Project: Object Detection for Autonomous Driving

Rakesh Reddy Kondeti
rakesh.kondeti@student.uni-luebeck.de

Neelam Sattur
n.sattur@student.uni-luebeck.de

Abstract

The fundamental prerequisites in the perception block of autonomous vehicles is object detection. In this project, we apply YOLOv3 on the KITTI-2D dataset to produce an object detection network that makes bounding box predictions for an image without the need for expensive preprocessing or deep evaluations.

1. Introduction

Although self-driving cars are gaining more and more prevalence in the real-world, there is still a need to provide autonomous vehicles with reliable perception systems to facilitate and improve the vehicle's ability to understand the surrounding environment. Object detection is one of the key elements of autonomous driving, which can identify an object from the known set of objects, and provide information about its position within the image.[4]
This makes it necessary to develop an algorithm which is reasonably accurate while operating on strictly the input image. The report begins with a brief introduction to the objection detection algorithms and followed by the overview of related previous works. In particular, YOLOv3 is discussed in detail and the implementation of this network using the Github repository[5] on a reduced KITTI 2D dataset. Finally, a summary and future scope is discussed.

2. Related Work

2.1. OverFeat

OverFeat is a Convolutional Network-based image classifier and feature extractor used to recognize images and extract features. The main idea is that it jointly performs object recognition, detection, and localization. OverFeat is eight layers deep, and depends heavily on an overlapping scheme that produces detection boxes at multiple scales and iteratively aggregates them together into high-confidence predictions.[4]

2.2. VGG16

VGG16 is a convolutional neural network model which was an attempt to usurp OverFeat's dominance in object classification and detection by exploring the effects of extreme layer depth. It was one of the famous models submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.[2]

2.3. Fast RCNN

In Fast RCNN, the input image is fed to the CNN, which in turn generates the convolutional feature maps. Using these maps, the regions of proposals are extracted. Then, a Region of Interest (RoI) pooling layer is used to reshape all the proposed regions into a fixed size, so that it can be fed into a fully connected network. It uses a single model which extracts features from the regions, divides them into different classes, and returns the boundary boxes for the identified classes simultaneously.
The main drawback pf using Fast RCNN is that it also uses selective search as a proposal method to find the RoI, which is a slow and time consuming process.[7]

2.4. YOLO

YOLO refers to "You Only Look Once" is one of the most versatile and famous object detection models.YOLO is a recent model that operates directly on images while treating object detection as regression instead of classification. YOLO has the poorest performance out of all the above models, but more than makes up for it in speed. YOLO algorithms divide all the given input images into the $S \times S$ grid system. Each grid is responsible for object detection.
These grid cells predict the boundary boxes for the detected object. For every box, we have five main attributes: x and y for coordinates, w and h for width and height of the object, and a confidence score for the probability that the box containing the object.[3]

YOLOv3 is capable of making predictions at a rate of approximately 11 frames per second, and achieves a mAP

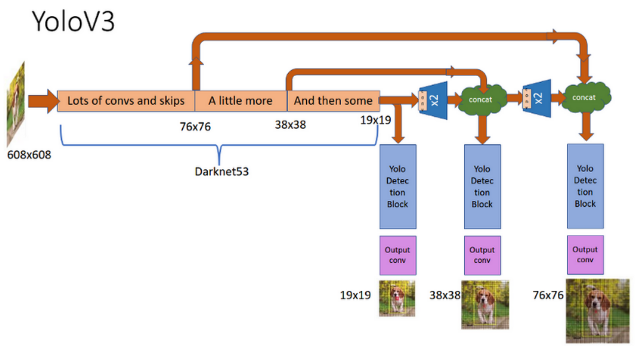


Figure 1. The network architecture of YOLOv3

accuracy of around 12.83%.

