Palo Alto Streets Vehicle Detection and Tracking

Yan Wang
Stanford University
yan12@stanford.edu

Fangran Wang
Stanford University
frwang@stanford.edu

Shanduojiao Jiang
Stanford University
sj99@stanford.edu

June 3, 2022

Abstract

This is the final report for CS231N Deep Learning for Computer Vision, Spring 2022. In this project, we proposed the study of object detection and tracking problem for autonomous vehicles. Specifically, we investigated and compared several YOLO model variants [11] on Udacity Self Driving Car Dataset [13]. Then, we collected and annotated our own driving scene dataset in Palo Alto, and ran YOLOv5 [8] on our own custom dataset. Lastly, we performed transfer learning on our Palo Alto dataset starting with the model weights pre-trained on the COCO dataset [9]. For the Udacity Self Driving Car Dataset, we did not see a significant difference of performance among YOLO model variants. However, for the Palo Alto dataset, we have witnessed a significant improvement on the performance after transfer learning.

1 Introduction

Recently, the surge in the number of intelligent vehicles has revolutionized people's everyday commute. Some of the most fundamental problems for such vehicles include lane detection, semantic segmentation, and object detection. In this project, we will investigate object detection for its importance in ADS (autonomous driving systems), and its intricacy of real-time scene understanding and video data input. With an attempt to better serve the Stanford/Palo Alto community, we are specifically interested in developing our models using real-life driving scene data collected in Palo Alto. We have divided our project into three phases:

- **Phase 1: Object Detection with Existing Dataset:** For this phase, we used the Udacity Self Driving Car dataset [13] to train and compare different YOLO model variants, including YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and our custom YOLOv5 models. [11] The input to our model is an image with objects such as cars and pedestrians, and the output of our model would be the predicted bounding boxes and confidence scores of the objects. After we have trained the models, we converted a driving video taken in Palo Alto to a non-continuous, digital format - a.k.a. frames - and tested the model performance on these frames.

- **Phase 2: Custom Data Collection and Annotation:** Two data collection methods were used: the first one is taking pictures of the streets in Palo Alto from the passenger seat, and the second one is converting the videos taken from dashboard cameras into frames. After the data was collected, we used Roboflow [1] to manually draw the bounding boxes for vehicles present in each image. For this phase, we have collected and annotated 251 images in total.

- **Phase 3: Transfer Learning on Custom Dataset:** After we have the custom dataset ready, we performed transfer learning using

YOLOv5s [8] with weights pre-trained on COCO dataset, and reran the model on the Palo Alto driving video mentioned in Phase 1. We have noticed a significant improvement in the model performance on this video specifically.

2 Related Work

Object tracking has long been a heated yet challenging problem in computer vision for its difficulties in tracking objects under conditions such as abrupt object motion, changing appearance patterns of both the object and the scene, nonrigid object structures, object-to-object and object-to-scene occlusions. There are two main areas for object detection and tracking: motion-based methods and appearance-based methods. Motion-based methods usually require motion sensors in addition to visual inputs, and thus will not be the focus for this project. Appearance-based methods, however, can detect objects directly from frames and are more widely used in object tracking for intelligent vehicles.

To correctly track the moving objects, the model would first need to detect objects and their positions in each frame of the video. Earlier works include "Viola-Jones detector", [15] which goes through all possible locations and scales in an image to see if any window contains a human face. HOG (Histogram of Oriented Gradients) Detector [4] is another one of the successful feature extraction tools for pedestrian detection. With the development of neural networks, RCNN [7] was introduced for object detection. It starts with the extraction of a set of object proposals (or bounding boxes) by selective search, and then each proposal is rescaled to a fixed size image and fed into a CNN model trained on ImagNet [5] to extract features, and lastly, linear SVM classifiers are used to predict the presence of an object within each region and to categorize the objects. Later on, Fast RCNN [6] was introduced to reduce the training time by simultaneously training the detector and the bounding box regressor under the same network configurations. Faster RCNN [12] further improved the efficiency of the networks by introducing Region Proposal Network that enables nearly cost-free region proposals.

