Project: Car detection on KITTI dataset

Muhammad Haris Ibrahim
muhammad.ibrahim@student.uni-luebeck.de

Abstract

Detecting cars on the road is a critical task for autonomous driving. This must be done with high precision, high recall and in real-time. In this paper, we develop a 2D car detector by fine tuning the YOLO-v3 object detector to detect cars on a scaled down version of KITTI dataset. We achieved 51 percent mean Average Precision after fine-tuning YOLO-v3 model previously trained on MS COCO dataset. Improvements to the training process are required to achieve better results.

1. Introduction

Object detection is a critical task for autonomous driving. An autonomous vehicle must be able to detect and keep track of objects of interest in its environment. The classification of objects is not enough, their position and orientation in the environment must also be predicted. Another challenge is that these detections must be performed at high precision and recall but also in real-time on edge devices. The objects of interest can be lane line markings, traffic signs, vehicles, pedestrians and more.

In this project, we focus only on detecting cars. Car detection is specially important in autonomous driving to avoid collisions and plan safe trajectories even at low levels of autonomy.

There are many deep learning based approaches for object detection. We adapt YOLO-v3 [13], a general purpose 2D object detector based on a Convolutional Neural Network (CNN) architecture. For training and evaluating our car detector, we will use a simplified version of KITTI dataset containing cars with 2D bounding box labels [4].

YOLO-v3 is an improved version of YOLO [12] which achieves higher mean Average Precision (mAP) on 2D object detection datasets like MS-COCO [10] while still being real-time. YOLO-v3 is a end to end CNN and performs predictions in only 1 pass.

Transfer learning is then used to finetune it for car detection on our version of KITTI dataset. We obtained 51 percent mean Average Precision (mAP) on a validation set.

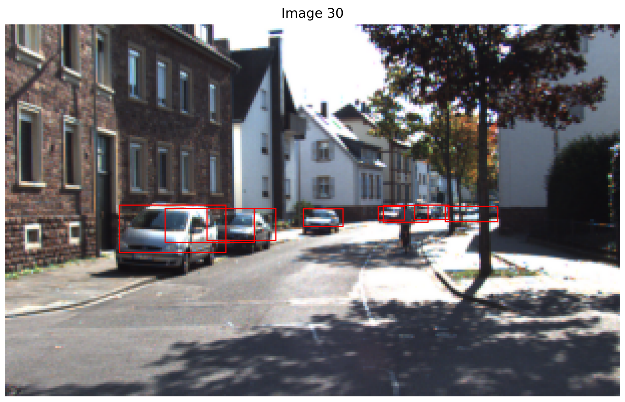


Figure 1. Detections of the model developed in this project on a sample image from the test set.

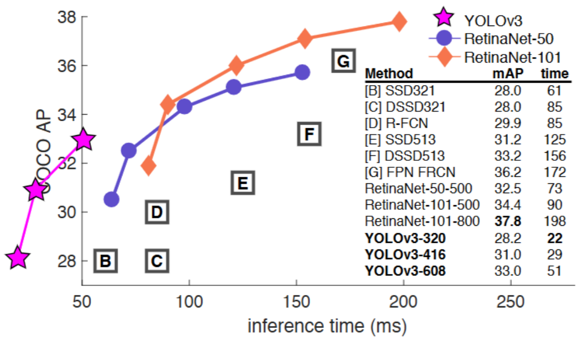
2. Related Work

Since the success of Convolutional Neural Network (CNN) based approaches like Alexnet [6] on image classification tasks, many similar approaches have been designed for object detection. Region Proposal CNNs (RCNN) [14] and its various variants use handcrafted algorithms to generate proposals for regions of interest. These proposals are then passed to a CNN which produces class and confidence predictions. The predictions are then fed back into the region proposal algorithm. This means that multiple passes are required to perform good predictions. This makes these networks slow at inference time. These multi-stage detectors are, therefore, not realtime.

YOLO architecture and Single Shot Detector [11] solved this problem by performing the object detection in only one evaluation of the network. These are end to end CNNs. Out of these two, YOLO is the fastest.

YOLO architecture frames the object detection problem as a regression problem. It divides the image into a SxS grid, where S is the grid size. For each of these grid cells, there are three bounding box predictors in the output layer. A bounding box prediction consists of 4 coordinates of the bounding box. Each grid cell also includes a class prediction.

