PyTorch implementations and benchmarking of 2019 AI CIty Challenge models - using enriched labelsets for vehicle object detection

CS230 Project Report: Computer Vision; Koen Frankhuizen {koenfr}; https://youtu.be/Qcno_unfl0E
Github: https://bitbucket.org/koen_private/cs230_vehicle_tracking/src/master/

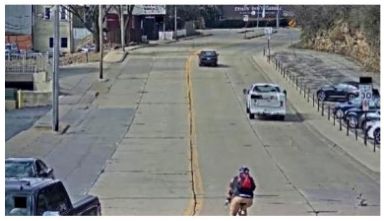
Abstract
This project focusses on object detection and object classification, inspired by the previous year work of Z. Tang, et all [2] on vehicle tracking. For this work, the focus lies on a Pytorch implementation of the object detection and object classification part, an essential preparation set for vehicle tracking. Two models were implemented: a YoloV3 model and an SSD model, with VOC pretrained weights. The data of study was the AI City Challenge 2019 dataset. Over 400 images were relabelled to improve training data quality. After hyperparameter tuning for the new training set, performance of both models was compared based on IDF1 and mAP score. The YoloV3 trained up to a F1 score of 0.67 with a mAp of 0.55 and the SSD model scored a IDF1 of 0.56 with a mAP score of 0.22. Results are benchmarked against the AI City Challenge 2018 results (IDF1). The SSD model was shown to have trouble with classification, whereas the YoloV3 model classified very well. The YoloV3 model is chosen for future work.

Introduction
The project is focused on building Pytorch object detection and classification models – YOLOv3 and SSD that will be benchmarked against the 2019 AI City Challenge (AICC) dataset. As the AICC dataset has only limited ground truth labels, training will be done on pretrained models. The trained models will then be run on a random sampling of vehicle images from the AICC for validation and comparison using the identification F1 score (IDF1).

This project is interesting as it enables us to train popular object detection & classification models on an existing, large, dataset and evaluate their performance on a dataset that reflects real-world images. The AICC challenges to tackle image reidentification without license plates – object detection and classification is therefore an essential pre-processing step before feeding a Siamese network ([1]). This project will focus on the detection and classification. The YoloV3 approach used by the 2018 winners **[2]** formed the inspiration / motivation behind this project. My goal is to implement the YoloV3 object detection and classification in Pytorch and compare it with a SSD implementation. The 2018 City Challenge results form a benchmark for my performance.

Dataset
2019 AI City Challenge Track 1 dataset includes 3.25 hours of videos from 40 cameras across 10 intersections. For this project, three of these intersections have been selected . The 2019 AICC dataset came with only a few ground truth vehicles labelled per image, which was not sufficient for vehicle object detection. 400 images were therefore hand labelled with vehicle bounding boxes to capture all vehicles. The dataset captures three different scenes, all based on CCTV images:





For the training data, I combine images from two vehicle datasets – the hand labelled AICC dataset and the NEXET dataset. By using a pretrained model feasible results can be acquired even with the limited dataset.

Split train/dev/test: The dataset used was small. Therefore, a minimal set was reserved for validation (10% and the set was only split into a train and validation set - the main goal of the project was model comparison.

Training Set extension NEXET or VOC pretrained? The NEXET data set consists of over 50,000 images in a variety of lighting and weather conditions: day time (~50%), night (~46%), and twilight (~4%). Day images will align most closely to the current dataset. Next to this, VOC pretrained weights were downloaded and tested for training.

*Labelling***:** The dataset includes 5 vehicle categories: car, van, pickup-truck, truck, and bus which is consistent with the labelling of the AI City Challenge dataset. Although the labelling supported 5 different classes, two classes were dominant: cars (62%) and pick-up trucks (30%). Therefore the resulting class accuracy was low, in particular for the rarely occurring classes. This will be further discussed in the discussion section.