As mentioned before, our project focuses on YOLO (You Only Look Once)[11] - one of the most popular and successful models for object detection. YOLO frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. Unlike previous methods which include multiple neutral networks, YOLO uses one single convolutional network to simultaneously predict multiple bounding boxes and the corresponding class probabilities, which produces fast and accurate results.

After the objects and their bounding boxes are predicted, the system needs to connect the objects in each frame to form a continuous result. To tackle this problem, Bewley et al. proposed SORT (Simple Online and Realtime Tracking)[2], which is based on Kalman filter [18] and the Hungarian algorithm [17]. Later, to enable the system to track objects through longer periods of occlusions, Wojke et al. proposed DeepSORT [19] that incorporates a convolutional neural network which has been trained to discriminate pedestrians on a large-scale person re-identification dataset. For this project, we noticed that simply stitching the predicted frames back together would generate a good result, and thus we did not incorporate SORT or DeepSORT.

3 Methods

Our goal in the object tracking task for this project is to estimate object tracklets for "Car" and "Pedestrian", in light of the abundance of labels in these two classes. Specifically, we expect to produce rigid bounding boxes confining the detected objects. Therefore, the inputs to our model are labeled street images and driving-video frames, and the outputs are multiple parameters of the bounding boxes that describe the positions and probabilities of predicted object categories. To achieve our goal, we use YOLO as our main model architecture, and we build our project on top of existing codebase from

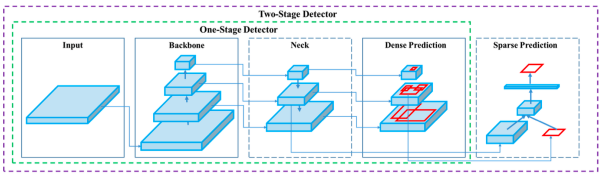

Figure 1: YOLOv4 Object Detector[3]

Ultralytics.[14]

3.1 YOLO Architecture

You Only Look Once (YOLO) v5[8] is adopted in the current project. As a single-stage object detector, the deep-learning architecture contains three important parts: Backbone, Neck and Head. The backbone layers act as feature extractors. In YOLO v5, the CSP (Cross Stage Partial) Networks,[16] which have shown significant improvement in processing time with deeper networks, are used as a backbone to extract rich in informative features from an input image. Model Neck is mainly used to generate feature pyramids, which help models to generalize well on object scaling and to identify the same object with different sizes and scales. SSP together with PANet [10] are used as the neck to get feature pyramids. The model head, which is the same as the previous YOLOv3 and YOLOv4, is mainly used to perform the final detection part. It applies anchor boxes on features and generates final output vectors with class probabilities, objectness scores, and bounding boxes. Since YOLOv5 leverages similar architecture as YOLOv4[3], an illustration of YOLOv4 is shown in Figure 1.

In YOLOv5, Leaky ReLU is used as the activation function in hidden layers and Sigmoid is used as the activation function in the final detection layer. To calculate the model loss, which is a compound loss calculated based on objectness score, class probability score, and bounding box regression score, we use binary cross-entropy with logits loss function from PyTorch.

To gain a better understanding of YOLOv5, it is helpful to understand the previous versions of the YOLO model, which boost the speed and accuracy of real-time object detection[11]. The YOLO system divides the input image into an $S \times S$ grid. The grid cell that contains the center of the object is responsible for detecting the object. Each grid cell can predict B bounding boxes and their confidence scores. And each prediction of a bounding box is defined by x, y, w, h and confidence. The (x, y) is the coordinate of the box center relative to the bounds of the grid cell. The w and h represent the weight and the height of the bounding box.

For our Phase 1, we trained YOLOv5s (small), YOLOv5m (medium), and YOLOv5l (large) with the difference being model depth multiple and layer channel multiple. YOLOv5 models have about 25 hyperparameters for different training settings. We used the default values which are optimized for YOLOv5 COCO model.

We then trained a custom YOLOv5 based on YOLOv5l but it uses Adam instead of SGD as the optimizer. We also trained a custom YOLOv5 model which not only uses Adam, but also has different depth and layer multiples. The detailed difference is shown below in Table 1.