YOLO-v3 also divides the images into SxS cell, but it



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 2. speed (ms) vs accuracy (AP) on MS. COCO test set. YOLOv3 outperforms many multistage and single stage object detectors in AP while being faster than all of them. This image is adapted from [13] which is itself a version of image from [9]

produces class probabilities for each bounding box not just for each grid cell. Predictions are also made at three different scales. Prediction of the bounding box coordinates are also as offsets of bounding box priors (anchor boxes). These anchor boxes are based on shapes of bounding boxes common for the problem. The network outputs a large number of predictions. Bounding box and class predictions are made at 3 scales, SxS grid cells per scale, and 3 per grid cells. Non Maximum Suppression (NMS) is applied to keep only the best predictions. YOLO-v3 has already been applied to many object detection problems like face detection [7], predestrian detection [16] and much more.

A brief comparison of the performance of many object detection frameworks in shown in Figure 2.

There are more recent and improved versions of YOLO such as [15] that are not developed by the original authors. Recent works like Detection Transformer [2] use a CNN backbone with a transformer encoder and decode blocks. These approaches achieve higher mAP precision. They also remove the need for anchor boxes and NMS. Because of the limits of the dataset, limited time and GPU compute, we use the YOLOv3 architecture and not the newer ones.

3. Method

In this section, we provide a summary of the YOLO-v3 architecture. You should skip to the evaluation section to learn about the specific adaptions done by us.

3.1. Bounding Boxes

YOLO-v3 like many other object detectors does not predict the absolute coordinates of the bounding box. The bounding box centre coordinates are predicted as offsets to position of the top left corner of the grid cell. The width and height of the box are predicted as offset of the width and height of anchor boxes. These anchor boxes are predefined based on the likely shapes of anchor boxes for the problem at hand. The bounding box offsets are t_x, t_y, t_w, t_h. If the position of the top left of grid cell relative to the top left of the image is given by (c_x, c_y) and the bounding box has width and height p_w, p_h, then the bounding box coordinates are calculated by:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$

The ground truth values will be the left side of the above eqautions. We rearrange the equations to find the ground truth offsets.

The authors used clustering analysis to find the anchor box dimensions which have the most overlap with ground truth boxes on MS COCO dataset. We did not do such analysis on our dataset and used the anchor box dimension from the orignal source code.

There are 09 anchors boxes—03 at each scale. Each grid cells within a scale, predicts three bounding boxes as offsets of these anchors boxes. The anchor boxes are in sets of three where the size of boxes in each set is according to the prediction scale.

3.2. Objectness Score

Along with 04 coordinates of the bounding box, the network also predicts a objectness score for the bounding box. At test time, this is the confidence of the model in predicting that an object exists in the box and how well the box fits the object. At training time, this score should be the interesection over union between the predicted box and the ground truth box. The loss function forces this behaviour.

3.3. Class Predictions

The network predicts a class probability for each bounding box. At training time, we use Cross entropy loss for classification prediction of the box. This is probability that there exists an object of class i given that there is an object in the box, $\Pr(\text{Class}_i|\text{Object})$. We can multiply this by objectness score to get the classification confidence.

3.4. Features extraction backbone

The backbone of the network is a 53 layer CNN named as Darknet 53. The overall summary of the layers is shown in Table 1.

The authors did not explain any further details about the backbone. After spending sometime with the source code, our understanding is that each CNN block in the Table consists of a CNN layer, followed by leaky ReLU activation

	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			

Table 1. **Darknet-53.**

layer, and lastly a Batch Normalization layer. The filter sizes are mentioned in the Table. The 1x, 2x and such labels in the Table show the number of repeats. So, 2x means that such a block is repeated 2 times. It should be noted that the architecture includes skip connections like Resnets [5]. The authors trained this backbone for image classification on ImageNet dataset [3]. Further details about the training process were not provided. We assume a strategy similar to the YOLO-v1 was used.

3.5. Three Scale Predictions

The extracted features of the backbone are then passed to the detection heads at three different scales. This is inspired by Feature Pyramid Networks [8]. The size of the outputs are decided by dividing the image into SxS grid. There are three different grid sizes for three different scales. The idea is to use a larger grid size to help detect large object, then a medium size grid and then finally a smaller grid size. At each scale, the network has SxS predictors. Each grid cell has three predictors. Each predictor then predicts 4 bounding box coordinates (as offsets to three anchors), 1 objectness score and C class probabilities for C classes. So the total predictions at each scale are $S \times S \times [3 * (4 + 1 + C)]$. After the output of the feature extract backbone, several convolutional layers are added. The overall structure can be seen in Figure 3

The detection head for the first scale then adds two more Convolutional blocks. For the second scale prediction, we take the output of the network just where the first detection head is attached and then upscale it by 2X. We then concatenate this feature map with a feature map a couple of blocks before. The idea is to use earlier feature maps along with upscaled current feature maps to make predictions on a higher resolution. This helps with detecting medium and small objects. The second detection head is then attached here which then adds two more CNN blocks to make predictions. The process is repeated to get predictions for the third scale.

