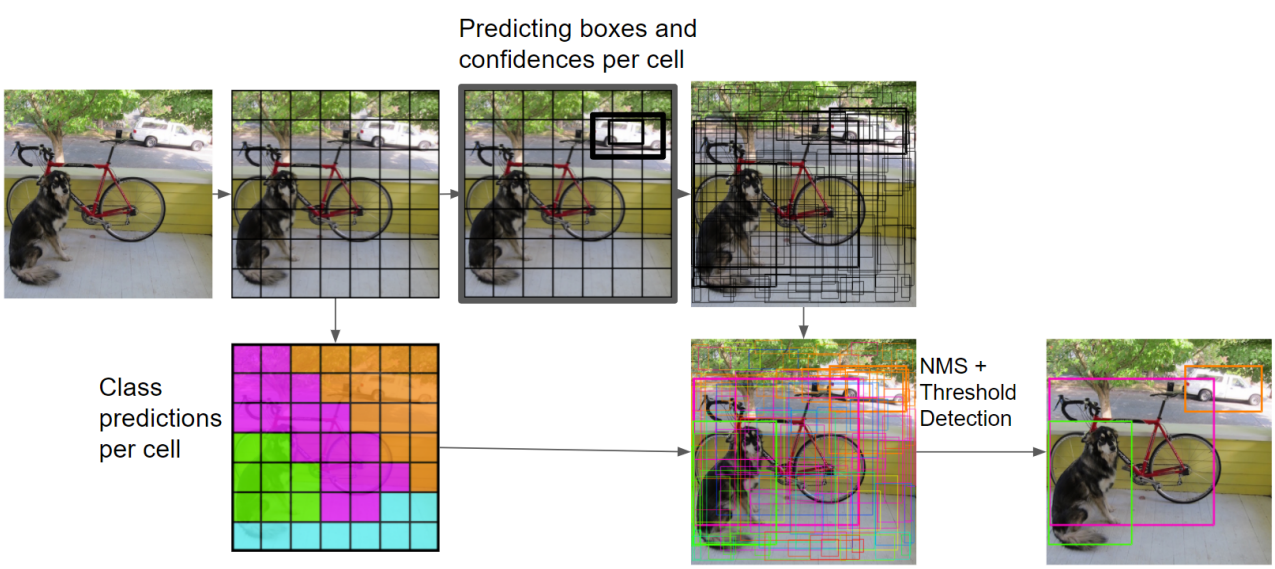

Figure 1. High-level overview of YOLO

Figure 2. Visualization of Object Tracker goal

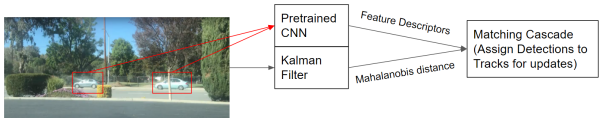

Figure 3. Visualization of portions of DeepSORT

- this state of a vehicle that is updated with every frame it appears in is referred to as a "track," as depicted in Figure 2. Thus, with each frame and its corresponding detections, the tracker must decide what existing track to update, or if there are none, then creating a new track for that vehicle, as it probably hasn't shown up before. The matter of counting vehicles then simplifies to the number of tracks that the tracker is monitoring.

DeepSORT builds off of a Kalman filter - we plug in the bounding boxes from the object detector at each frame into the filter - the Mahalanobis distance between Kalman predictions and new frame data is used as input into their matching/assignment algorithm for new incoming frames - one approach can be to run this using the Hungarian algorithm. However, DeepSORT builds on just using the Mahalanobis distance by also adding an appearance descriptor for each bounding box, and keeps track of recent appearance descriptors for each track. The aforementioned Mahalanobis using the Kalman predictions as well as a cosine distance metric using the feature descriptors are both fed into their matching cascade algorithm that finally assigns new detections to corresponding tracks - this flow is visualized in Figure 3. Specific algorithm details and equations (including how parameters like frame threshold for detection among others) can be found in the original paper [10].