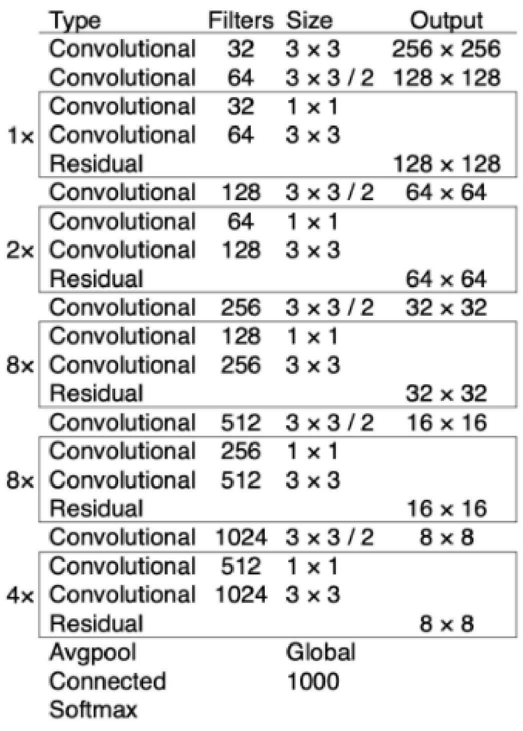
3. Method

3.1. Architecture

The most popular and current version of the YOLO approach is the third iteration called YOLOv3. It has refined the design of the previous version by using multi-scale prediction and bounding box prediction through the use of logistic regression.
YOLOv3 uses a variant of Darknet, a framework to train neural networks, which originally has 53 layers. Inspired by ResNet and FPN (Feature-Pyramid Network) architectures, YOLO-V3 feature extractor, called Darknet-53 (it has 52 convolutions) contains skip connections (like ResNet) and 3 prediction heads (like FPN).
For the detection task another 53 layers are stacked onto it, accumulating to a total of a 106-layer fully convolutional architecture, as shown in Figure 1.
In the convolutional layers, kernels of shape 1x1 are applied on feature maps of three different sizes at three different places in the network. The algorithm makes predictions at three scales, given by downsampling the dimensions of the image by a stride of 32, 16, 8 respectively.
Downsampling, the reduction in spatial resolution while keeping the same image representation, is done to reduce the size of the data. Every scale uses three anchor bounding boxes per layer. The three largest boxes for the first scale, three medium ones for the second scale and the three smallest for the last scale. This way each layer excels in detecting large, medium or small objects.[1]

3.2. Darknet framework

Darknet is an open source neural network framework written in C and CUDA. 2It is fast, easy to install, and supports CPU and GPU computation. It features YOLO and displays the information as it loads the "config" file and the weights. It then classifies the image and prints the top-10 classes for the image file.[6]



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

Figure 2. Darknet - 53 Backbone

3.3. Yolo Loss

YOLO predicts multiple bounding boxes per grid cell. To compute the loss for the true positive, we only want one of them to be responsible for the object. For this purpose, we select the one with the highest IoU (intersection over union) with the ground truth. This strategy leads to specialization among the bounding box predictions. Each prediction gets better at predicting certain sizes and aspect ratios. YOLO uses sum-squared error between the predictions and the ground truth to calculate loss. The loss function composes of:

1. Classification loss,

$$\sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \bar{p}_i(c))^2 \tag{1}$$

where, $1_i^{obj} = 1$ if an object in cell **i**, otherwise 0. $\bar{p}_i(c)$ denotes the conditional class probability for class **c** in **i**.

2. Localization loss(errors between the predicted bound-

ary box and the ground truth),

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj} [(x_i - \bar{x}_i)^2 + (y_i - \bar{y}_i)^2] +$$

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj} [(\sqrt{w_i} - \sqrt{\bar{w}_i})^2 + (\sqrt{h_i} - \sqrt{\bar{h}_i})^2]$$

where, $1_i^{obj} = 1$ if an object in cell **i**, otherwise 0.
λ_{coord} increases the weight for the loss in the boundary box coordinates.