	Small	Medium	Large	Custom1	Custom2
Depth	0.33	0.67	1.0	1.0	1.4
Layer	0.50	0.75	1.0	1.0	1.3
Optimizer	SGD	SGD	SGD	Adam	Adam

Table 1: Difference between YOLO Model Variants

3.2 Transfer Learning

Since our custom dataset only contains 251 examples, with details explained later in the Dataset Section, we believe training the model with transfer learning is most suitable to achieve our goal. We first edited the data configurations file which describes the dataset parameters to specify the paths to our custom train, validation, and test datasets, the number of classes, and the names of the classes in the

same order as their index. Then, we initialized the model with weights from a pre-trained COCO model by modifying the "weights" argument. We preserved the same backbone as the pretrained COCO model, which consists of 12 layers, by specifying the "freeze" argument, and only trained the model's head. Then, as an attempt to achieve meaningful improvements by incrementally adapting the pretrained features to the new data, we performed fine-tuning, which consists of unfreezing the entire model obtained earlier, and re-training it on our custom data with a very low learning rate. Once we obtained satisfying training performances, we further boosted the predictions accuracy by applying test-time augmentations (TTA) - each image is being augmented by performing horizontal flip and applying 3 different resolutions, and the final prediction is an ensemble of all these augmentations.

3.3 Loss and Evaluations

As mentioned briefly before, YOLOv5 loss function is composed of three parts and computed as follows:

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{ classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{1}
$$

1. box_loss: the Mean Squared Error that describes the bounding box regression loss.
2. obj_loss: the Binary Cross Entropy that describes the confidence of object presence. This is also called objectness loss.
3. cls_loss: the Cross Entropy that describes the classification loss.

The loss function consists of three parts: where λ_{coord} is the parameter to increase the loss from bounding coordinate predictions and λ_{noobj} tries to decrease the loss from confidence predictions for boxes that don't contain objects. The first two term represent the box loss. The following two terms denote the object loss (C denotes confidence). The last term is the classification loss.[11]

To measure the object detectors quantitatively, we use Average Precision (AP) for each class and mean Average Precision (mAP). With Precision (which measures how much of the bounding box predictions are correct), Recall (which measures how much of the true bounding boxes were correctly predicted), and IoU (Intersection over Union) values for each class, we calculate the AP for each class. Specifically, AP@.5 means the average precision with the IoU threshold equals 0.5. AP@[.5:.95] means the average AP over different IoU thresholds, from 0.5 to 0.95 with a step of 0.05. Then, we get mAP by taking the average of AP values over the 11 classes. We will mainly use these two metric in our experiments to compare the performance of different models.

4 Dataset

For our project, we have three datasets:

1. **Palo Alto Driving Video:** Throughout this project, we have one driving video taken in Palo Alto for testing the object tracking performance of our models. The video, which has around 20 seconds, is preprocessed and broken down into 603 frames during testing, and stitched back together after the models predict the objects and their bounding boxes in each frame.
2. **Udacity Self Driving Dataset:** We use the Udacity Self Driving Car Dataset re-labeled by Roboflow to train the models in Phase 1. The dataset contains 15,000 images (with size 1280 x