Approach to relabelling (using image 99 from the AI City Challenge as example):

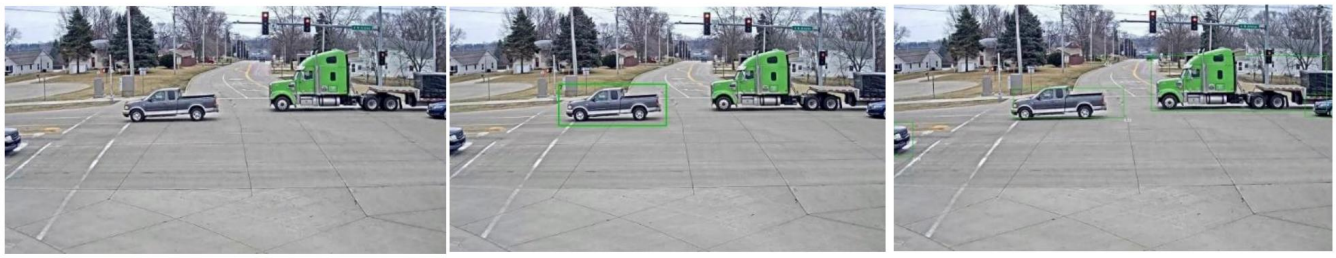


Figure 1:Sample image of the raw file (left) and the ground truth file (middle) and final, labelled file (right).

Green boxes form extra manual labelling. After the manual labelling, I manually added the labels. Each of this cars will be, if known labelled with a type (in this example: [?, pick-up, truck])

Model architecture search and hyperparameter optimization:

YoloV3 and SSD

Two models were picked for comparison, the YoloV3 model (with Darknet53 base) and an SSD model (with VGG16 base). The baseline was set by an SSD run with input size 224, learning rate 1e-4 and 20 epochs.
The models were implemented locally in Pytorch ([5] and [6]) and adjusted to support the different training images and a hyperparameter search. The SSD model was self-implemented whereas the YoloV3 model is a direct implementation from the GIT -with some small adjustments for the hyperparameter search. Although YoloV3 and SSD are commonly used / well known model architectures, a short overview of their architecture is presented here:

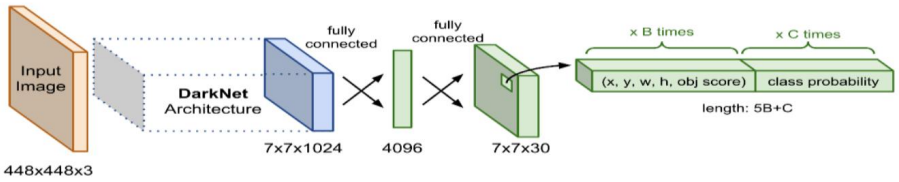


Figure 2: yolo architecture (image from https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html)

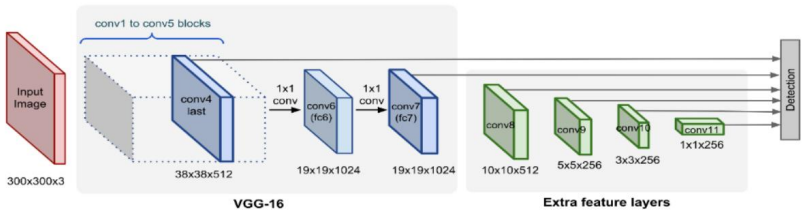


Figure 3: SSD architecture (image from https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html)

Loss functions:
Both models use a loss build up from two parts: a localization loss and a classification loss. For YoloV3, both losses are computes as a sum of squared errors with scale parameters to balance prediction of objects and boxes without objects for the localization loss.

$$\mathcal{L}_{\text{loc}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$\mathcal{L}_{\text{cls}} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \big(1_{ij}^{\text{obj}} + \lambda_{\text{noobj}}(1 - 1_{ij}^{\text{obj}})\big)(C_{ij} - \hat{C}_{ij})^2 + \sum_{i=0}^{S^2} \sum_{c \in \mathcal{C}} 1_i^{\text{obj}} (p_i(c) - \hat{p}_i(c))^2$$

Figure 4: YoloV3 loss (from from https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html)

