Final Year Project Report

Real-time Object Detection for Autonomous Driving using Deep Learning

Majid Manzoor,

Yasir Hanif

Dated: 20.10.2022

Examiner: Prof. Dr. Inam-Ullah

Department of Computer Science

Mohi-Ud-Din Islamic University Nerian Sharif, Pakistan

Contents

1 Introduction

1.1 Object Detection for Autonomous Driving

Autonomous driving is one of the most anticipated technologies of the 21st century and one of the most active research topics at the moment. Autonomous driving attempts to navigate roadways without human intervention by sensing and reacting to the vehicle's immediate environment. Autonomous driving systems rely critically on object detection, which involves identifying, classifying, and precisely localizing road users, such as vehicles, pedestrians, cyclists, and traffic signs, in real time. The two-stage detector R-CNN (introduced in 2013) combined selective search region proposals with CNN-based feature extraction and SVM classifiers, achieving approximately 53.3% mAP on PASCAL VOC 2012. Still, its inference speed was extremely slow, often several seconds per image, due to per-region CNN processing [1]. Fast R-CNN (2015) addressed these inefficiencies by computing a single convolutional feature map per image, applying RoI pooling for region proposals, and consolidating classification and bounding-box regression in one network. This design reduced training time by roughly 9× and testing time by about 213× compared to R-CNN, and achieved around 66.9% mAP on VOC 2007 and 65.7–68.4% on VOC 2012, depending on training data [2]. Faster R-CNN further integrated region proposal generation via an end-to-end Region Proposal Network (RPN), sharing convolutional features between proposal and detection branches. With a VGG-16 backbone, it delivered 69.9–78.8% mAP on VOC 2007 and 67.0–75.9% on VOC 2012, reducing inference to around 0.2 s per image (~5 FPS) [3].

In contrast, the YOLO (You Only Look Once) family, first introduced in 2015, reframes detection as a single-stage regression task: a unified CNN processes the full image in one pass, divides it into a grid, and predicts bounding boxes and class probabilities simultaneously, enabling inference at tens to hundreds of frames per second at some cost to localization precision—especially for small or overlapping objects [4]. YOLOv4 (2020) introduced architectural improvements such as Cross-Stage Partial (CSP) connections, robust loss functions, and advanced augmentation techniques, yielding ~65.7% AP on COCO at ~65 FPS on a Tesla V100 GPU [5]. Community benchmarks report YOLOv7-E6 achieving ~55.9% AP at ~56 FPS, significantly exceeding many transformer-based detectors in speed while maintaining competitive accuracy [6].

1.2 The Berkeley DeepDrive (BDD100K) Dataset

Autonomous driving research has been significantly advanced by comprehensive datasets; however, many traditional driving datasets are constrained by limited visual diversity, restricted scene variety, and insufficient annotation richness [7]. To address these limitations, Yu et al. (2018) introduced the Berkeley DeepDrive dataset (BDD100K), currently the largest and most diverse open driving video dataset, comprising 100,000 video clips (~40 s each at 30 fps) captured across diverse geographic regions in the United States, and annotated for ten heterogeneous tasks to support robust multitask learning [8]. Each clip includes GPS/IMU data, and keyframes sampled at the 10-second mark are richly annotated, covering image-level tagging, 2D bounding boxes for ten road-object categories, pedestrian, rider, car, truck, bus, train, motorcycle, bicycle, traffic light, and traffic sign across all 100,000 frames [9].

The dataset spans a wide range of weather conditions (clear, overcast, rainy, foggy) and lighting scenarios (daytime, nighttime, dawn/dusk), facilitating the development of models resilient to environmental variation [8]. In total, BDD100K includes approximately 1.8 million road-object instances across 100,000 images, supporting downstream tasks such as semantic segmentation (19 classes), instance segmentation, drivable area segmentation, lane marking detection, object tracking, and domain adaptation [10]. The diversity in scene types and annotations makes BDD100K particularly suited for evaluating autonomous driving models under real-world variability and multitask settings.

