Project: Object Detection for Autonomous Driving

Fazli Faruk Okumus
fazli.okumus@student.uni-luebeck.de

Mevluede Tigre
mevluede.tigre@student.uni-luebeck.de

Abstract

Vehicle detection and recognition is an important part of autonomous vehicles for traffic and road administration. In this paper, we present a case study of how the You Only Look Once (YOLO)v5 object detection algorithm can be implemented to detect vehicles. This method uses deep learning technology. First, we take the vehicle images from the KITTI dataset. Then the new training set is generated to retain the detailed features that the object detection network needs to learn, during the network detection process, the image is detected, and the coordinates are located with position mapping. The experimental results show that the trained algorithm can accurately identify vehicles. Moreover, some results are compared with the different YOLO object detection algorithms.

1. Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. Enabling computers to see the world in the same way humans do is still a complex challenge. In the current technology, algorithms for object detection would allow computers to do similar tasks like driving cars in real-time with detection of various objects in digital images or videos and is a key technology behind advanced driver assistance systems (ADAS) by detecting lanes, cars, signs or pedestrian such as. In this paper, we will try detect cars on the high-way and give the technical details of the algorithm that we use and lastly discuss the results.

2. Related Work

The state-of-the-art methods can be categorized into two main types. One-stage methods(region proposal based) consists of models like RCNN [4], SPP-NET [7], Fast RCNN [3], FasterRCNN [13]. Such models reach the highest accuracy rates, but are typically slower. [6] Second framework is two stage-methods(regression-based) that consists of MultiBox, G-CNN, YOLO, SSD [9]. They treat object detection as a simple regression problem by taking an input image and learning the class probabilities and bounding box coordinates. Such models reach lower accuracy rates, but are much faster than two-stage object detectors. [14]

3. Method

YOLO model has several benefits over other methods of object detection :

Speed: This algorithm improves the speed of detection because it can predict objects in real-time.

High accuracy: YOLO is a predictive technique that provides accurate results with minimal background errors.

Learning capabilities: The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

So we decided to move on with YOLOV5 to solve the detection problem, because it can give us high mAP(mean average Precision) values without sacrificing speed.

3.1. YOLO Model

YOLO model unify the separate components of object detection into a single neural network and works using the following three techniques: Residual blocks, Bounding box regression, Intersection Over Union (IOU) as shown in Figure 1

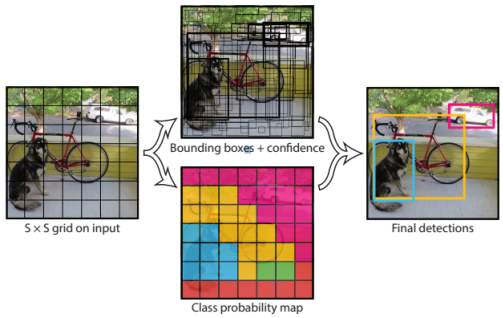

Figure 1: Model divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B * 5 + C) tensor. [11]

- **Residual blocks** The image is split into a square grid of dimensions S×S. Every grid cell will detect objects that appear within them and gives a confidence score for each box. The classification score will be from 0 which represents the lowest confidence level to 1 which reflects the highest confidence level. If no object exists in that cell, the confidence scores will be 0, and if the model is completely certain of its prediction, the score is 1.
- **Bounding box regression** A bounding box is an outline that highlights an object in an image.Each of these bounding boxes is made up of 5 numbers: the x position, the y position, the width, the height, and the confidence as shown in figure 2. [10]

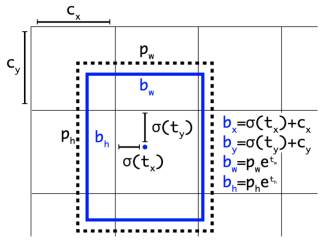

Figure 2: Bounding boxes with dimension priors and location prediction

- **Intersection Over Union** describes how boxes overlap.It is the area of the intersection of the predicted and ground truth boxes divided by the area of the union of the same predicted and ground truth boxes [8]. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects. The final detection will consist of unique bounding boxes that fit the objects perfectly.

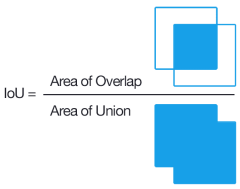

Figure 3: IoU

- **Accuracy** AP (Average precision) is a popular metric in measuring the accuracy of object detectors. Average precision computes the average precision value for recall value over 0 to 1.
  Precision measures how accurate is the predictions.
  Recall measures how good you find all the positives[8]. For example, we can find 80 percent of the possible positive cases in our top K predictions. The mathematical expresssions are given in the equations below.