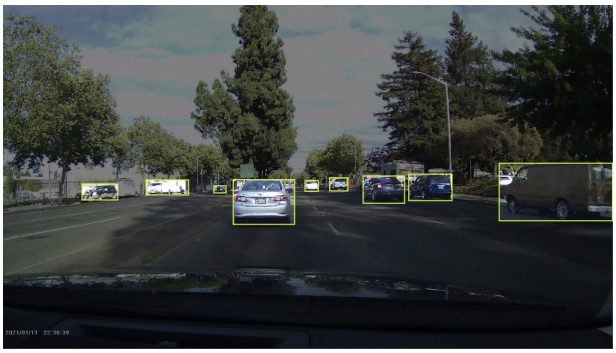

Figure 2: An example of annotated Palo Alto Dataset

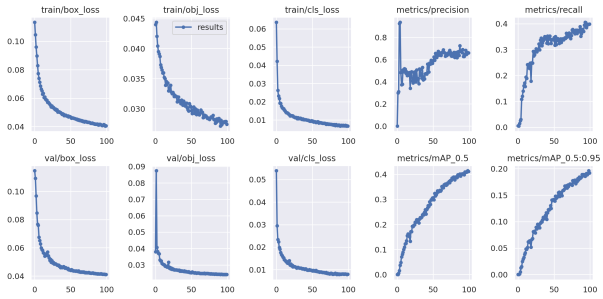

Figure 3: YOLOv5s Result

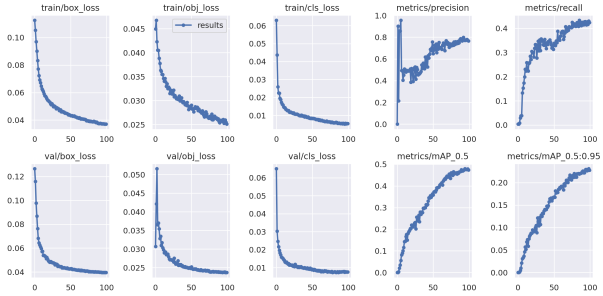

Figure 4: YOLOv5m Result

1280) with 97,942 labels across 11 classes. The 11 classes includes car, pedestrian, biker, truck, and 7 different types of traffic lights. The labels contain object class information and are formatted to align with YOLO annotations: Object Class ID, Object Center X Coordinate, Object Center X Coordinate, Bounding Box Width, Bounding Box Height.

3. **Palo Alto Custom Dataset:** Since the goal of our project is to detect and track vehicles, we only annotate cars in our custom dataset. There are a total of 251 annotated images in our custom dataset, and an example is shown in Figure 2. As mentioned above, we used two data collection methods for creating our own custom dataset. The first one is taking driving scene pictures of the streets in Palo Alto from the passenger seat, and the other one is converting the videos taken from dashboard cameras into frames. To make our models more robust, we specifically included pictures of occluded vehicles, images with only parts of vehicles present, images with no interested objects, and images with crowded vehicles. Since Roboflow supports exportation of the annotated data into YOLO format, we used Roboflow to manually draw the bounding boxes of the cars present in each image in our custom Palo Alto dataset. We divided the dataset into training set (70%), validation set (20%), and test set (10%) for transfer learning.

5 Results

5.1 Results of YOLO Variants

The Udacity Self Driving Car Dataset contains 15,000 images, but due to the limit of computational power, we used a subset of 2000 images for training, 500 images for validation, and another 500 images for test. We trained the baseline models from scratch for 100 epochs. For the optimizer, we used the default SGD (learning_rate = 0.01, momentum = 0.937, weight_decay = 0.0005) for all runs. Here we present the results for YOLOv5s, YOLOv5m, and YOLOv5l in Figures 3, 4, 5. We observe that mAP does get slightly better as the model becomes larger and more complicated, so the results align with our expectation.

To further improve the results, we trained two

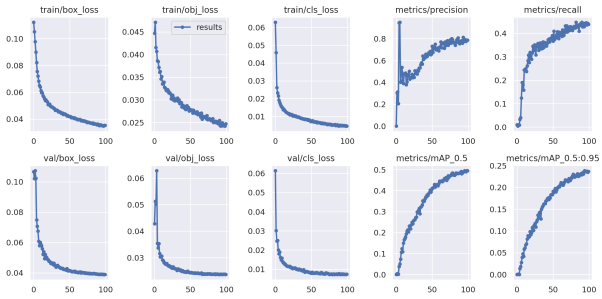

Figure 5: YOLOv5l Result

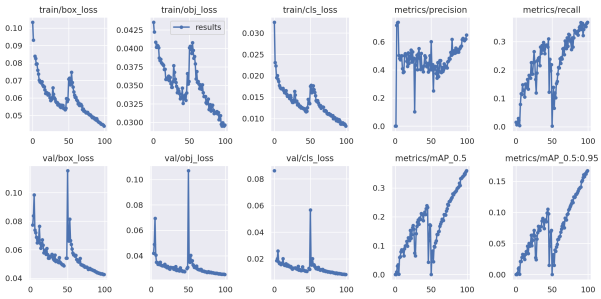

Figure 7: Custom2 Result

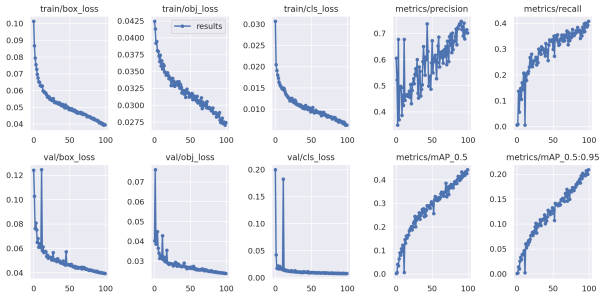

Figure 6: Custom1 Result

Model Name	AP@.5	AP@[.5:.95]
YOLOv5s	0.755	0.423
YOLOv5m	0.765	0.446
YOLOv5l	0.776	0.455
custom1	0.775	0.447
custom2	0.740	0.407
Transfer	0.886	0.579