The appearance descriptor comes from a pre-trained CNN that runs on the detected bounding box of the object from the detector. However, the pre-trained model is trained on people re-identification datasets, rather than vehicles. In the interest of time, we went ahead and initially used the pre-trained model anyway, hoping that the results weren't too bad. However, there were obvious loss cases, that will be examined in the later Initial Results section, that we suspect may have been at least partially affected by the fact that the appearance descriptors probably weren't very good for matching vehicles to each other. One part of the DeepSort paper mentions how critical the descriptor similarity can be by itself (especially if the camera is not stationary, as in many of our recordings), so this made sense as a potential area of improvement. As a result, we looked for ways to improve this specific aspect - the 2 main options were to either try retraining the existing CNN on an existing car reidentification dataset, or trying to integrate some other feature descriptor into the DeepSORT system (either an existing one like SIFT, ORB, etc. or even my own). We didn't have a really good sense of time required for either (which was limited), but ended up attempting the former.

In order to try to train a new encoder model, the authors

of the DeepSORT have a separate repository [3] with some example training code they used to train on the people datasets [10]. For the most part, we adapted similar code, but instead trained on the Stanford Cars dataset [6]. Ideally, we probably would have liked to train on a more comprehensive and much larger dataset (e.g. VehicleNet [12] looked promising), but due to hardware limitations (i.e. storage, as training was also taking up a lot of computer space), and in the interest of time, we chose the much smaller cars dataset (it was also conveniently already in tensorflow's datasets [4]). Setting up the training proved to be rather tedious as the existing code did not seem to run on any modern version of Tensorflow (which I already wasn't that familiar with), so I ended up having to modify many parts of it (much of which was migrating to using tf.compat.v1, among several other changes). More details on the training results will be specified in the Experiment section, as I did eventually get my training code at least to run. Other than that, I also tried implementing some parts of DeepSort from scratch, but it proved difficult and probably not worth our team to finish integrating it into the rest of the system code-wise, so at some point we decided to simply use most of the given implementation (after all, there isn't really much variation to Kalman filter implementations anyway...). We played around with tuning some of the parameters (like number of frames needed for detection, how many frames until a track expired, etc.), but within reasonable values, these did not end up changing results much.

4. Experiment

4.1. Experimental Data

The data this was tested on was collected from an iPhone 7+ in areas around San Jose, sometimes recorded by hand (with a slightly moving camera), and other times fixed when a setup was feasible. These were primarily recorded around eye-level (around the same elevation as the cars) because they were recorded by hand or from another car. We attempted to have some level of diversity in traffic/population, occlusion, and distance from camera (which we suspect actually has noticeable effects on the results). Figure 4 provides some examples of the types of scenes these recordings took place. Unlike much of the data of past works, a fair amount of the data is quite up close, introducing another challenge where a single vehicle's appearance can greatly vary as it turns from/away the camera (for example, the front back of cars is very different from side views) - many past works take videos from surveillance cameras on height, where the appearance of the vehicle in the camera's view doesn't change that much,

[3]https://github.com/nwojke/cosine$_{m}etric_{l}earning$

[4]https://www.tensorflow.org/datasets/catalog/cars196

Figure 4. Some examples of still frames from different recordings. These are only a subset of the recordings taken.

especially compared to cases where a vehicle completely turns to or away from the camera in our recordings. Most of the recordings originally ranged from 15 seconds to a minute, though many, but not all, on the longer side were broken up into 5-10 second chunks using FFMPEG in order to make the manual groundtruth labeling easier and more accurate.

We also tried testing on some existing video datasets like GRAM-RTM [4], but we don't seem to be able to download the video data from the site [5] (the linked paper also doesn't seem to load). We had a lot of extra data to use beyond the recordings mentioned in the data table and in the repository, but since we already encountered recurring issues within this small subset, we decided to just focus on those problems.

Example video outputs (with the latest model) can be found in the repository.