SSD uses a different approach: a L1 loss for the localisation loss and softmax loss (over the multiple classes for the classification loss.

$$\mathcal{L}_{\text{loc}} = \sum_{i,j} \sum_{m \in \{x,y,w,h\}} 1_{ij}^{\text{match}} L_1^{\text{smooth}}(d_m^i - t_m^j)^2$$

$$L_1^{\text{smooth}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

$$\mathcal{L}_{\text{cls}} = -\sum_{i \in \text{pos}} 1_{ij}^k \log(\hat{c}_i^k) - \sum_{i \in \text{neg}} \log(\hat{c}_i^0), \text{ where } \hat{c}_i^k = \text{softmax}(c_i^k)$$

Figure 5: SSD loss (from from https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html)

Upfront, whereas SSD is known to deliver very accurate results, SSD is know to be less efficient in detecting small sized objects compared to YoloV3**[8]** because it only uses the top feature layers for object detection.

Hyperparameter tuning
The hyperparameter tuning focussed on three hyperparameters: the input size of the image, the learning rate and the number of epochs used for training. These specific hyperparameters where chosen because the I implemented existing model architectures (SSD & YOLO) which are extensively tuned and optimized - tuning the other hyperparameters of the network therefore is not likely to be an efficient time investment.
Epochs: The number of epochs is important because we only have a small training set to (re-)train the network on, therefore the network will be prone to overfitting.
Input size: expected is that increasing input size will always increase performance (for IDF1 metric). For this project we have not chosen a satisfying metric such as detection speed - which might lead to a tradeoff between (small/large) improvement for the optimizing metric versus small/large improvement for the satisfying metric.
Learning rate: learning rate is one of the central hyperparameters to tune.

The hyperparameter tuning was done in a two step approach:
(1) Wide range, random search*:
- image size = [224, 320, 416, 608, 800, 1024]; learning rate = [5e-3, 1e-3, 5e-4, 1e-4, 1e-5, 5e-5]; number of epochs = [5, 10, 20, 40, 60, 100]
* due to time constraints, the random search has been performed on a limited training set. Of course, the results might not be fully representative. The grid search is performed on the full training set.
(2) A small sized range grid search for final tuning
image size = [320, 416, 608]; learning rate = [1e-3, 1e-4, 1e-5]; number of epochs = [20, 60, 80]

Final optimal architecture is

Model	Number of epochs	Input size	Learning rate
SSD-baseline	20	224	1e-4
SSD-optimized	60	608	1e-4
YoloV3-optimzed	26	416	1e-4

Table 1: model parameter choices. Input size 608 for YoloV3 lead to almost no improvement versus much longer train time.
I used early stopping for Yolov3.

Training
Training was first tested on both the Nexet dataset and with use of pretrained weights. The Nexet dataset lead to very slow convergence for the SSD model as well as low (+/- 10%) IDF1 score on the AI City Challenge based dataset – the image set was not suitable for transfer learning. Therefore, early in the process I decided to use pretrained VOC weights instead.

Results
Quantitative approach
All models converged well, with the YoloV3 showing continues convergence but overfitting for high epochs. Below, the loss is plotted versus the number of epochs for each of the models:

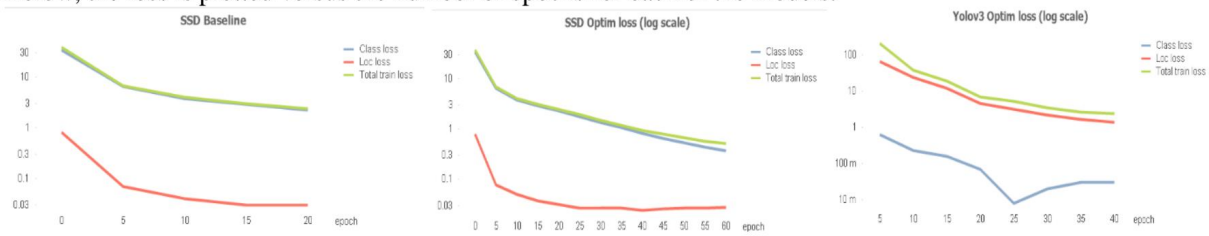


Figure 6 (i – iii): SSD baseline loss, SSD Optimized loss and YoloV3 optimized loss (versus number of epochs)

Final results for the three architectures are

Model	IDF1 score (test)	mAP score (test)	IDF1 score (train)
SSD-baseline	0.44	0.03	n/a
SSD-optimized	0.56	0.22	0.62
YoloV3-optimzed	0.50	0.48	0.9

Table 2: model performance (IDF1 score and mAP score – see Annex).

