Escuela de Ingeniería Informática & Department of Computer Science and Communications

Final Year Project

Bachelor's Degree in Informatics Engineering
Computer Science

Evaluating YOLO in real computer vision tasks

Autor: Amine Mbarek

Escuela de Ingeniería Informática & Department of Computer Science and Communications

FINAL YEAR PROJECT

Bachelor's Degree in Informatics Engineering
Computer Science

Evaluating YOLO in real computer vision tasks

Autor: Amine Mbarek
Tutores: Valentín Cardeñoso Payo
Mohamed Hammami
Fatma Bouhlel

I dedicate my dissertation work:

To my loving parents, Sadok and Khiriya Mbarek, a special feeling of gratitude goes to you. Your unwavering support, encouragement, and insistence on tenacity have been a constant source of motivation and strength.

To my sister Ahlem and my brother Ahmed, who have never left my side. Your support has been very special to me, and I am deeply grateful for your love and understanding.

To all my professors, for their invaluable guidance and wisdom.

To all my friends, who offered their support and encouragement throughout this journey.

Acknowledgements

I wish to express my sincere gratitude and appreciation to everyone who assisted and supported me during this project.

First and foremost, I would like to express my deepest gratitude to my tutors, Mr. Valentín Cardeñoso Payo, Mr. Mohamed Hammami and Mme. Fatma Bouhlel, whose constant reviews, corrections, and guidance were fundamental to the development of this research. Their knowledge, patience, and support were invaluable throughout the entire process.

I am also grateful to the University of Valladolid and the University of Sfax for providing me with the opportunity to carry out this project. Their resources and academic environment significantly contributed to the successful completion of my research.

My thanks also go to all the professors at the Faculty of Sciences in Sfax for their invaluable help and their admirable human qualities, which I have greatly appreciated throughout my course.

Last but not least, I would like to express my gratitude to the Entornos de Computación Avanzada y Sistemas de Interacción Multimodal (ECA-SIMM) Group at the University of Valladolid for welcoming me as one of its members.

To all of you, I owe a great part of this achievement. Thank you very much.

Resumen

El objetivo principal de este trabajo es evaluar la efectividad del modelo YOLO en tareas reales de visión por computadora, específicamente en el contexto de vehículos autónomos, utilizando el conjunto de datos KITTI. La motivación surge de la necesidad de mejorar la seguridad en la conducción autónoma, donde la detección precisa de peatones, ciclistas y otros vehículos es crucial para evitar accidentes. Las tareas realizadas incluyen la investigación y análisis de soluciones actuales, la preparación y preprocesamiento del conjunto de datos KITTI, el entrenamiento y optimización del modelo YOLO, y la evaluación de su rendimiento en escenarios de conducción variados. Los resultados demuestran una mejora significativa en la precisión y eficiencia del sistema de detección de objetos, contribuyendo al desarrollo de tecnologías de conducción autónoma más seguras y confiables. En conclusión, este trabajo avanza en la comprensión y aplicación de la tecnología de detección de objetos, reduciendo potencialmente la incidencia de lesiones y muertes en el tráfico vial.

Abstract

The main objective of this project is to evaluate the effectiveness of the YOLO model in real-world computer vision tasks, specifically within the context of autonomous vehicles, using the KITTI dataset. The motivation stems from the need to improve safety in autonomous driving, where accurate detection of pedestrians, cyclists, and other vehicles is crucial to prevent accidents. Tasks performed include research and analysis of current solutions, preparation and preprocessing of the KITTI dataset, training and optimization of the YOLO model, and evaluation of its performance in various driving scenarios. The results demonstrate a significant improvement in the accuracy and efficiency of the object detection system, contributing to the development of safer and more reliable autonomous driving technologies. In conclusion, this work advances the understanding and application of object detection technology, potentially reducing the incidence of road traffic injuries and fatalities.

Résumé

L'objectif principal de ce projet est d'évaluer l'efficacité du modèle YOLO dans des tâches réelles de vision par ordinateur, spécifiquement dans le contexte des véhicules autonomes, en utilisant le jeu de données KITTI. La motivation découle de la nécessité d'améliorer la sécurité dans la conduite autonome, où la détection précise des piétons, des cyclistes et des autres véhicules est cruciale pour prévenir les accidents. Les tâches effectuées incluent la recherche et l'analyse des

solutions actuelles, la préparation et le prétraitement du jeu de données KITTI, l'entraînement et l'optimisation du modèle YOLO, et l'évaluation de ses performances dans divers scénarios de conduite. Les résultats démontrent une amélioration significative de la précision et de l'efficacité du système de détection d'objets, contribuant au développement de technologies de conduite autonome plus sûres et plus fiables. En conclusion, ce travail fait progresser la compréhension et l'application de la technologie de détection d'objets, réduisant potentiellement l'incidence des blessures et des décès liés au trafic routier.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over time, car safety systems have significantly advanced, beginning with fundamental features like seat belts introduced by Volvo in 1959 and crumple zones by Mercedes-Benz. The 1980s saw the widespread adoption of airbags and anti-lock braking systems (ABS). By the 1990s, electronic stability control (ESC) and side airbags became common. The 2000s brought advanced driver assistance systems (ADAS) such as lane departure warnings, adaptive cruise control (ACC), and blind-spot detection. Recent years have seen the emergence of autonomous features like automatic emergency braking (AEB) and pedestrian detection systems, with continued advancements towards fully autonomous vehicles and vehicle-to-everything (V2X) communication.

Despite these advancements, according to the World Health Organization, approximately 1.19 million people die each year due to road traffic crashes. Vulnerable road users, such as pedestrians, cyclists, and motorcyclists, account for more than half of all road traffic deaths. Given these statistics, the need for advanced object detection systems is evident. The implementation of such systems, particularly in autonomous vehicles, has the potential to significantly reduce these tragic numbers. Recent research shows that broadside and pedestrian collisions represent only about 5.7% of AV accidents, significantly lower than the 42.1% seen with conventional vehicles [33]. This significant reduction is achieved by accurately identifying and responding to various road hazards, pedestrians, and other vehicles.

This final project focuses on evaluating the YOLO (You Only Look Once) algorithm in real-world computer vision tasks relevant to autonomous driving. The objective is to assess the performance of YOLO using the KITTI dataset, which is renowned for its complexity and diversity in autonomous driving scenarios. We will delve into the architecture and functioning of the YOLO algorithm, examining how it can be optimized and adapted for use in the dynamic and complex environments that autonomous vehicles navigate. By leveraging the comprehensive data provided by the KITTI dataset, we aim to enhance the accuracy and efficiency of object detection systems in identifying various road hazards, pedestrians, cyclists, and other vehicles.