1.3 Approach

The goal of our project is to detect and classify traffic objects in video streams in real time by training and evaluating two state-of-the-art detection models, YOLO and Faster R-CNN, on the Berkeley DeepDrive dataset (BDD100K). We aim to compare their performance in autonomous driving contexts by measuring both frames per second (FPS) and mean average precision (mAP), seeking to approach the current state-of-the-art detection performance on BDD100K, which achieves 45.7 $mAP_{0.5}$ using a hybrid incremental network [11]. Our evaluation emphasizes live video processing capability and detection accuracy under realistic driving scenarios.

2 Related Work

2.1 Object Detection Models

Recent object detectors are generally categorized into **two-stage detectors** and **one-stage detectors** [12].

Two-stage detectors operate in two sequential phases: first, a **region proposal** stage that identifies potential object locations (e.g., via a Region Proposal Network in Faster R-CNN), followed by a **classification and localization** stage that refines these proposals. This architecture enables higher localization accuracy and superior performance on small or densely packed objects, but results in lower inference speed due to the computational overhead associated with processing numerous proposals. In contrast, one-stage detectors, such as YOLO and SSD, eliminate the region proposal mechanism and directly regress bounding box coordinates and class probabilities in a single network pass over the image. This streamlined pipeline dramatically improves inference speed, often achieving real-time performance, but typically exhibits somewhat lower accuracy compared to two-stage detectors, particularly in challenging scenarios involving small objects or complex scenes [12]. In essence, two-stage models provide higher precision at the cost of speed, whereas one-stage models deliver faster inference suitable for real-time use, though with potential reductions in detection accuracy in difficult cases.

Two-stage detectors

R-CNN was the first region-based CNN for object detection using selective search for region proposals and a CNN for feature extraction on each region proposal. Because of its complex multistage training and slow test time, a faster version named fast R-CNN was proposed. Fast R-CNN extracts the features of an image once and then produces the region proposals, which are then classified. Faster R-CNN, the most representative region-based detector, improved the region-based CNN further by using a region proposal network (RPN) instead of an external region proposal method for producing region proposals, which lead to a high speed-up in test time [13]. Mask R-CNN is an extending work to Faster R-CNN adding a fully connected mask head to the network. It is mainly used for instance segmentation tasks.

One-stage detectors

YOLO and its successive versions (v2, v3, and v4) are **one-stage object detectors**, renowned for real-time detection thanks to their exceptional inference speed. The original YOLO model divides an input image into an S×S grid, where each grid cell predicts multiple bounding boxes and associated class confidence scores in a single forward pass, achieving frame rates of approximately 45 FPS, albeit with moderate localization accuracy [14]. YOLOv2 (2016), also known as YOLO9000, introduced anchor boxes, batch normalization, and higher-

resolution classifiers, attaining mAP around 78.6% on PASCAL VOC at ~40-67 FPS, significantly improving both accuracy and generalization [15]. YOLOv3 (2018) made further enhancements by adopting the deeper Darknet-53 backbone, implementing multi-scale predictions (feature pyramid style), and logistic class prediction, resulting in ~57.9 mAP at approximately 28 FPS on a Titan X GPU [16]. YOLOv4 (2020) introduced architectural innovations such as CSPDarknet-53 backbone, Mish activation, Mosaic augmentation, SPP, PANet neck, DropBlock regularization, and CIoU loss to reach ~65.7% AP50 on the COCO dataset at ~65 FPS on a Tesla V100 GPU, outperforming prior versions in both speed and accuracy [17].

By contrast, the Single Shot MultiBox Detector (SSD) is another one-stage detector that uses multiple default bounding boxes of varying scales and aspect ratios at each feature map location. SSD predicts class confidences and box offsets simultaneously across several feature maps with different resolutions, which enables efficient handling of objects at multiple scales; for example, SSD300 achieves ~72.1% mAP at ~58 FPS on VOC2007 using 300×300 input resolution, and SSD500 reaches ~75.1 mAP at similar speeds, making it competitive with two-stage detectors while maintaining real-time performance [18].