The Darknet 53 backbone trained on ImageNet is the modified with these new layers. The combined model is then trained on MS COCO object detection dataset. The results are shown in Figure 2. The loss function is explained in the YOLO-v3 and YOLO-v1 papers. We used the same loss function.

4. Evaluation

In this section, we describe how we adapted the YOLO-v3 for our purposes.

4.1. Dataset

We used a scaled down version of KITTI dataset. It contains only 1 class for cars. There are 3712 training images with single labelled car to multiple cars in an urban environment. The test set consists of 3769 images. The train set contains many images without any labelled cars. We remove them and end up with only 3177 images. We then divide into train and validation sets with 80/20 split (2541, 636 images).

4.2. Data Augmentation

We used data augmentation strategies like random crop. color jitter (randomly change brightness, contrast, hue, staturation), affine transformation (change zoom level, translate left and right, rotate, and shear), horizontal flip, random blurring, colour channel shuffling, and grey scale. All these were applied with various probabilities. Finally the images are standardized (mean 0, std 1). These transforms are also applied to the bounding box labels. These data augmentation strategies reduce overfitting and make the model robust to colour variation due to weather and lightning, camera angles, artifacts and more.

4.3. Training Strategy

We download the pretrained weights of YOLO-v3 for MS COCO. We then finetune the model using transfer learning. We first change the last layer in each scale to adapt the output for only 1 class instead of 80 in MS COCO.

We then freeze all the layers except the last layer in each scale. This means that all layers contribute to forward pass but the gradients in backward pass are only calculated for

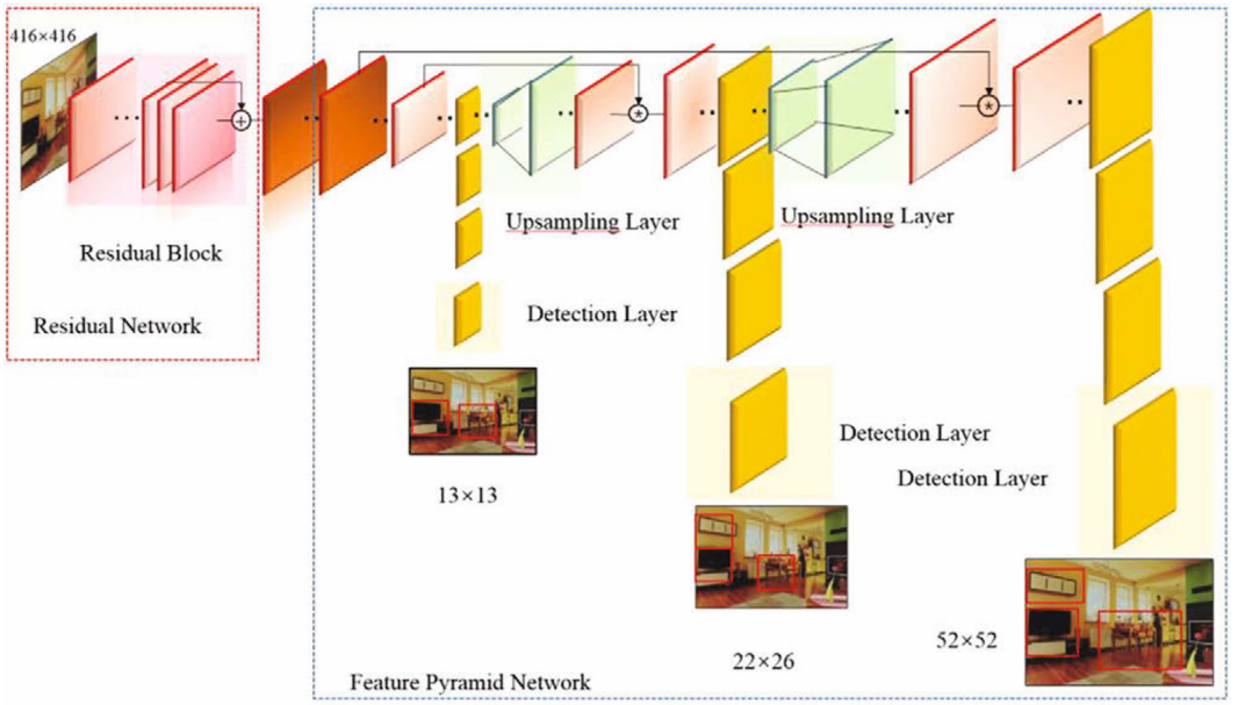


Figure 3. The overall backbone and the three scale detection heads of YOLO-v3. This is taken from [7]

the last layer in each scale. We used Adams optimizer with a weight decay of 1e-04. The YOLO loss function is a weighted sum of classification loss, no object loss, objectness score loss and bounding box coordinate loss. Their weights are initially set as (1, 10, 1, 10), respectively.