This thesis is presented in 9 chapters:

1. **Introduction**: Overview of car safety advancements and the need for better object

detection systems in autonomous vehicles.

2. **Objectives and Scope**: Outlines the project's primary goal of developing and evaluating an object detection system using YOLO and the KITTI dataset, along with specific tasks.

3. **Planning**: Breaks down the project phases, timeline, and estimated costs.

4. **Conceptual Framework**: Reviews machine learning and deep learning principles relevant to computer vision and object detection.

5. **State of the Art**: Explores the latest advancements and challenges in object detection for autonomous vehicles.

6. **You Only Look Once (YOLO)**: Discusses the YOLO algorithm's evolution, versions, and architectural improvements.

7. **Dataset**: Details the KITTI dataset, its components, and preparation for training the detection model.

8. **Experiments and Evaluation**: Describes the experimentation setup, evaluation metrics, and results.

9. **Conclusions**: Summarizes project contributions, results, and suggests future research areas.

Chapter 2

Objectives and Scope

2.1 Objectives

The primary objective of this project is to evaluate the effectiveness of the YOLO model in real-world computer vision tasks, specifically within the context of autonomous vehicles, using the KITTI dataset. This overarching goal is broken down into several secondary objectives that will guide the project's development. Additionally, specific tasks are outlined to achieve these objectives.

2.1.1 Principal Objective

- **Evaluate an Object Detection System:** The main objective is to evaluate the performance of the YOLO (You Only Look Once) model in real computer vision tasks by leveraging the KITTI dataset. This involves analyzing the model's effectiveness in accurately detecting various objects within the dynamic and complex environments encountered by autonomous vehicles.

2.1.2 Secondary Objectives

- **Understand YOLO:** Gain a comprehensive understanding of the YOLO (You Only Look Once) model, including its various versions, functionalities, and practical applications in computer vision.
- **Research and Analysis of Current Solutions:** Conduct thorough research and analysis of existing and prior solutions in the field of object detection for autonomous vehicles.
- **Dataset Preparation and Preprocessing:** Prepare and preprocess the KITTI dataset to ensure it is suitable for use with the YOLO model.
- **Model Training and Optimization:** Train and optimize the YOLO model to enhance its accuracy and efficiency in detecting objects.

- **Performance Evaluation:** Evaluate the performance of the trained YOLO model using standard metrics and tests in various driving scenarios.

2.2 Scope

The scope of this project encompasses the following key areas:

1. **Define the Work and Develop a Plan:** Clearly define the scope of the project and develop a detailed plan with specific stages and timelines.
2. **Study the Problem:** Thoroughly investigate the problem of object detection in autonomous vehicles, including specific challenges and requirements.
3. **Locate Similar Solutions:** Identify and analyze existing similar solutions, evaluating their strengths and weaknesses.
4. **Develop Our Solution:** Develop a customized solution based on the YOLO model, adapting it to the specific needs of the project.
5. **Test and Experiment:** Conduct tests and experiments with the developed solution, making adjustments and improvements based on the results obtained.

By completing these tasks, we aim to achieve the primary objective of effectively evaluating the YOLO model's performance in real-world computer vision tasks, thereby contributing to the advancement of object detection systems in autonomous vehicles. This evaluation is expected to provide valuable insights that enhance the safety and reliability of autonomous driving technologies.

Chapter 3

Planning

This chapter outlines the planned phases and associated costs for the development and evaluation of the YOLO model in real computer vision tasks.

3.1 Phases and costs

This section provides an overview of the planned phases and associated costs for the project development.

3.1.1 Phases

Table 3.1 outlines the planned phases of the project development along with their respective durations.

Activity Name	**Days**
Preliminary research and literature review	1 - 9
Definition of objectives and scope	10 - 15
Development of the conceptual framework	16 - 24
State-of-the-art and comparative analysis	25 - 33
Study and application of the YOLO algorithm	34 - 43
Dataset preparation	44 - 52
Design and execution of experiments	53 - 72
Evaluation and analysis of results	73 - 81
Writing the final document	82 - 87
Final review and corrections	88 - 90

Table 3.1: Planned phases of project development.

3.1.2 Costs

The following is a detailed breakdown of the estimated costs associated with each phase of the project:

- **Preliminary research and literature review**: 9 days, approximate cost of €270 for access to databases and scientific articles.
- **Definition of objectives and scope**: 6 days, time spent without additional cost.
- **Development of the conceptual framework**: 9 days, estimated time cost of €540 (based on a rate of €15/hour for 4 hours/day).
- **State-of-the-art and comparative analysis**: 9 days, estimated time cost of €540.
- **Study and application of the YOLO algorithm**: 10 days, estimated time cost of €600.
- **Dataset preparation**: 9 days, estimated cost of €540 in time and €450 in materials.
- **Design and execution of experiments**: 20 days, estimated cost of €1,200 in time and €75 for 75 hours of GPU A40 usage.
- **Evaluation and analysis of results**: 9 days, estimated time cost of €540.
- **Writing the final document**: 6 days, estimated time cost of €360.
- **Final review and corrections**: 3 days, estimated time cost of €180.

The total estimated cost of the project is €5745, considering both material resources and time spent.

Chapter 4

Conceptual Framework

4.1 Introduction

The conceptual framework of this research is grounded in the foundational principles and advancements of machine learning and deep learning. This chapter aims to provide a comprehensive overview of these fields, focusing on the algorithms, techniques, and architectures that form the backbone of modern computer vision systems. We will explore key concepts, such as supervised and unsupervised learning, delve into the evolution of deep learning with a particular emphasis on neural networks, and discuss the impact of convolutional neural networks (CNNs) in transforming the landscape of image processing and object detection. This framework sets the stage for understanding the methodologies and technologies applied in the subsequent chapters of this work.