2.2 Datasets for Object Detection

Several labeled datasets with bounding boxes were created to tackle specific detection problems. They are used as benchmarks to compare different architectures and algorithms and set goals for solutions. Popular datasets for general object detection are PASCAL VOC, MS COCO (Microsoft Common Object in Context), and ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Datasets for object detection in traffic and driving scenes tailored for autonomous driving are, for example:

- **KITTI**: This dataset covers traffic scenes of the city of Karlsruhe. The scenes are divided into 5 categories, including annotations (8 classes) and bounding boxes [19].
- **Waymo Open:** This dataset contains 1950 driving segments and considers 4 object classes. Each segment consists of 20s driving. The data also considers various environmental and urban traffic scenes [20].

- **nuScenes:** This dataset considers complex urban driving scenes with 3D object annotations for autonomous driving. It contains 1000 scenes of 20s each, 23 object classes, around 1,4 million camera images of the cities Boston and Singapore [21].

3 System Description

3.1 YOLO (You Only Look Once)

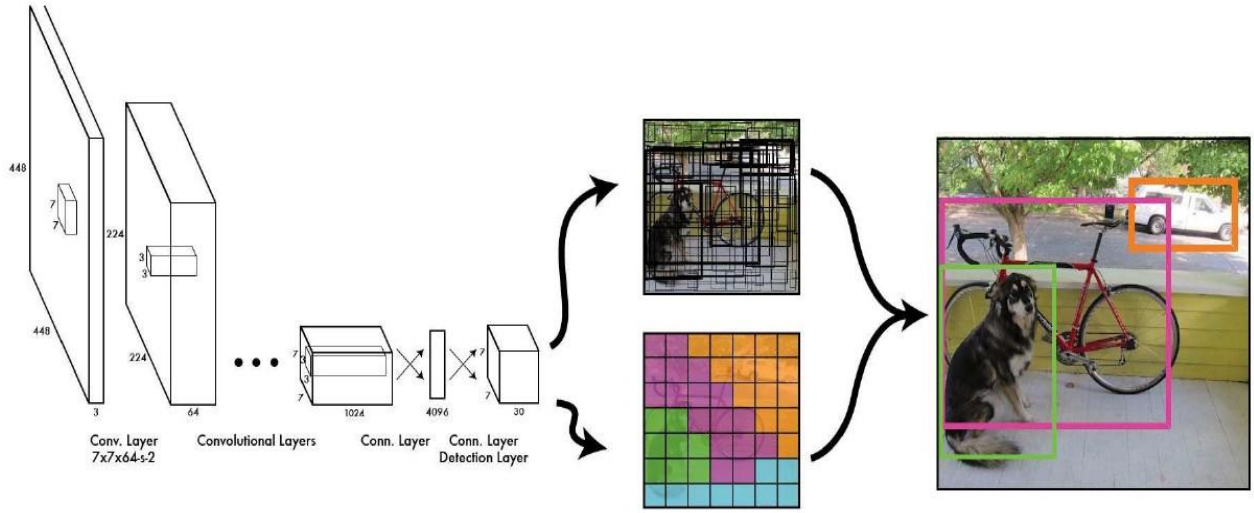

Figure 3.1: Pipeline of YOLO's algorithm [23]