4.2. Initial Results and Loss Cases

Due to the nature of the dataset, the only evaluation metric used here is accuracy of predicted number of vehicles. Since the number of vehicles is relatively low for the most part (since we were manually counting to determine groundtruths), both absolute and percentage differences are reported. Both the initial results prior to trying the (probably under-trained) new model as well as updated results afterwards are all in the provided Table 1. There is still a considerable amount of data that can be gone through, however, the issues present in this small subset proved to be present in many other examples as well (just from brief tests), so we decided to just focus on this subset for the sake of having a few decent examples for analysis and testing.

As is evident from the data in the table, the biggest issue that we ran into was significant overcounting in many scenes. The fence cases (as in the bottom right example of figure 4) were an exception to this, as we found out that

[5]https://gram.web.uah.es/data/datasets/rtm/index.html

Recording	Predicted Count	Actual Count	Δ	Updated Count	Updated Δ
low_occlusion_a	24	14	+10 (+71.4%)	22	+8 (+57.1%)
low_occlusion_b	13	8	+5 (+62.5%)	13	+5 (+62.5%)
low_occlusion_c	2	2	0 (+0%)	2	0 (0%)
low_occlusion_d	4	4	0 (+0%)	4	0 (0%)
med_occlusion_a	26	15	+11 (+73.3%)	28	+13 (+86.7%)
med_occlusion_b	7	4	+3 (+75%)	6	+2 (+50%)
med_occlusion_c	2	2	0 (0%)	2	0 (0%)
med_occlusion_d	6	4	+2 (+50%)	6	+2 (+50%)
high_occlusion_a	14	9	+5 (+55.6%)	14	+5 (+55.6%)
high_occlusion_b	23	13	+10 (+76.9%)	20	+7 (+53.8%)
fence_a	0	2	-2 (-100%)	N/A	N/A
fence_b	0	40+	-(40+) (-100%)	N/A	N/A

Table 1. Table of all results. Actual counts were determined by manually counting, so may include human error.

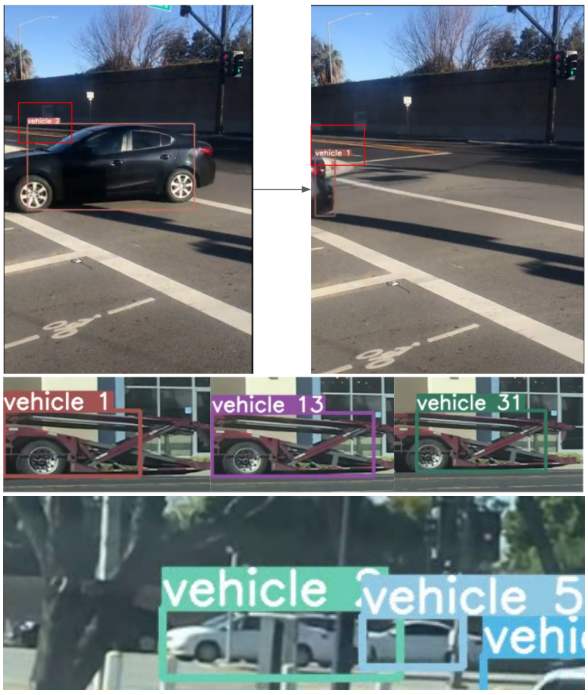

Figure 5. Some failure cases.

the issue there was just that YOLO wasn't able to detect any vehicles in a setup like that where there was a fence between the camera and the vehicles, resulting in 0 detections across the board - we decided not to focus on these exceptional cases.

Figure 5 includes some of the main loss patterns we noticed going through the recordings. In the first example, while the car is accurately tracked going across the camera, in the final frames it becomes part of a new track and is identified as a completely separate vehicle - there were also many other examples of this especially when the camera was close to the traffic and/or the recordings were taken from angles where cars would be turning. In the second example, the exact same part of a truck is re-identified as part of being 3 separate tracks (i.e. vehicles) throughout a roughly 30 second clip - and this issue is was pretty commonplace when there were prominent stationary vehicles in the scene. Even when there is a bit of noise in the detected bounding box (which slightly varies, as in the image), we still should expect that the tracker should be able to figure out that the vehicle is the same. In the last example, it is difficult for YOLO to distinguish between a bunch of vehicles when there is heavy occlusion both amongst the vehicles themselves as well as the terrain - considering that these are also difficult as a human and are difficult for YOLO as well (YOLO struggles if there are many overlapping small objects in a small area as these overlapping cars can often look like), we decided to not worry as much about these cases either.