4.2 Machine Learning

Machine Learning [4] is a branch of Artificial Intelligence that aims to design and develop algorithms that allow computers to improve their performance in performing a task from experience. However, unlike traditional programming, where software is developed based on pre-established rules, in Machine Learning, we let computers "program themselves" through the information in the provided data [27]. This process is known as training. Once this stage is completed, the obtained model will be able to obtain answers for new, unknown data sets.
The most widely adopted machine learning methods are supervised learning and unsupervised learning:

- **Supervised Learning**: supervised learning involves designing a model that connects training data to a set of output values. This means that the training data provided to the algorithm includes the desired solutions, known as labels. This method allows the algorithm to learn by comparing its actual output with the taught outputs to find errors and adjust the model accordingly. Supervised learning enables the model to predict label values on additional unlabeled data.

- **Unsupervised Learning**: in this case, the example data used to train the algorithm is not labeled, meaning we do not know the outcome. The goal is to find associations or patterns among the data based on their similarities. As examples of unsupervised learning techniques, we can differentiate between two groups: those oriented towards clustering (searching for groupings within the data), which include algorithms like k-means, k-medians, or self-organizing maps, and those oriented towards dimensionality reduction, such as principal component analysis [27].

4.3 Deep Learning

Deep learning is defined as a class of machine learning techniques that enable computational models composed of multiple processing layers to learn representations of data with multiple levels of abstraction [23]. This concept has emerged since the 2000s and it relies on artificial neural networks which are systems inspired by the biological neurons in the human brain. However, artificial neural networks have been exciting since the middle of the 20th century. The first artificial neuron was mentioned in 1943 when Warren McCulloch and Walter Pitts published their first mathematical and computational model of the biological neuron: the formal neuron, which is directly inspired by its biological counterpart.

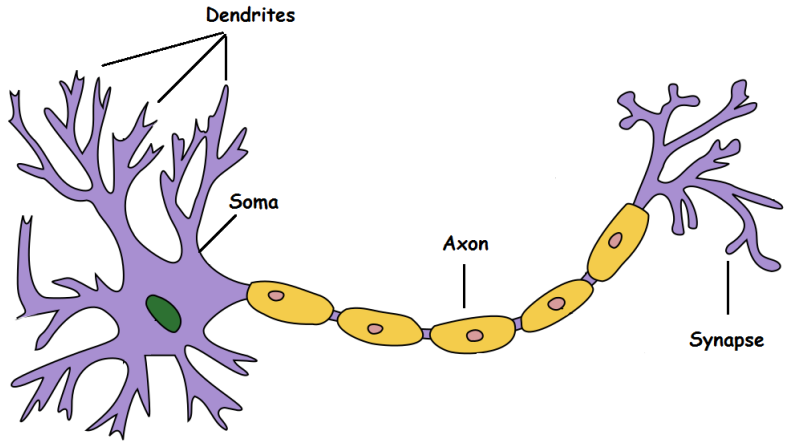


Figure 4.1: A biological neuron

A biological neuron is composed of a soma (cell body), an axon and dendrites as shown in Figure 4.1. The dendrites form a network of neural receptors that transmit electrical signals from other neurons to the neuron's body. A neuron receives electrical signals through the dendrites, and when the neuron receives enough signals in a given time (a few milliseconds), it triggers its own signals. The functioning of a neuron is relatively simple, but when a neuron is connected to thousands of others and there are billions of neurons, it creates networks capable of solving extremely complex problems. Warren McCulloch and Walter Pitts proposed the first artificial neuron which is a very simplified model of the biological neuron. The neuron shown in Figure 4.2 has one or more binary inputs and an output. The way it operates is simple:

the neuron activates its output (output active = 1) if its inputs exceed a certain threshold. From this neuron, it is possible to construct any network of artificial neurons capable of solving logical operations.

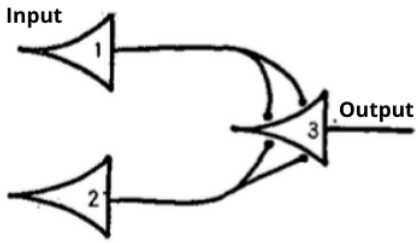


Figure 4.2: Formal neural network functioning as the OR logical operation.

Nowadays, the neuron architecture has developed and evolved significantly from its early conceptualization. As illustrated in the Figure 4.3, a modern artificial neuron consists of multiple inputs $(x_1, x_2, \ldots, x_n)$, each associated with a weight $(w_{1j}, w_{2j}, \ldots, w_{nj})$. These inputs are processed through a transfer function that computes the net input (net_j) by summing the weighted inputs. This net input is then passed through an activation function, which determines the neuron's output (o_j). The activation function introduces non-linearity into the model, enabling the network to capture complex patterns in data. Additionally, a threshold (θ_j) can be applied to control the activation. This intricate structure allows deep learning models to perform sophisticated tasks, learning representations from raw data through multiple layers of abstraction.

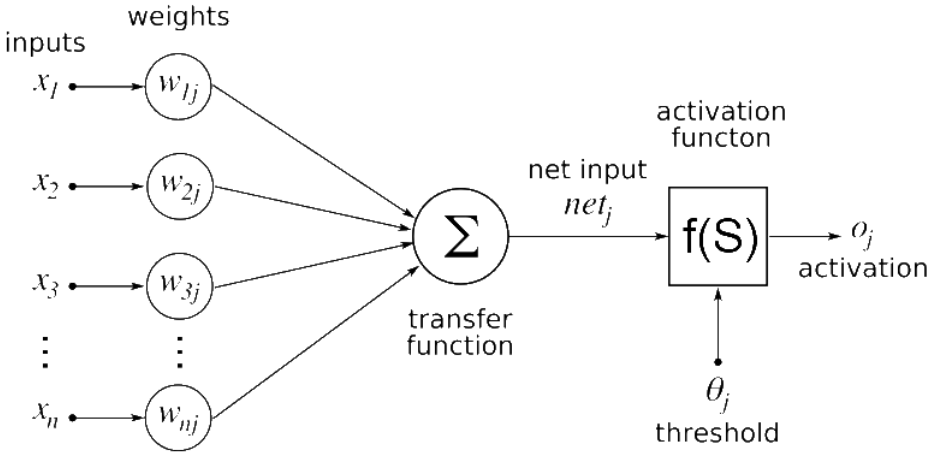


Figure 4.3: Artificial Neural Network

Deep learning uses neural networks with multiple layers to automatically learn and extract features from raw data. This process involves stacking many layers of artificial neurons, each layer learning increasingly abstract representations of the data. In contrast to traditional machine learning, where a significant portion of the process involves manual feature extraction by experts, deep learning automates this step. As illustrated in Figure 4.4, machine learning typically involves an initial step of feature extraction performed by a human, followed by

classification through algorithms such as decision trees. On the other hand, deep learning integrates feature extraction and classification into a single, end-to-end process handled entirely by the neural network. This integration allows deep learning models to achieve remarkable results across a wide range of applications such as voice recognition [7], surveillance [19], natural language processing (NLP) [30], and so forth.