You Only Look Once (YOLO) is a modern object detection algorithm developed and published in 2015 by Redmon et al. [14]. The name of the algorithm is motivated by the fact that the algorithm only looks once at the image and requires only one forward propagation pass through the neural network to make predictions, unlike other state-of-the-art object detection algorithms, which work with region proposals and look at the image multiple times. YOLO uses a single end-to-end convolutional neural network that processes RGB images of size 448 x 448 and outputs the bounding box predictions for the given image **(Figure 3.1).** It reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities [22]. The algorithm divides the input image into an S x S grid (in the paper, S = 7). For each grid cell, it predicts B bounding boxes (in the paper, B = 2), where each bounding box consists of 4 coordinates and a confidence score for the prediction, and C class probabilities per grid cell, taking the highest one as the final class. All of these predictions are encoded as an S x S x (B * 5 + C) tensor, which is being outputted by the neural network **(Figure 3.2)**. What the algorithm finally does is identify objects in the image and map them to the grid cell containing the center of the

object. This grid cell will be responsible for predicting the final bounding box of the object and will have the highest confidence score.

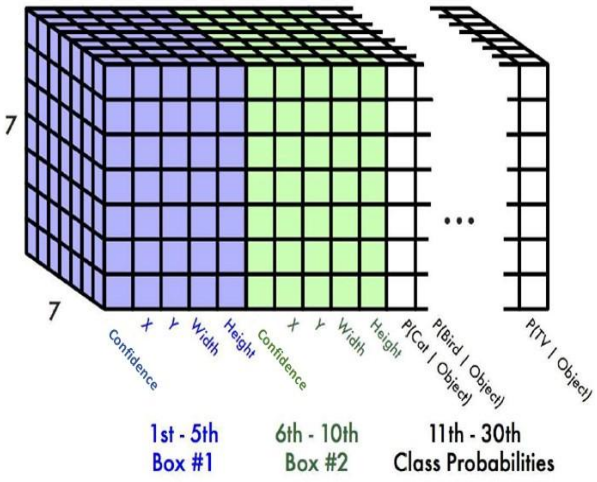

Figure 3.2: Output tensor of YOLO [23]

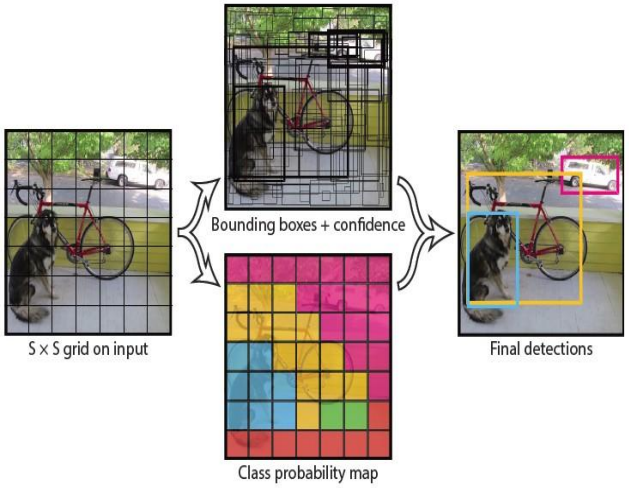

Figure 3.3: SxS grid and final detections [23]

In the example **(Figure 3.3),** each cell of the 7 x 7 grid is represented by a vector of size 30 representing a particular area of the image. Each vector contains 2 bounding box predictions (5 values each) and 20 conditional class probabilities P(class|object). The first step upon extracting a valid prediction is to choose the bounding box with the higher confidence score and check if the confidence score is above a predefined threshold (threshold = 0.25 in the paper) to output it as a valid prediction. This confidence score represents the prior in the conditional probability for the class prediction stating the probability that the given grid cell is the center of an object with a correct bounding box. To extract the class prediction YOLO outputs the conditional probability with the highest score. YOLO spatially defines each bounding box by four coordinates (X, Y, Width, Height), where (X, Y) represent the center of the bounding box relative to the cell, while (Width, Height) represent the width and the height of the bounding box relative to the whole image. Because of this, a bounding box can be bigger than the cell where it was predicted. The cell is only used as the anchor point for the prediction. One disadvantage of this approach is the fact that every cell is able to predict only one object. If multiple objects have their center points in the same cell, only one will be predicted.