Table 2: Average Precision for Cars

custom YOLOv5 models. The first one is based on YOLOv5l but with Adam instead of the default SGD optimizer. The second custom model has larger depth and layer multiples and uses Adam optimizer. The results of our custom model are presented in Figures 6, 7.

Since the majority of the driving scene images only contain vehicles as moving objects, to better compare the results, we present the AP@.5 and AP@[.5:.95] only for class "cars" for the 3 baseline models and our custom model in Table 2. The AP values for baseline models align with our expectation that as the model gets larger, the performance becomes better. However, it takes YOLOv5m twice the time to train as YOLOv5s does, and YOLOv5l even four times to train, so the training time gets exponentially longer as we increase the depth and layers of the model.

Our two custom models perform slightly worse than the 3 baseline models. Comparing Figure 5 and Figure 6, we notice that the training process with Adam is not as smooth as we use SGD and the performance is better when we use SGD. Although this is out of our expectation, one possible reason is we use default hyperparameters for both models; we know that Adam optimizes best when it is fine-tuned, so the default hyperparameters may not optimizes in its best way. Besides, in our Custom2 model, the performance is not improved as expected. One possible reason is the model has become too complicated and needs more epochs to get similar results. The training result figure shows that the model still has room for improvement, but due to limited computational power, we did not train enough epochs to see its full potential.

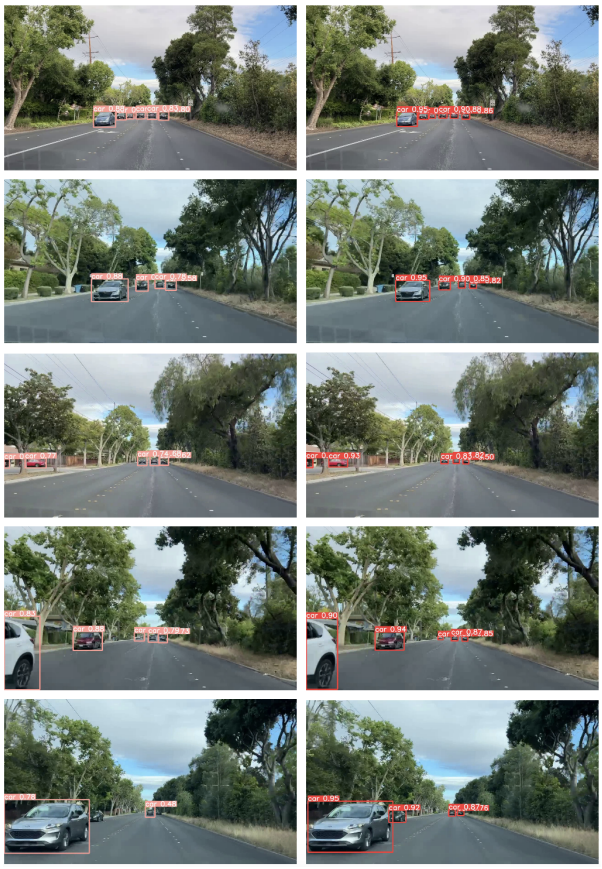

Figure 8: Vehicle Tracking in Video, YOLOv5l(left), Transfer learning(right)