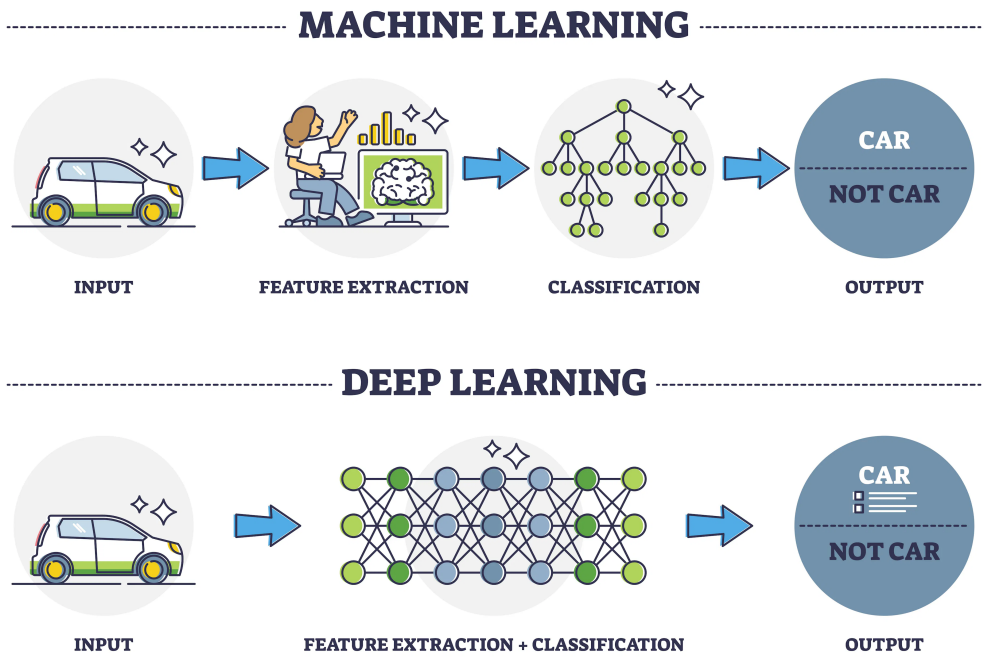


Figure 4.4: Diffrence between Machine learning and Deep Learning

Classification focuses on identifying the class of an image from the object it represents, as shown in Figure 4.5. In contrast, object detection focuses on identifying and locating objects within the image, as illustrated in Figure 4.6.

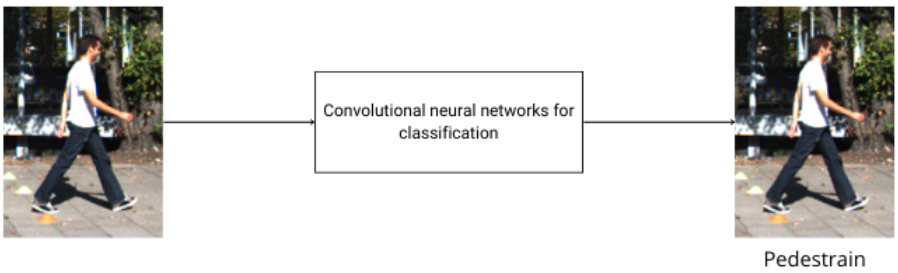


Figure 4.5: Pedestrian classification

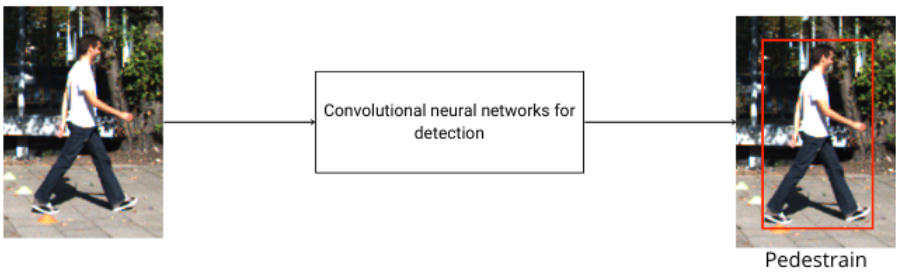


Figure 4.6: Pedestrian detection

4.3.1 Convolutional Neural Network

CNN is a popular and widely used algorithm in deep learning [34]. It is a type of deep learning model for processing data that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex [46]. It was first introduced by Kunihiko Fukushima and later developed by Yann LeCun, who integrated CNN with back-propagation theory to recognize handwritten digits and document recognition [11]. CNN is an extension of the multilayer perceptrons (MLPs) designed to effectively address their main disadvantages [47]. It aims to limit the number of inputs while preserving the strong 'spatially local' correlation of natural images. CNN automatically extracts features from input images, is resistant to slight distortions, and uses weight sharing to reduce the number of network parameters [47].

Let's see how a convolutional neural network works. A CNN architecture is formed by a stack of processing layers: convolutional layer (CONV), pooling layer (POOL), fully connected layer (FC), and a softmax layer for classification. The last layer can be substituted with another type of classifier such as a Support Vector Machine (SVM) (*cf.* Figure 4.7).