3.1.1 Model Architecture

The model architecture consists of 24 convolutional layers followed by 4 pooling layers and 2 fully connected layers **(Figure 3.4)**. It uses 1 x 1 convolutions to reduce the number of feature maps, which is motivated by the Inception Modules of GoogLeNet [24]. Furthermore, it applies the Leaky ReLu activation function after all layers except for the last one and uses dropout between the two fully connected layers in order to tackle overfitting.

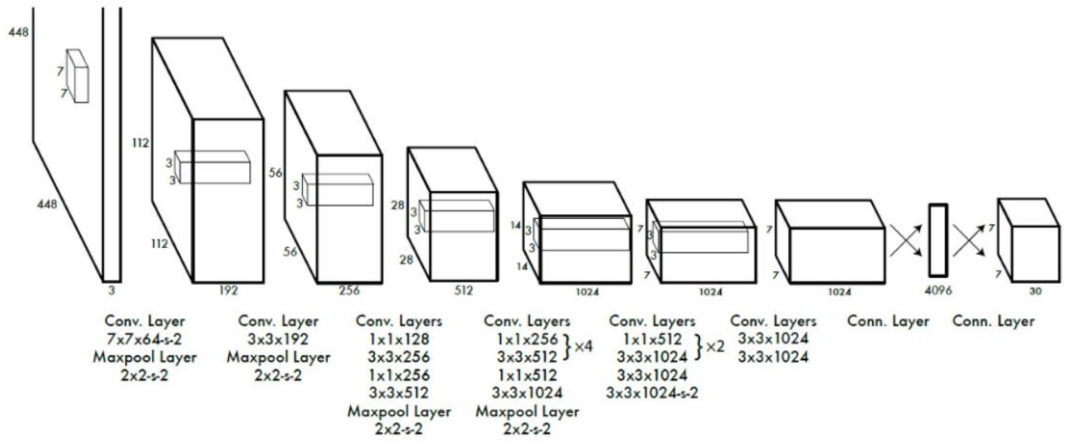

Figure 3.4: YOLO's model architecture

3.1.2 Loss Function

The YOLO algorithm uses a custom loss function to control the different output domains and their influence on the final loss by using special hyperparameters **(Figure 3.5)**:

1. First term: penalizes bad locations for the center coordinates if the cell contains an object.
2. Second term: penalizes bad bounding box width and height values. The square root is present so that errors in small bounding boxes are more penalizing than errors in big bounding boxes.
3. Third term: penalizes small confidence scores for cells containing an object.
4. Fourth term: penalizes big confidence scores for cells containing no object.
5. Fifth term: simple squared classification loss.