We trained on our train set for 20 epochs on a learning rate of 3e-05. The loss went down but the rate of decrease stagnated at the end.

We then unfreeze all the layers in each detection head. We then trained for 30 more epochs on a learning rate of 3e-05 and faced the same problem.

We also observed that at this stage the model has perfect classification accuracy because there is only 1 class. The network has a bad no object score (predicting bounding boxes with high objectness score in grid cells with no objects) and it still performed badly in localizing the bounding boxes (quantified by IOU between ground truth and pred box). We therefore changed the weights of the loss to (1, 15, 10, 20). Namely, we increased the no object penalty by 1.5 times, we increase the objectness score penalty by 10 times, and we double the weight of bounding box coordinates loss. We then train for 10 more epochs.

Finally, we unfreeze two more CNN blocks before each detection head. This means that the CNN blocks before each head now also receive gradient and are updated. We decrease the learning rate to 1e-05 and train for further 60 more epochs.

The above training procedure is not planned but reactive in nature as we trained and observed the model performance and tweaked parameters.

4.4. Discussion and Results

We acheived a mAP of 51 percent on both the training set and validation set. We never used the validation set for any automated hyperparameter optimization. Therefore our test accuracy is likely to be around 51 percent mAP as well.

Beyond the accuracy metric, we observed that our model has some trouble properly localizing some cars. We think this is due to many reasons.

First, we used the anchor boxes suitable for MS COCO. Our dataset has only rectangular objects. Using anchor boxes well suited for cars should improve mAP.

Second, our training procedure should be optimized. Hyperparameter optimization in terms of learning rate schedules, how many layers to unfreeze, and loss weights can be done with more time and GPU compute at hand (which we did not have). Some sample predictions of the model on test set can be seen in Figure 1 and Figure 4. We observed that sometimes the model confused motorists, trains and trucks with cars. We think that this can be improved by including labels for those classes as well. Finally, we think that training on a larger, more diverse dataset with higher resolution

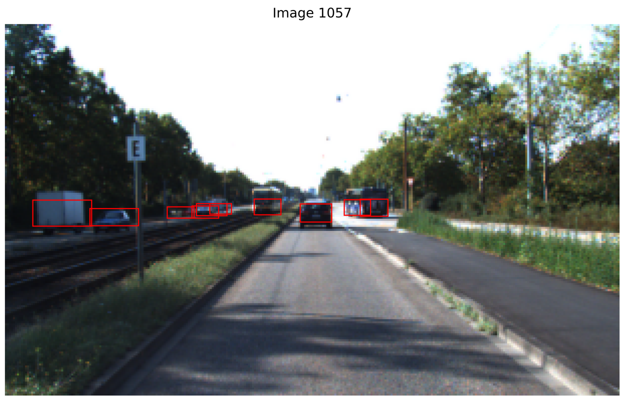


Figure 4. Detections of the model developed in this project on a sample image from the test set.

Set	mAP
Train	0.51
Validation	0.51

Table 2. Results. The mAP is calculated using PASCAL VOC 2007 metric using instructor provided code.

images should improve car detection for practical applications.

5. Conclusion and Future Work

We adapted YOLO-v3 2D object detector to perform car detection on a scaled down version of KITTI dataset. We acheived a mAP of 51 percent. Hyperparameter optimization is required in terms of finetuning the pretrained model on our dataset. Clustering analysis can be done to find better anchor boxes. These measures should improve performance. In future, newer object detection frameworks based on Detection Transformer should be investigated. We learned about 2D object detection by doing this project and coded many things from scratch.

6. Acknowledgments

We adapted a Pytorch implementation of YOLO-v3 by [1].

References

[1] Sana Persson Aladin Persson. YOLOv3-PyTorch. `https://github.com/SannaPersson/YOLOv3-PyTorch`, 2022. Accessed: 2025-06-12.

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[3] 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.

[4] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[7] Lin, Li Zheng-Chun, Jiang Dian, Wang Yun-Zhi, Chao Jing, and Zhang. Yolov3: Face detection in complex environments. In *International Journal of Computational Intelligence System*, pages 1153–1160. Press, 2020.

[8] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[9] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*, pages 740–755. Springer, 2014.

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[12] 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.

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

[14] 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.

[15] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.

[16] Yulin Yang, Baolong Guo, Cheng Li, and Yunpeng Zhi. Microsoft coco: Common objects in context. In *An Improved YOLOv3 Algorithm for Pedestrian Detection on UAV Imagery*, pages 253–261. Springer Singapore, 2020.