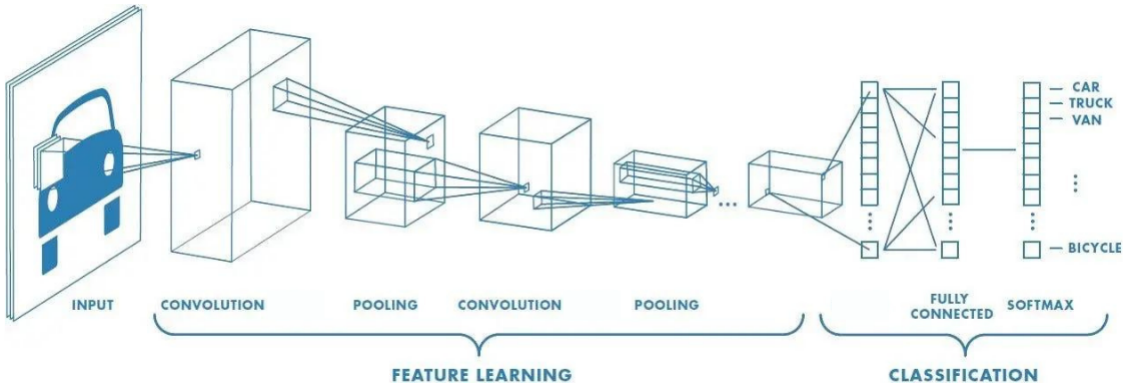


Figure 4.7: Standard architecture of CNN

4.3.2 Hyper-parameters of Convolutional Neural Network

The performance and efficiency of CNN are significantly influenced by a set of hyperparameters. These hyperparameters are crucial as they control the network's architecture, training process, and overall behavior. Here, we delve into some of the essential hyperparameters of CNNs, their roles, and their impact on network performance.

- Number of layers: It defines the depth of the network.

- Number of filters: It represents the number of times a filter has been applied within a layer.

- Size of filter (f)

- Padding(p): It involves inserting zeros at the edges of the input image to control the size of the resulting image (*cf.* Figure 4.8). Indeed, following the application of a filter to an image, the size of the image is reduced. Consequently, the pixels at the edges of the input image are not considered if a filter is applied to the filtered image. The value of p, which allows preserving the size of the image after applying a filter, is calculated as follows in the case where a stride $s = 1$:

$$p = \frac{f - 1}{2} \tag{4.1}$$

- Stride (s): It is the step size at which we move the kernel across the input (*cf.* Figure 4.9). The horizontal stride represents the step size at which we move the kernel horizontally across the image, while the vertical stride represents the step size at which we move the kernel vertically across the image. To simplify, we will use the same value for both the horizontal and vertical strides to simplify the model.

- Pooling strategy: This is a function applied within the pooling layer to extract, for example, the maximum value ('Max pooling') or the average value ('Average pooling') (*cf.* Figure 4.10) from a block of size ($f \times f$).

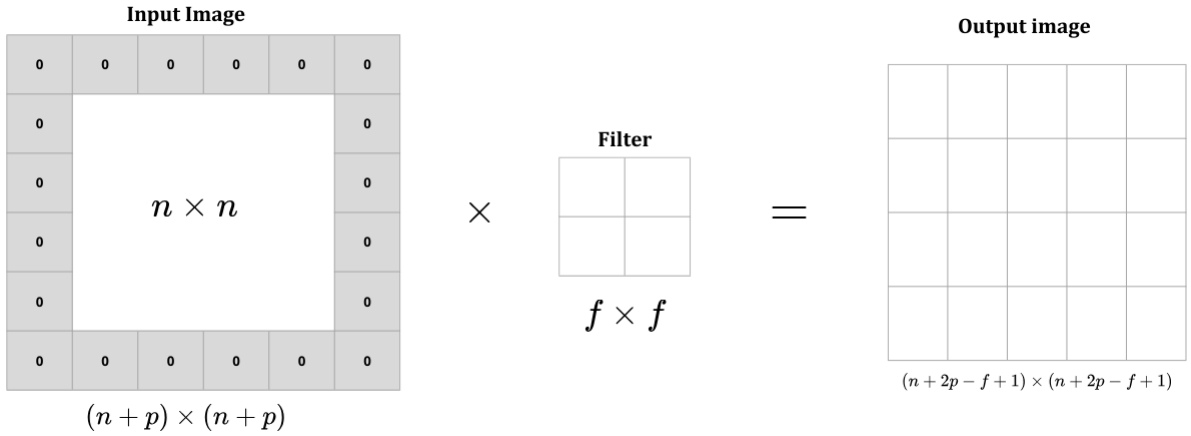


Figure 4.8: Application of a filter on an image with padding $p = 1$

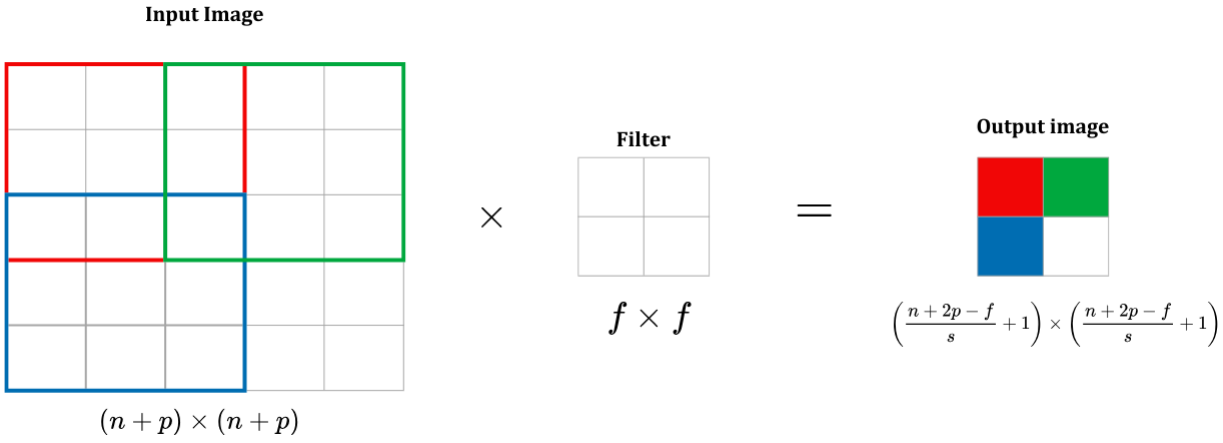


Figure 4.9: Application of a filter on an image with padding $p = 0$ and $s = 2$

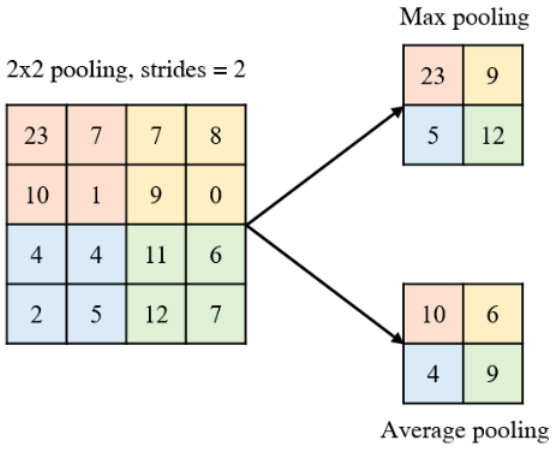


Figure 4.10: Pooling Process

Convolutional Layer

A convolution involves sliding a kernel over the input feature map, computing the product between each element of the kernel and the input element it overlaps, and summing the results to obtain the output at each location [8]. It produces an output image called a "feature map". In general, multiple filters are applied to progressively extract features from the image (edges, regions, textures, and more). The convolutional layer introduces non-linearity transformations through activation functions. Usually, activation functions follow the application of the convolution product. There are various activation functions, such as Rectified Linear Units (ReLU), Hyperbolic Tangent (TanH), and Linear (*cf.* Figure 4.11). However, ReLU has been found to achieve better performance in most situations and it is defined as follows [5]:

$$a = \max(0, x) \tag{4.2}$$

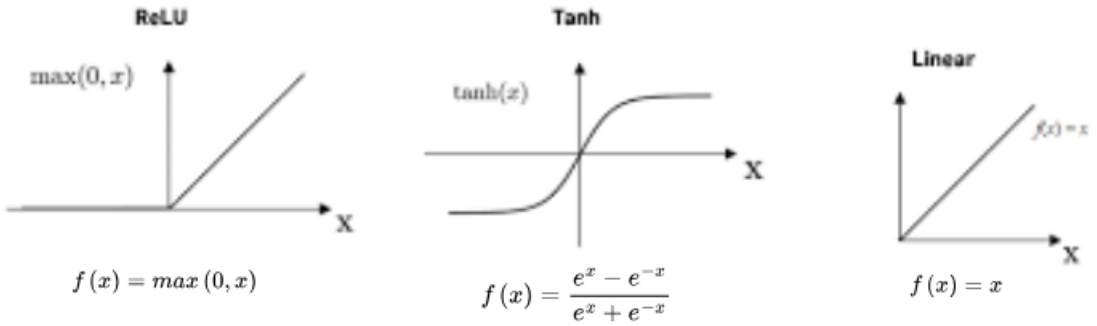


Figure 4.11: Activation functions

Pooling Layer

The pooling layer is often placed between two convolutional layers. It receives multiple feature maps as input and then reduces the size of each one. In simpler terms, the pooling operation involves reducing the size of the feature maps while preserving their important characteristics. There are different pooling strategies, including max pooling, which selects the maximum value from the input data, average pooling, which computes the average value of the input data (*cf.* Figure 4.10), and probabilistic pooling, which randomly selects a value from the input data [5].

Fully Connected Layer

The output feature maps from the final convolutional or pooling layer are transformed into a one-dimensional array of numbers and connected to one or more fully connected layers. In these layers, each input is weighted and connected to every output. Using a fully connected layer to learn non-linear combinations of these features is an effective strategy. As a result, fully connected layers are frequently used as the final layers in a CNN. They compute the weighted sum of the features from the previous layer, incorporating the specific parameter inputs to achieve a desired output result.

Softmax Layer

The softmax layer is a loss layer that uses the probabilities returned by the softmax activation function to determine the class of an image. This function normalizes the output of the final layer of the convolutional neural network to produce a vector of n real numbers (n corresponds to the number of classes). These numbers, whose sum equals 1, reflect the probabilities that the input image matches each of the original classes.

4.4 Computer Vision

Computer vision aims to design systems capable of analyzing and interpreting digital images and videos using filtering techniques. These systems employ algorithms and mathematical models to extract relevant information from visual data. Computer vision thus enables the replacement of the human eye and brain for observation and judgment, offering numerous possibilities for research and industry [16].
Since the 1950s, scientists and engineers have strived to develop methods allowing machines to perceive and understand visual data. The first experiments date back to 1959 when neurophysiologists exposed a cat to various images to observe the correlation with its brain's responses. They found that the cat initially reacted to contours and sharp lines, which scientifically indicated that image processing began with simple shapes such as straight edges. Around the same time, the first computer image scanning technology was developed, enabling computers to scan and acquire images. Another significant milestone was reached in 1963 when computers succeeded in converting two-dimensional images into three-dimensional shapes. In the 1960s, artificial intelligence emerged as an academic field of study, marking the beginning of the quest for artificial intelligence to solve the challenge of human vision [20]. In 1974, optical character recognition (OCR) technology was introduced, allowing the recognition of printed text in any font. Similarly, intelligent character recognition (ICR) could decipher handwritten text using neural networks. Since then, OCR and ICR have found their use in document and invoice processing, license plate recognition, mobile payments, machine translation, and other commonly used applications [20].
In the year 2000, research primarily focused on object recognition, while in 2001, the first real-time facial recognition applications appeared. The standardization of labeling and annotating visual datasets began to develop in the 2000s [20].
Here are some notable advancements in convolutional networks, ordered chronologically:

- **LeNet**: LeNet, a 7-level convolutional network introduced by LeCun et al. in 1998 [42], which classifies digits, has been applied by several banks to recognize handwritten numbers on scanned checks in 32x32 pixels. The ability to process higher resolution images requires more and larger convolutional layers, thus this technique is limited by the availability of computational resources.

- **AlexNet**: Initially, AlexNet was written with CUDA to run with GPU support, which

participated in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. AlexNet significantly outperformed all its previous competitors and won the challenge by reducing the top-5 error from 26% to 15.3%. It was designed by the Supervision group, consisting of Alex Krizhevsky [22]. AlexNet had a major impact in the field of machine learning, particularly in the application of deep learning to computer vision. AlexNet contained eight layers; the first five were convolutional layers, and the last three were fully connected layers. It had two parallel CNN pipelines trained on two GPUs with cross-connections and used the non-saturating activation function ReLU, which showed improved training performance compared to Tanh and Sigmoid.

- **ZFNet**: ZFNet was proposed by Matthew Zeiler and Rob Fergus [48]. ZFNet won the ILSVRC 2013. It was an improvement on AlexNet by fine-tuning the architecture hyperparameters, notably by increasing the size of the intermediate convolutional layers and making the stride and filter size on the first layer smaller.

- **VGGNet**: VGGNet secured first place at the ILSVRC competition in 2014 for the VGG group. It is an improvement over AlexNet and consists of 19 layers. This architecture strikes a balance between the depth of the CNN, which positively impacts its performance, and its requirements in terms of memory and computation time [40].