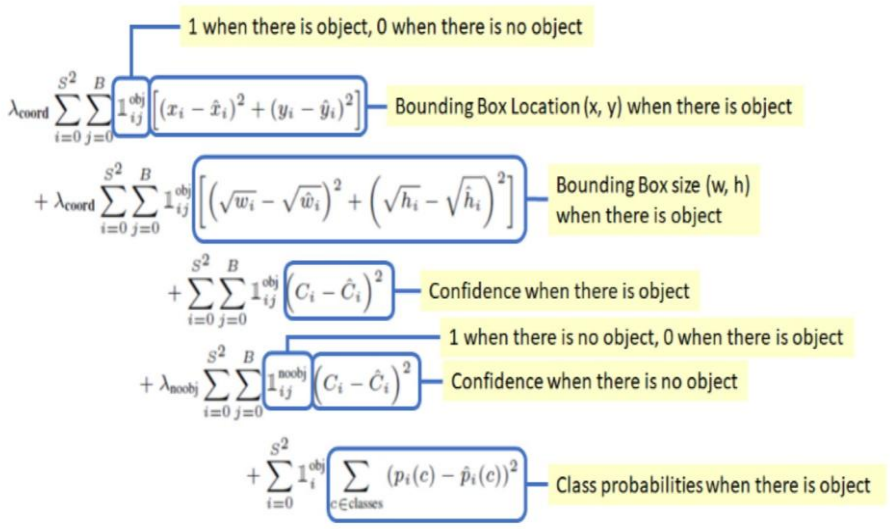

Figure 3.5: YOLO's custom loss function for every grid cell

3.1.3 Training YOLO on BDD100K

We implemented and trained YOLO completely from scratch by using only the BDD100K dataset. Since there are numerous objects inside the images of the BDD100K dataset, we decided to increase the split size for the YOLO algorithm from 7 to 14 in order to mitigate the problem of multiple object centres falling into one cell **(Figure 3.6)**. By deploying a much finer grid, we increased the number of output parameters from 1127 to 4508, as well as the number of parameters inside the last 5 layers. To achieve this, we adjusted the stride of the 23rd convolutional layer from 2 to 1 to retain the output size of 14 x 14.

Figure 3.6: YOLO with split size 14 on the BDD100K dataset

Furthermore, we added batch normalization between all layers to increase the training speed and retained the original loss hyperparameters from the paper during training (co-ord=5 and no-

obj=0.5). Finally, we trained YOLO for 100 epochs with a learning rate of 1e-5 and a batch size of 10. Our YOLO algorithm produces 2 bounding box predictions per grid cell on a 14 x 14 grid. The input image has dimensions (3, 448, 448), and the algorithm produces a tensor of size (14, 14, 23) as output.

3.2 Faster R-CNN

The architecture of Faster R-CNN [13] consists of three parts: the network backbone, the region proposal network (RPN), and the older version of this algorithm called fast R-CNN **(Figure 3.7)**. The network backbone is in general, a classification network like VGG-Net or ResNet pretrained on an image classification dataset. It is used to generate high-resolution feature maps and requires an image size of 640 x 640 pixels. In our approach, we used ResNet50 as the backbone network pretrained on the ImageNet dataset.

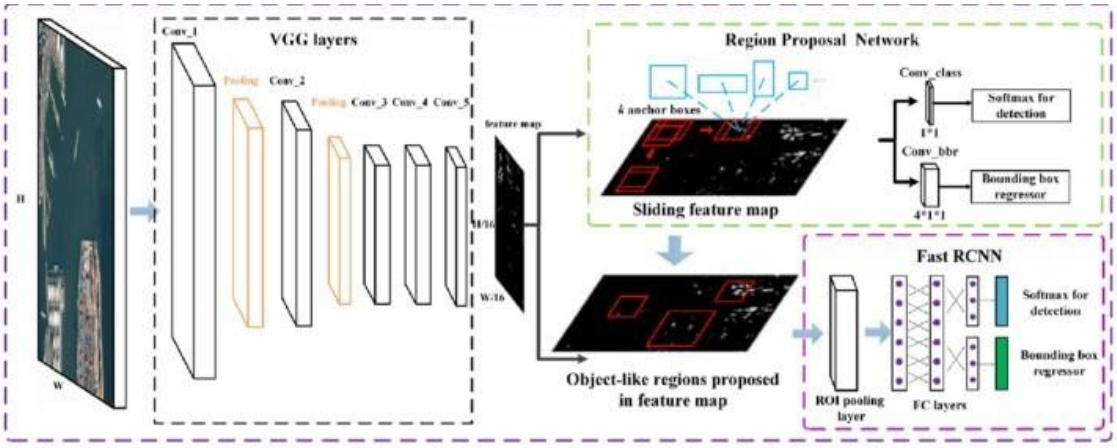

Figure 3.7: Faster R-CNN model architecture [25]

3.2.1 Region Proposal Network

The region proposal network consists of a single convolutional layer, which is then divided into 2 separate convolutional layers to predict a classification score and bounding boxes for the region proposals. The network uses predefined anchor boxes to generate approximately 2.000 region proposals. An anchor is a smaller part of an image. In our approach, we are using anchors with the sizes {128,256,512} with the aspect ratios of {0.5,1,2}, which leads to 9 different anchor boxes **(Figure 3.8)**. Each of these anchors will be slide over the window with a stride of 16 and cuts out the overlapping parts of the image.