We suspect that a big part of these problems is because DeepSORT may have trouble identifying that these are the same vehicle when matching the detections to existing tracks - this is particularly difficult in the former case as the butt/front of a vehicle are totally different from side/overhead views, even if we as humans can easily identify they are from the same vehicle. As a result, we decided to try training the CNN used to generate the appearance descriptor on a cars dataset to hopefully try to alleviate some of these issues.

4.3. (Attempt at) Training CNN for Descriptor

For the retraining, we trained on the relatively small Stanford cars196 dataset [6], which only had a bit over 8000 training images. Due to some weird technical issues that we could never figure out (possibly due to some weird Tensorflow version issues), we had to significantly downscale the input images prior to plugging them into the

[6]https://www.tensorflow.org/datasets/catalog/cars196

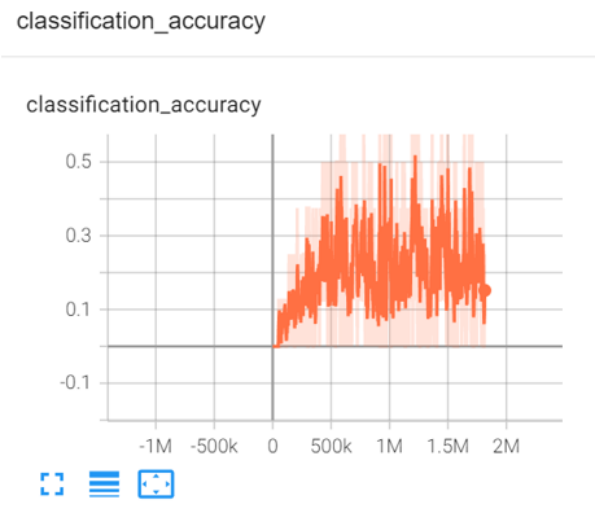

Figure 6. Classification Accuracy - not that great

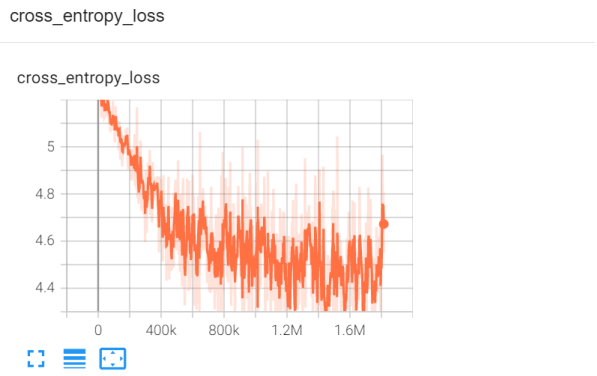

Figure 7. Training Loss - at least it improves!

training workflow. We also were using a pretty small batch size of only 8 due to hardware limitations. Unfortunately, due to time limitations, we (almost certainly) did not fully train the new model. Ideally, we would have given it more time, and if it improve with a bit more time, and the accuracy still wasn't great, then trying to train on a much larger dataset like VehicleNet, which has 434,440 training images of 31,805 classes (though our computer was running out of space due to training) [12]. Some of the training graphs from Tensorboard can be seen in Figures 6 and 7 - as can be seen, while it was certainly improving, it did not reach anywhere close to good accuracies (30-40% classification accuracies, as in Figure 6) after over 12 hours of training (again, would have liked to train for longer even with the unideal setup and technical issues, but we were limited by time). However, we figured it was worth a shot to at least try this new model to generate a different descriptor, as is done for the following results.