- **GoogleNet**: GoogleNet was invented by Szegedy et al. [43] and it was the winner of the ILSVRC 2014. Its main contribution was the addition of an inception module. This module applies convolutions of different sizes to extract features at various scales.

- **ResNet**: Residual Network, developed by Kaiming He et al., won the ILSVRC 2015 [17]. The ResNet architecture is composed of 152 layers, making it seven times deeper than previous architectures. This number of layers has had a positive impact on the correct classification rate. ResNet surpasses the performance of VGGNet and GoogLeNet architectures in terms of correct classification rates. However, it is computationally intensive.

These advancements in computer vision have laid the groundwork for more complex tasks, such as object detection, which involves identifying and localizing objects within an image. The next section will delve into the evolution and significance of object detection in the realm of computer vision.

4.5 Conclusion

This chapter has provided a comprehensive overview of the conceptual framework underpinning our research, focusing on the essential aspects of machine learning and deep learning, with a particular emphasis on convolutional neural networks. We have explored the historical context, theoretical foundations, and practical implementations of these technologies, illustrating their evolution and significance in the field of computer vision. By understanding these concepts,

we have established a solid foundation for the subsequent chapters, where we will apply these principles to specific tasks in autonomous driving and object detection.

Chapter 5

State of the Art

5.1 Introduction

The field of object detection has witnessed significant advancements in recent years, driven by the development of sophisticated algorithms and the increasing computational power available for training complex models. This chapter provides an overview of the current state-of-the-art techniques in object detection, highlighting key methodologies and their applications, particularly in the domain of autonomous vehicles. We will explore various convolutional neural network (CNN) architectures that have revolutionized object detection, discuss the specific challenges faced in this field, and review the methods developed to address these challenges. This exploration sets the stage for understanding the technical foundation upon which modern object detection systems are built.

5.2 Object Detection

Object detection is important in computer vision which aims to identify and locate objects present in an image or video. It uses convolutional neural networks (CNNs) to analyze and classify specific features of the image, such as edges and shapes. Then, it uses this information to identify objects and their locations in the image. It plays an essential role in many applications, such as autonomous vehicles where an understanding of the visual environment is required the needs of object detection include accuracy, speed, the ability to process scenes and adaptation to different object categories.

Several specialized CNN architectures have been developed for this purpose. Region-based CNNs (R-CNN) was proposed by Ross Girshick in 2014. As illustrated in Figure 5.1, This model introduced a three-module approach: Region Proposal Generation, Feature Extraction, and Classification and Localization [15]. The Region Proposal Generation module defines the set of candidate detections available for the detector. The Feature Extraction module uses a large convolutional neural network to extract a fixed-length feature vector from each region. Finally, the Classification and Localization module utilizes class-specific linear SVMs to classify

and precisely locate objects. The R-CNN achieved a mean average precision (mAP) de 53.7% on PASCAL VOC 2010 and 31.4% on the 200-class detection dataset ILSVRC2013 [15]. It outperformed the best previous model OverFeat [38], which had a top result of 24.3%.

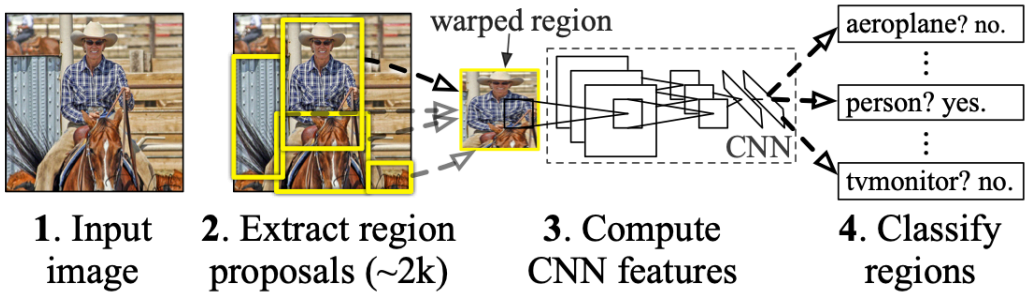


Figure 5.1: R-CNN Architecture: Regions with CNN features

This approach was later refined in Fast R-CNN by Ross Girshick [14] which improved speed and accuracy. Fast R-CNN takes an entire image and a set of object proposals as input. First, it processes the entire image with several convolutional (conv) and max pooling layers to produce a feature map. Next, for each object proposal, a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is then fed into a sequence of fully connected (FC) layers, which ultimately branch into two sibling output layers: One layer produces softmax probability estimates over K object classes plus a "background" class and one layer produces four real-valued numbers for each of the K object classes. Each set of 4 values encodes the refined bounding box positions for one of the K classes. Fast R-CNN achieves a result of 66.1% on VOC2010 and the best result on VOC12 with a mAP of 65.7% (and 68.4% with additional data) [14].

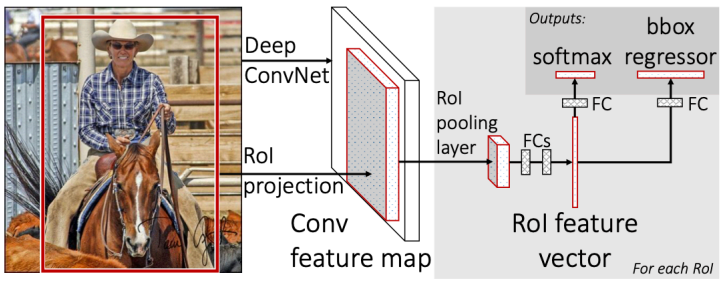


Figure 5.2: Fast R-CNN Architecture

In 2016, Shaoqing Ren proposed Faster R-CNN [18]. Faster R-CNN uses the proposed regions in Fast-CNN and introduced a Region Proposal Network (RPN), that takes the feature map (of any size) and produces a set of rectangular object proposals, each with a precision score (*cf.* Figure 5.3). The result obtained by this model with the mscoco (COCO val) database is a mAP@0.5 of 41.5% and a mAP@0.5 of 42.7% on COCO test-dev.