The results show that the YoloV3 model trained reasonably well on the new dataset (**[3]**) but is still strongly overfitting. The SSD results lag behind research **[4]**. Longer training did not lead to improvement on the IDF1 or mAP score, raising the suspicion that the dataset was to small to train the SSD model properly. From a bias/variance perspective, bias is an issue for both models (training performance is not excellent) but the YoloV3 model also suffers from a high variance.

The mAP score is shown to be important to include, because that metric does not only penalize for bad detections but also included the classification quality. However, the IDF1 score of the SSD model shows that the model is able to detect well and improvements should focus on the classification part.

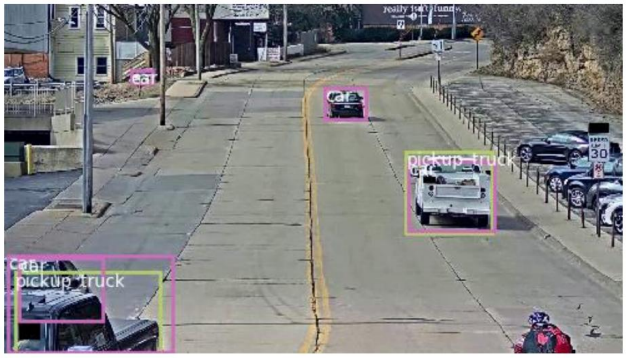
Qualitative analysis of results
Valid detections for each detection show that the SSD 608 much more accurately captures the vehicles compared to the SSD 224:

Figure 7 (i,ii,iii) valid detection results for baseline (left), SSD (middle) and Yolo (right)

YoloV3 also performed reasonably well on the other two scenes; example detections:

Figure 8 (i - iii): different YoloV3 results

However, sometimes the YoloV3 model shows to many boxes per object. Note that the models were trained not to detect the parked cars on the side for the right scene ->



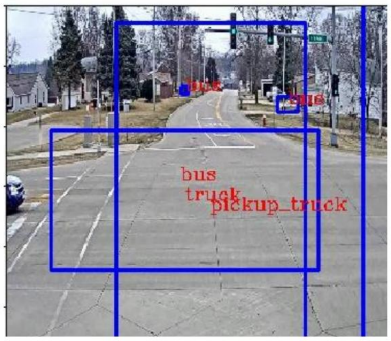


Figure 9: mislabelled SSD output

<- The SSD model sometimes had problems with both labelling and supressing unlikely detection boxes. An example which I tried to improve by hyperparameter training but still occurred in my final model, was the following detection, were the objects are obviously mislabelled but also three large detection boxes remain, which more seem to capture the background.

Conclusion and discussion

Model comparison:
Both models were able to converge well and show reasonable results. The YoloV3 performed better, in particular for classification, and trained much faster. An issue with the SSD model for the chosen training set is the small size of the training set – SSD has very complex architecture and therefore needs a lot of data to train properly on the new set (although for both models, pretrained weights were used).

Classification issues and overfitting:
For YoloV3, the decrease in test set performance for high epochs is also most likely due to overfitting to the small training set – using early stopping enabled the model to adapt to the new data but not to much, but did not resolve the issue of overfitting completely – the IDF1 score still shows a much higher performance for the train set compared to the dev/test set.
Future work should first of all use a far more extensive training set, and a training set which is much more balanced in classes. Busses were never detected, simply because they only occurred twice in the training set and therefore the 'optimal' prediction for the models was to never predict a bus.
Although the SSD model was extensively debugged, the much lower performance on classification is worrisome and an specific issue with my implementation cannot be excluded. In particular, some images (such as image 6) showed that the model mislabelled and sometimes unlikely boxes were not sufficiently supressed.