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP+FP} \quad TP = \text{True positive} \\ & \qquad\qquad\quad TN = \text{True negative} \\ \text{Recall} &= \frac{TP}{TP+FN} \quad FP = \text{False positive} \\ & \qquad\qquad\quad FN = \text{False negative} \end{aligned} \tag{1}$$

The general definition for the Average Precision (AP) is finding the area under the precision-recall curve above

$$AP = \int_0^1 p(r)dr \tag{2}$$

But, to calculate it we are using 101-point interpolated AP as COCO suggest. For COCO, AP is the average over multiple IoU (the minimum IoU to consider a positive match). AP@[.5:.95] corresponds to the average AP for IoU from 0.5 to 0.95 with a step size of 0.05 [8].

3.2. YOLOv5 Architecture

The YOLO model is made up of three key components: the head, neck, and backbone.

- **YOLOv5 Backbone** employs CSPDarknet(whose architecture shown in figure 4) and made up of convolutional layers to detect key features of an image and process them. It is first trained on a classification dataset and trained at a lower resolution than the final detection model, as detection requires finer details than classification. [1]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4: Darknet-53 [12]

- **YOLOv5 Neck** uses PANet to generate the features from the convolution layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates.

- **YOLOv5 Head**

  The head is the final output layer that generate predictions from the anchor boxes for object detection.

4. Evaluation

4.1. Experimental Setup

1. **Settings of Colab** For training on Colab efficiently, we need to enable GPU for our COLAB notebook. Afterwards, we need to connect our drive to our notebook because later on, we will use drive for load data, import .yaml file and create train-validation folders for YOLOV5 training.

2. **Load Data convert to YOLOV5 format** First of all, we need to insert numpy files into our code. And then, for the training, we need to create different folders for labels and images for each training and validation sample seperately. Afterwards, we will fill these folders with our KITTI images and its labels. Also, we split 20 percent of train data as validation to use them to test training procedure at the end of each epoch. Set validation percentage as 20 because according to research this is kind of rule of thumb and we adopted this setting. [2]

3. **Cloning the YOLOv5l Repository** To get the YOLOV5 architecture, we just basically clone repository into our drive folder. To train the architecture, we need to give where is the train,validation,test datas as .yaml file. To do that, we just copy our preconfigured .yaml file into source code instead of using the default one.

4. **Debug Tools** For debugging of YOLOV5 during the training, we employee weight and biases and tensorboard. Weight and biases give us a so useful real-time plots about mAP,loss etc. Also we can see bounding boxes that assign during the training and validation results. Additionally, we use for TensorBoard tool for offline debugging and visualization.

5. **Train**

   (a) **DATA**
   Our dataset is reduced KITTI dataset. Objects other than car inside the images are eliminated. x_{train} data is basically numpy file that includes images as a float array. y_{train} includes how many objects exist for each class inside that image and object's bounding boxes in the format of $[x_{min}, y_{min}, x_{max}, y_{max}]$. By the way, we have only one class and that is car and depicted as 0 inside y_{train}. So, number of class we had for this scenario is 1, because no labels are required for background images in the YOLO use case.

   (b) **YOLOV5**
   In the YOLO family, there is a compound loss which is calculated based on objectness score, class probability score, and bounding box regression score.

   YOLOV5 have used Binary Cross-Entropy with Logits Loss function from PyTorch for loss calculation of class probability and object score. We also have an option to choose the Focal Loss function to calculate the loss. In YOLO v5, the default optimization function for training is SGD. However, you can change it to Adam by using command-line argument. But we choose move on with default choices. Additionaly, we call initial weight for YOLOV5L from command line.

   (c) **Hyperparameter Selection**

   - **Epochs.** Started with 200 epochs. We saw no significant improvement in loss and precision after 100th epoch. Afterwards we decided to reduce number of epoch to prevent overfitting. On the other other hand, when we reduced number of epoch less than 100, we got less accuracy values.
   - **Image size.** COCO trains at native resolution of –img 640, so default value of YOLOV5 does not fit for us because of our image size. So we chose image size as 320.
   - **Batch size**. We should use the largest –batch-size that your hardware allows for, cause small batch sizes produce poor batchnorm statistics and should be avoided. So we leave these hyperparameter as 32.