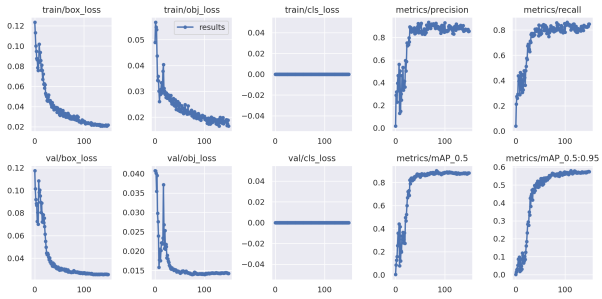

Figure 9: Transfer Learning Result

5.2 Transfer Learning

The results in previous experiments are descent but still have room for improvement. We decided to apply transfer learning on our custom Palo Alto dataset using pretrained YOLOv5s model which is initialized with weights pretrained on the COCO dataset. During transfer learning, we fixed the backbone part which consists of 12 layers. The result after training the model on our custom Palo Alto dataset for 150 epochs is displayed in Figure 9. AP@.5 on the validation dataset is 0.886 and AP@[.5:.95] is 0.579, which are better than any of the models trained from scratch.

5.3 Object Tracking on Video

Finally, we compare how well our two methods work on object tracking using one driving video taken in Palo Alto. The videos lasts around 20 seconds and consists of 603 frames. Here we select the same 5 frames from the object tracking video using the baseline YOLOv5l model and our transfer learning model. The results are presented in Figure 8. The results do not differ too much in detecting the cars, but the transfer learning model is able to detect the cars with a much higher confidence score. Another difference is the cars are loosely bounded by the boxes generated by the baseline YOLOv5l model while cars are tightly bounded by the boxes generated by transfer learning model. This difference is more obvious to identify when we compare two videos instead of two sets of frames because the bounding boxes of baseline YOLOv5l move around the cars and sometimes the model cannot continuously detect some cars. Therefore, the transfer learning method is better for our tracking objective.

6 Conclusion

In this project, we aim to develop an object tracking model for intelligent vehicles in Palo Alto streets. We gained hands-on experience in collecting and annotating dataset, developing and training deep learning models, transfer learning and fine tuning models, and analyzing and presenting our results.

Specifically, we achieved our goal of comparing different YOLO model variants on Udacity Self Driving Dataset, creating and annotating our own Palo Alto driving scene dataset, and performing transfer learning with our custom dataset. Although there isn't a significant difference in performance for different YOLO model variants on the existing Udacity Self Driving Dataset, we have seen an evident improvement of model performance on the Palo Alto Driving Video once we performed transfer learning with our custom dataset.

In the future, we would like to collect more driving videos in Palo Alto to include different and complicated scenarios such as change of lighting in a video, sudden appearance or disappearance of vehicles, or long occlusion of objects. We would also like to detect and track other objects such as pedestrians, bikers, and traffic lights. Finally, we would like to explore SORT and DeepSORT to connect the objects in each frame to form a continuous result instead of stitching the predicted frames together.

7 Contributions

We build our project on top of the existing codebase from Ultralytics. Here is a breakdown of what each person on our team contributed:

- Yan Wang: Developing and training YOLO model variants, annotating custom data, modifying models for transfer learning, analyzing results.
- Fangran Wang: Training YOLO model variants, creating and training custom YOLO model, collecting and annotating majority of the custom data.
- Shanduojiao Jiang: Training YOLO model variants, collecting and annotating custom data, writing majority of project report, creating poster.

References

[1] Give your software the power to see objects in images and video. 1

[2] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016. 2

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. 3

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:886–893, 2005. 2

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 2

[6] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015. 2

[7] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. 2

[8] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, AlexWang1900, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, Feb. 2022. 1, 2, 3

[9] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 1

[10] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534, 2018. 3

[11] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 1, 2, 3, 4

[12] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. 2

[13] Roboflow. Udacity self driving car object detection dataset, Feb 2020. 1

[14] Ultralytics. Ultralytics/yolov5: Yolov5 in pytorch gt; onnx gt; coreml gt; tflite. 3

[15] Paul A. Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR (1)*, pages 511–518. IEEE Computer Society, 2001. 2

[16] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of CNN. *CoRR*, abs/1911.11929, 2019. 3

[17] Wikipedia contributors. Hungarian algorithm — Wikipedia, the free encyclopedia, 2022. [Online; accessed 3-May-2022]. 2

[18] Wikipedia contributors. Kalman filter — Wikipedia, the free encyclopedia, 2022. [Online; accessed 3-May-2022]. 2

[19] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017. 2