Faster-RCNN implementation: For this project I tried to implement a Faster-RCNN as well but I did not manage to get that model training on my custom dataset – for a more elaborate comparison a Faster-RCNN should be included in future work. Faster-RCNN can deliver high quality results (but slower training and detection).

Overall conclusion: Overall I did not manage to 'beat' the AI City challenge 2018 baseline set by the used models (YoloV3, SSD and Faster-RCNN). The AI City challenge contenders show benchmark F1 scores over 86% **[9].**
Note that the winners **[2]** relabelled and augmented a large dataset.
Most likely I did not meet the benchmark is because my training set was much to small – a training set of 400 images is small for Computer Vision problems. Time constraints for manual labelling prevented me to label more within this project scope. However the models did adapt well to the new input images and provided reasonable results, in particular for the IDF1 score, and therefore this project would form a good basis to build forward on. My choice of model would then, based on the comparison in this work, the YoloV3 model, based on the much higher mAP score. Noted must be that a comparison on a much larger dataset could lead to different results.

Annex: evaluation scores

The main metric chosen is the IDF1 score, aligning with the AI City Challenge metrics. The IDF1 score will give a good sense of the quality of object detection, however it does not penalize for bad classification score. Therefore, I also include the mAP score to have a better view of the overall performance of the models. The IDF1 score and the mAP score together provide insight into were the model performs well and were not.

Main evaluation Metric: (ID)F1 score*:* I will use the identification F1 score to measure the performance of our models. The identification F1 score is an extension of the regular F1 score, adjusted to measure the performance on vehicle identification:

$$\text{IDF1} = \frac{correct\ id\ detections}{avg.\#\ of\ ground{-}truth\ and\ computed\ detections} = \frac{2\text{IDTP}}{2\text{IDTP} + \text{IDFP} + \text{IDFN}}$$

IDF1 is build up by calculating the True Negative ID (IDFN), True Positive ID (IDTP) and False Positive ID (IDFP). This minimizes the cumulative false positives and false negatives and the overall costs of mis assigning.

Supporting evaluation metrics: The IDF1 allows for ranking of the overall performance (balancing recall and precision), but the precision (IDP) and recall (IDR) can provide insight into tracking trade-offs:
$\text{IDP} = \frac{IDTP}{\text{IDTP} + \text{IDFP}}$, $\text{IDR} = \frac{IDTP}{=\text{IDTP} + \text{IDFN}}$

mAP [7]:
Another common used metric for detection algorithm is the Mean average precision (mAP) score that is calculated as the average precision (AP) over all classes and/or all correctly (by IoU) boundary boxes, with $AP = \int_0^1 p(r)\, dr$.

References

[1] Xinchen Liu, Wu Liu(B), Tao Mei, and Huadong Ma, A Deep Learning-Based Approach to Progressive Vehicle Re-identification for Urban Surveillance

[2] Z. Tang, G. Wang, H. Xiao, A. Zheng, J.N. Hwang, "Single-camera and Inter-camera Vehicle Tracking and 3D Speed Estimation Based on Fusion of Visual and Semantic Features", CVPR Workshop (CVPRW) on the AI City Challenge, p 108-115, 2018.

[3] Joseph Redmon, Ali Farhadi "YOLOv3: An Incremental Improvement", 2018, https://pjreddie.com/media/files/papers/YOLOv3.pdf

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian zegedy, SSD: Single Shot MultiBox Detector,

GIT Implementations and non-scientific references:
[5] https://github.com/parlstrand/ml_playground/blob/master/computer_vision/object_detection/ssd.ipynb

[6] Erik Lindernoren, https://github.com/eriklindernoren/PyTorch-YOLOv3

[7] https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3

[8] https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d

[9] https://www.aicitychallenge.org/2018-winners/