   Training results are depicted in figure 5

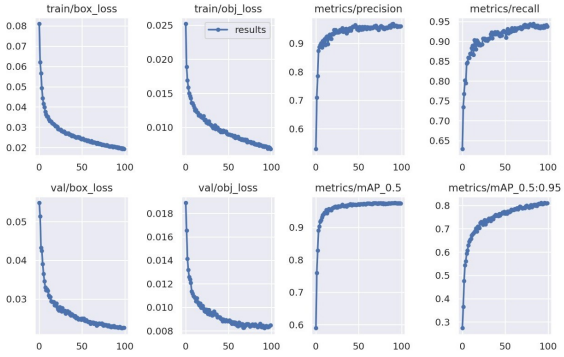

Figure 5: Loss, mAP of training of YOLOV5 by using TensorBoard (epoch=100)

6. **Test** After training, train script will save the model inside the drive. Afterwards you can use it to do detection on test data provided. You can see results in Figure 6

Figure 6: Test result (epoch 100)

4.2. Discussion

In our case, we needed to figure out which structure is be better to be used. To this end, we did training with YOLOV5S and also YOLOV5L. To get higher mAP, we chose YOLOV5L as our main structure. Because it gives us a roughly 10 percent more performance in terms of average precision, as you can see in the Table 1

Structure	mAP 0.5	mAP 0.5-0.95
YOLOV5S-30 EPOCH	97.05	76.35
YOLOV5S-50 EPOCH	97.13	75
YOLOV5S-180 EPOCH	97.5	77.69
YOLOV5L-50 EPOCH	97.58	78.8
YOLOV5L-100 EPOCH	97.58	81.16
YOLOV5L-200 EPOCH	97.78	82.82

Table 1: As you can see YOLOV5L has better mAP result

We chose large structure rather than xlarge , because xlarge yolo structure have approximately 40-50 percent worse performance in terms of speed than large structure as you can see in Figure. 7.

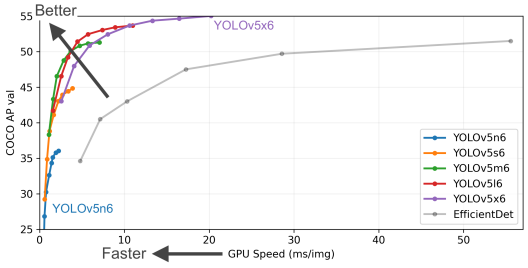

Figure 7: Speed vs AP [5]

Afterwards, we run the training algorithm with different number epoch. After roughly 100 epoch nothing much changed in terms of mAP and precision. To save training time and to avoid overfitting danger, we chose epoch size of 100. Figure 8 shows mAP for different trainings.

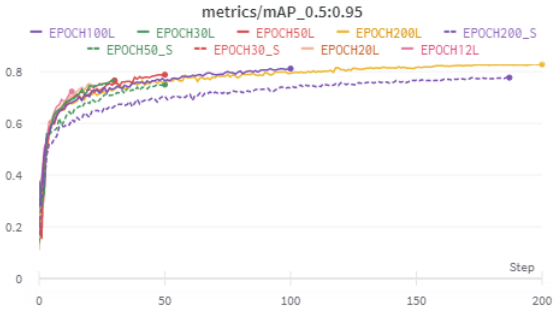

Figure 8: mAP vs number of epoch

After several training with different hyperparameter settings, we tested on the images provided in figure 9 and each of them failed. Our training images contain objects which appear smaller than the object appear in Figure 9 . Also this objects contains different texture than the usual and also, there is a sudden brightness changes throughout the car which is not included in training set generally. Because of these reasons, our trained machines failed on this particular image. You can also see one of the successful result of our trained machine in the Figure 6.

Figure 9: Object that we could not capture

5. Conclusion and Future Work

We did different training with different hyperparameter settings. Our final machine with final settings we chose can get 78.8 mAp on [0.5-.0.95] threshold settings. Test results shows that it perform well on images that generally look like images which is in training set. Additionally, we can get better results on different appearances of objects in images like in Figure 9, and with more sparse training datasets.

References

[1] Ani Aggarwal. Yolo explained. available at https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31, 2021.

[2] The Data Detective. The 80/20 split intuition and an alternative split method. available at https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d, 2020.

[3] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[5] glenn jocher. ultralytics/yolov5. available at https://github.com/ultralytics/yolov5 pretrained-checkpoints, 2022.

[6] Karimi Grace. Introduction to yolo algorithm for object detection. available at https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/: :text=YOLO/, 2021.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[8] Jonathan Hui. map (mean average precision) for object detection. available at https://jonathan-hui.medium.com/map-mean-average precision-for-object-detection-45c121a31173, 2018.

[9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[10] Gaurav Maindola. Introduction to yolov5 object detection with tutorial. available at https://machinelearningknowledge.ai/introduction-to-yolov5-object-detection-with-tutorial/, 2021.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[12] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[14] Petru Soviany and Radu Tudor Ionescu. Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 209–214. IEEE, 2018.