3. Confidence loss(the objectness of the box),

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj} (C_i - \bar{C}_i)^2 \quad (2)$$

where, $1_i^{obj} = 1$ if an object in cell **i**, otherwise 0.
$\bar{C}_i$ is the box confidence score of box **j** in cell **i**

We do not want to weight absolute errors in large boxes and small boxes equally. i.e. a 2-pixel error in a large box is the same for a small box. To partially address this, YOLO predicts the square root of the bounding box width and height instead of the width and height. The total loss is calculated by adding up classification loss, localization loss and confidence loss

4. Evaluation

4.1. Experimental Setup

Training a deep neural network that involves compute-intensive tasks on extensive datasets can take weeks or even months using a single processor. Although many cloud computing resources are available, we decided to use Google Colab's free GPU version, which greatly reduced the training time. However, due to limitations in the free version, we could train only on smaller datasets till 1500 images.
Implementing a Neural network or a deep learning framework from scratch using a programming language is tedious and hence we did not consider building a neural network from scratch. Due to the growing popularity of the object detection algorithms and deep learning, there are already many different available. The fact that most of the available deep learning frameworks have APIs in python indicates that Python is one of the more adopted languages when working with machine learning. Therefore, the implementation of YOLOV3 done in Python, using Pytorch and Keras, with TensorFlow as a backend for handling low-level operations.

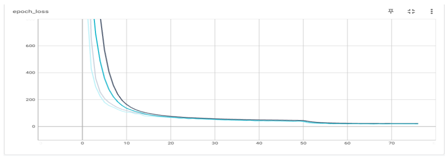


Figure 3. Epoch Loss for 500 images (black line: training loss, blue line: validation loss)

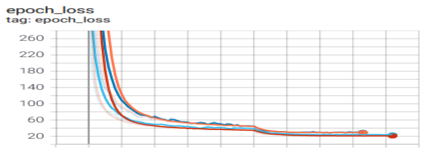


Figure 4. Epoch Loss for 1000 images

For this project, we used the git repository of Anton Muelehmann, University of California, Berkley. We made necessary changes to this Github repository to make it suit our requirements.[5]

4.2. Dataset

KITTI is a driving scene dataset used for object detection, tracking, semantic segmentation, and optimal flow purposes. It was collected around the city of Karlsruhe, Germany, by the Karlsruhe Institute of Technology(KIT) and Toyota Technological Institute of Chicago (TTI-C). We used already provided reduced KITTI-2D dataset, which consists of 3712 training images and 3769 testing images and each image is of resolution 185 x 306. However, only the car images are annotated and labeled. For the purpose of training, the training dataset is split into 90% training set and 10% validation set.

4.3. Preprocessing

The power of deep learning networks comes from their ability to learn patterns from large amounts of data. Thus, understanding the data used to train the YOLOv3 network implemented in this project is critical to achieve as good performance as possible. When we use a neural network like YOLO to predict multiple objects in a picture, the network is actually making thousands of predictions and only showing the ones that it decided were an object.
Resizing the images to match the the network is considered as one of the first steps that is done usually. But in our YOLOv3 network, this already taken care of. The image is resized 416 x 416, by keeping preserving all the information(by padding). And the given boundary box labels are slightly modified in terms of their position to match the format of network.

4.4. Training

During training of the network, we performed several runs on a smaller training dataset of around 50-100 images

TDS	E	TL	VL	FPS
500	83	28.39895	26.84009	19.2
1000	92	20.83002	22.42418	20.0

Table 1. Observed Results: Training Data Size(TDS), Epoch(E), Training Loss(TL), Validation Loss(VL) and Frames per second (FPS).

to overfit the model. Taking computation complexity into consideration, we did not train the network with complete dataset. However, we are able to run on a data size of 500 images and of 1000 images and the results are attached to the report.
Keeping the ratio of training to validation data as 0.1, 102 epochs were performed, out of which 51 epochs are made to run with a batch size of 32 and the remaining 51 epochs are made to run on a batch size of 4. When trained with 500 images, by excluding all the negative samples, data is trained on 396 samples and validated on 44 samples,and for 1000 images on 743 samples, validated on 82 samples. The plots of epoch losses seen in Figures 3 and 4.