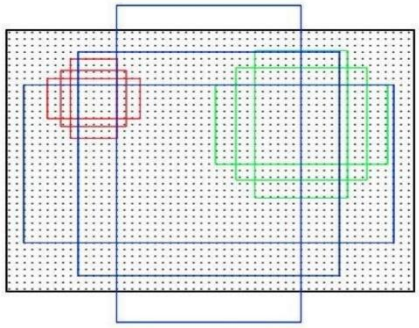

Figure 3.8: Anchor boxes with the sizes {128,256,512} with aspect ratios of {0.5,1,2} [26]

The classification score decides for all anchors whether they include an object or not. The bounding box regressions are for better identifying the objects in the anchors. After the prediction, the number of region proposals will be reduced with non-maximum suppression.

3.2.2 Fast R-CNN

Fast R-CNN, the last part of the architecture, gets the high-resolution feature maps from the network backbone and the region proposals from the region proposal network as input. To get a fixed size from the different-sized region proposals, a method called ROI pooling **(Figure 3.9)** is used, which stands for region of interest pooling.

For every region of interest from the input list, it takes a section of the input feature map that corresponds to it and scales it to some predefined size, in our case, 7x7. The scaling is done by:

Figure 3.9: Schematic representation of ROI pooling [27]

1. Dividing the region proposal into equal-sized sections, where the number of sections is the same as the dimension of the output.
2. Finding the largest value in each section.

3. Copying these max values to the output buffer.

After that, there are only 2 fully connected layers followed by two separate output layers, which predict the softmax score for every class and the corrected bounding boxes for every region proposal. As loss functions for the regression, they use log loss, and for the bounding boxes, they use smooth L1 Loss and backpropagate the losses together by factors that determine how much the bounding box and the classification loss should affect the entire loss.

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{x,y,w,h,\}} \text{smooth}_{L1}(t_i^u - v_i),$$

$$\text{smooth}_{L1}(x) = \begin{cases} 0.5x^2, & if\ |x| < 1 \\ |x| - 0.5, & otherwise \end{cases}$$

4 Experimental Results

4.1 Training

YOLO was trained for 80 Epochs with a decreasing learning rate of 1e-5 and batch size 10. Faster R-CNN was trained for 60 epochs with a decreasing learning rate of 1e-4 and batch size 16. The normalized total loss and the mAP of the training progress are shown in **Figure 4.1**.

4.2 Results

The implementation of the models and the processed data are found in this **GitHub repository.**

Furthermore, we also provide videos of real-time object detection with our models:

We measured the FPS and mAP for the evaluation and comparison of YOLO and faster R-CNN on a NVIDIA V100 SXM2 32GB (Figure 4.1).