4.4. Updated Results

The results using the new model were not much of an improvement over the original results - these are the last 2 "Updated" columns of the table. While there were slight improvements in some recordings (some overcounting was slightly lessened, like low_occlusion_a and high_occlusion_b), overall it did not make a big difference (and there were even cases where the prediction got worse, as in med_occlusion_a), and qualitatively, the issues illustrated in Figure 5 still remained present for most of the tested recordings - DeepSORT struggled to identify that vehicle detections of the same vehicle were actually the same vehicle and belonged to the same track. We suspect that the descriptor is still not very useful in terms of given the poor training results, but hypothesize that if we had well trained robust model (particularly when it comes to identifying the same vehicle from multiple angles), the descriptor would prove much more accurate and the matching would be more successful. The example output recordings provided in the repository use the updated model.

5. Conclusion

This was a big learning experience for me in many ways as this is one of our first times starting a project by using a lot of implementations "off-the-shelf" (YOLOv5 and lots of DeepSORT). There was a surprising amount of work needed to actually get these up and running in a way that was needed as well as putting them together. In particular, there was a lot of tedious work involved in actually getting my training code to even run due to a ton of version incompatibilities and the training code not having a requirements.txt or some equivalent. I also ran into storage/RAM limits on Colab (as well as writing videos taking awhile to load on drive) and didn't want to upgrade just for this project, so I ended up running and training on my own hardware - fortunately my computer isn't bad, so this wasn't too much of an issue once CUDA and CUDNN were setup (aside from running out of disk space during training).

There are a lot of directions one could take this for future work. One obvious way is to properly and fully train a CNN to generate a better vehicle feature descriptor on a vehicle dataset - the (probably incomplete) one used in this paper shows some promise that having an improved descriptor can improve accuracy, but there are a lot of areas of improvement, from training on a larger more comprehensive dataset (like the aforementioned VehicleNet), to fixing specific parts of training (e.g. not doing the significant downscaling prior to training for the input images would probably allow help with better accuracy even with a relatively small dataset), performing

additional data augmentation, etc. Due to the nature of the project, it would also be interesting to replace certain parts and replace them with alternatives, like replacing the object detector (YOLO) with something different like Faster RCNN or a single shot detector (where there is likely to be some runtime tradeoff for accuracy), or implementing an alternative multi object tracking algorithm (like that done by H. Song [5]). The general core of combining a Kalman Filter with a vehicle feature descriptor for matching detections to tracks also still intuitively seems promising and robust over many alternative, possibly more naive strategies, despite the lackluster results in this paper.

Thanks for reading!

References

[1] L. Ciampi, C. Gennaro, F. Carrara, F. Falchi, C. Vairo, and G. Amato. Multi-camera vehicle counting using edge-ai. *ArXiv*, abs/2106.02842, 2021.

[2] L. Ciampi, C. Santiago, J. P. Costeira, C. Gennaro, and G. Amato. Unsupervised vehicle counting via multiple camera domain adaptation. In *NeHuAI@ECAI*, 2020.

[3] S. Djukanovi'c, J. Matas, and T. Virtanen. Robust audio-based vehicle counting in low-to-moderate traffic flow. *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1608–1614, 2020.

[4] R. Guerrero-Gomez-Olmedo, R. J. Lopez-Sastre, S. Maldonado-Bascon, and A. Fernandez-Caballero. Vehicle tracking by simultaneous detection and viewpoint estimation. In *IWINAC 2013, Part II, LNCS 7931*, pages 306–316, 2013.

[5] H. L. Z. D. Huansheng Song, Haoxiang Liang and X. Yun. Vision-based vehicle detection and counting system using deep learning in highway scenes. 11(51), 2019.

[6] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. pages 779–788, 2016.

[8] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[9] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[10] N. Wojke and A. 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.

[11] N. Wojke, A. Bewley, and D. 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.

[12] Z. Zheng, T. Ruan, Y. Wei, and Y. Yang. Vehiclenet: Learning robust feature representation for vehicle re-identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.