4.5. Testing

For each of the trained network on 500 images and 1000 images, prediction was done on both training and testing samples. As the network has already trained on the training images, it predicts the training images better the testing images and can be seen in the Figure.5 , 6

During testing phase, the network runs the predictions on the entire testing dataset to detect the objects in the image by drawing the boundary boxes around them.

4.6. Discussion

The trained network is able to identify objects (cars) from the images in the most of the case and results obtained are quite satisfactory. When the model was trained on smaller amount of data, the model overfits thereby predicting the trained images better than the new images. Due to limited access to Google Colab GPU, we could train only on maximum of 2000 images which produces a mAP score of 65%.
However, in some cases, the model is not able to predict all the cars present in the image 7. This could probably be a poor training of the model due to small dataset, batch size or learning rate. The prediction results obtained for the trial is as shown in Figures 5 6.
Figure5 and6 show a positive prediction.
The table below shows the best epoch, training loss and validation loss for each run 1.

5. Conclusion and Future Work

In this project, we have approached the object detection problem by creating a simple network using YOLOv3 on a reduced KITTI 2D dataset. This implementation does not have expensive preprocessing and it's very fast at predicting the objects in the images.
Possible future work and improvements include trying to extend the training loss function to be more accurate, the model accuracy can be improved by performing data augmentation, setting parameters like batch size, number of epochs, learning rate, etc,. to optimal values and data that is to be trained and tested.

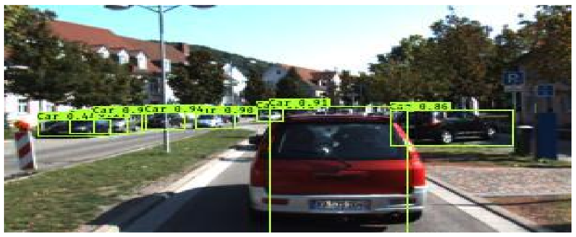


Figure 5. Network Prediction(trained on 500 images)

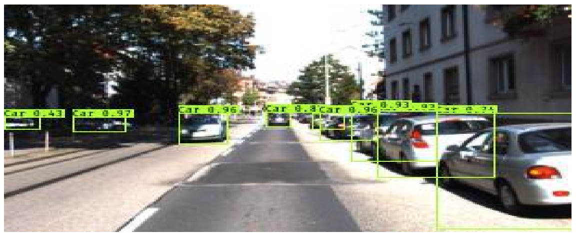


Figure 6. Positive Training Prediction on 1000 images

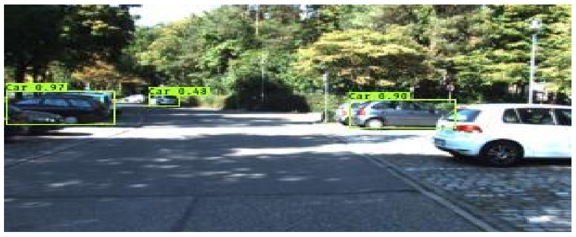


Figure 7. All cars are not detected.

References

[1] Karlijn Alderliesten. Yolov3 — real-time object detection. 2020.
[2] Muneeb ul Hassan. Vgg16 – convolutional network for classification and detection. 2018.
[3] Joos Korstanje. Yolo v5 object detection tutorial, 2020.
[4] Gene Lewis. Object detection for autonomous vehicles, 2014.
[5] Anton Muehlemann. Trainyourownyolo: Building a custom object detector from scratch, 2019.
[6] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.
[7] Pulkit Sharma. A step-by-step introduction to the basic object detection algorithms (part 1). 2018.