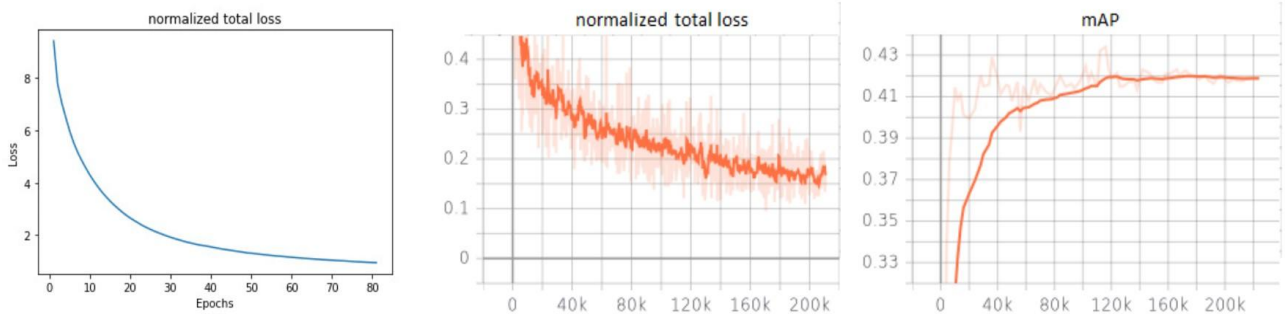

Figure 4.1: Normalized total loss for YOLO (left), normalized total loss for Faster R-CNN (middle) and mean average precision for Faster R-CNN (right)

Table 4.1: Comparison between YOLO, Faster R-CNN and Hybrid incremental net in mAP and FPS on the BDD100K dataset

	mAP	FPS
YOLO	18,6	212
FasterR-CNN	41,8	17,1
Hybridincrementalnet	45,7	N/A

Figure 4.2: Results on BDD100K: Faster R-CNN (left) and YOLO (right). More results from the YOLO and the Faster R-CNN architecture are shown in the appendix.

5 Conclusion and Future Work

In our project we implemented and trained a one-stage detector YOLO and a two-stage detector Faster R-CNN on the BDD100K dataset in the context of autonomous driving. As expected, the results of the evaluation showed that Faster R-CNN has a higher accuracy but lower FPS. In comparison YOLO has a much higher FPS, but also much lower accuracy because of its simple architecture. Future work includes further experiments with newer models, for example the newer versions of YOLO, since we used the first version of YOLO in this project. Future work in the long term would be reaching performances with high accuracy and high FPS which are suitable for the goal of autonomous driving.

Bibliography

[1] Girshick, R., Donahue, J., Darrell, T. & Malik, J. *Rich feature hierarchies for accurate object detection and semantic segmentation.* ~53.3% mAP on VOC2012 for R-CNN.

[2] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1440-1448).

[3] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, *28.*

[4] Redmon et al. You Only Look Once: *Unified, Real-Time Object Detection* (YOLO, 2015), single-stage, grid-based approach.

[5] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

[6] Community benchmark on YOLOv7-E6: ~55.9% AP at ~56 FPS outperforming transformer-based detectors on COCO.

[7] Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., ... & Darrell, T. (2020). Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2636-2645).

[8] Yu et al., *arXiv paper* May 2018 – description of ten tasks, geographic/environmental diversity, justification for multitask learning.

[9] BDD100K blog describing keyframe annotations for 100K videos and object bounding boxes.

[10] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[11] Prajjwal Bhargava. On generalizing detection models for unconstrained environments. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[12] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[14] 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.

[15] Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).

[16] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

[17] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

[18] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, September). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Cham: Springer International Publishing.

[19] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[20] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020.

[21] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[22] Manish Chablani. Yolo – you only look once, real-time object detection explained. https://towardsdatascience.com/ yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006 zuletzt besucht: 15.03.21, 2017.

[23] Sik-Ho Tsang. Review: YOLOv1, you only look once (object detection). https://towardsdatascience.com/ yolov1-you-only-look-once-object-detection-e1f3ffec8a89 zuletzt besucht: 15.03.21, 2018.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[25] Shilin Zhou Juanping Zhao Zhipeng Deng, Hao Sun. Multi-scale object detection in remote sensing imagery with convolutional neural networks. In *ISPRS Journal of Photogrammetry and Remote Sensing 145*, 2018.

[26] Mata. faster rcnn in rpn the anchor, sliding windows, proposals of understanding. https: //www.programmersought.com/article/31012543832/ zuletzt besucht: 15.03.21.

[27] Tomasz Grel. Region of interest pooling explained. https://deepsense.ai/ region-of-interest-pooling-explained/ zuletzt besucht: 15.03.21, 2017.

Appendices

Figure: Results on BDD100K: Faster R-CNN (left) and YOLO (right)