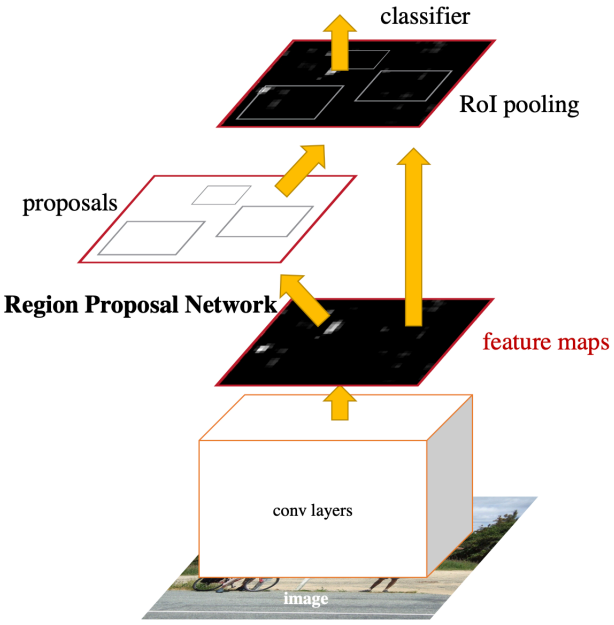


Figure 5.3: Faster R-CNN Architecture

Other notable architectures include YOLO (You Only Look Once), which will be discussed in chapter 6.

5.3 Autonomous vehicle

An autonomous vehicle is a type of vehicle that uses advanced technologies to operate without human intervention. It is equipped with cameras and other technologies to perceive the environment and make appropriate decisions. Automakers have developed these vehicles to improve road safety, considering the increase in fatal accidents on the roads.

5.4 Object Detection for Autonomous Vehicles

5.4.1 Challenges in Object Detection for Autonomous Vehicles

The challenges in object detection for autonomous vehicles are multifaceted and arise from various technical and environmental factors. Some of the key challenges include:

- **Scale and Shape Variability**: Different sizes of objects such as cars and cyclists, and multiple objects with different aspect ratios within the same window pose significant challenges for detection systems.
- **Occlusion and Overlapping Objects**: Detecting objects that are partially visible or

occluded by other objects requires advanced algorithms capable of handling incomplete information.

- **Real-Time Processing**: Real-time detection and interpretation of objects with high accuracy and speed are critical for the safety and efficiency of autonomous vehicles. This includes minimizing false positives and negatives.

These challenges highlight the need for continuous advancements in object detection algorithms, integration of robust models, and development of comprehensive datasets to improve the performance and reliability of autonomous vehicles.

5.4.2 Methods for object detection for Autonomous Vehicles

Object detection methods can be broadly categorized into two types: two-stage detectors and one-stage detectors.

Two stage detectors

Two-stage detectors first generate a set of object proposals and then refine these proposals to achieve precise detection. This approach typically involves a proposal generation stage followed by a classification stage.

- **Regionlets [26]:** Regionlets propose boxes through Selective Search and then re-localize them using a top-down approach. It extracts 1-D features from rectangular regions within the detection window and trains thousands of weak classifiers using RealBoost. These features are transformed into sparse binary vectors, which are then combined to form a high-dimensional feature space for robust detection. The model also includes a precise localization component that adjusts bounding box coordinates based on predicted localization errors, learned through support vector regression. This approach uses a voting mechanism for better accuracy and mitigates overfitting by sampling numerous bounding boxes around ground truth objects during training.

- **DPM-VOC+VP [32]:** DPM-VOC+VP extend the Deformable Part-based Model [10], which is an object detection method that uses a collection of deformable parts and discriminative training to accurately detect and localize objects within images, to 3D by connecting parts across various viewpoints and incorporating a 3D-aware loss function.

- **Mono3D [3]:** Mono3D generates 3D object proposals from a single monocular image using an energy minimization approach that places object candidates in 3D space and scores them based on semantic segmentation, context, size, location priors, and typical object shape (*c.f* Figure 5.4).

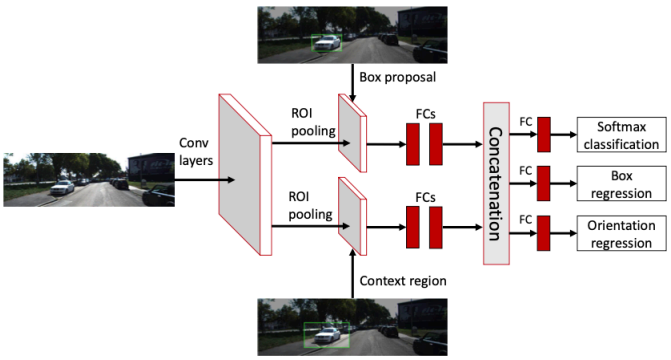


Figure 5.4: Mono3D architecture

- **LSVM-MDPM-sv [9, 10]:** LSVM-MDPM-sv is an adaptation of the DPM that utilizes latent support vector machines (LSVM) for supervised learning to improve object detection by training on a representative subset of clean training samples and fine-tuning the model parameters.

One stage detectors

One-stage detectors perform object localization and classification in a single forward pass through the network, offering faster processing times at the potential cost of lower accuracy compared to two-stage detectors.

- **Complexer-YOLO [39]:** Complexer-YOLO is a real-time 3D object detection that operates on semantic point clouds by incorporating visual point-wise class features predicting real 3D object dimensions.

5.5 Conclusion

In this chapter, we have examined the cutting-edge advancements in object detection, focusing on the evolution from traditional methods to advanced CNN architectures like R-CNN, Fast R-CNN, and Faster R-CNN. These technologies have significantly improved the accuracy and speed of detecting and localizing objects in images, which is critical for applications such as autonomous vehicles. We have also discussed the unique challenges in object detection for autonomous vehicles, emphasizing the need for real-time processing and robust algorithms to handle scale variability, occlusions, and overlapping objects. Understanding these state-of-the-art techniques and their applications provides a comprehensive foundation for the subsequent discussions on the implementation and evaluation of object detection systems in our research.

Chapter 6

You Only Look Once

YOLO appeared in 2015 and since then has become a standard in ANN applied to computer vision. This chapter summarises the main motivation for this work, an overview and objectives.

6.1 Scope

Nowadays, in the world of computers, finding and recognizing objects in pictures or videos is highly important. It helps in various things like making self-driving cars safer, keeping an eye on places with surveillance cameras, and even helping doctors spot diseases in medical scans.
Among the algorithms developed for this purpose, YOLO, "You Only Look Once", stands out as a new approach to object detection known for its balance between speed and latency. It has become almost a standard way of detecting objects in computer vision.
The primary objective of this chapter is to delve into YOLO, the difference between this approach and old algorithms, its architectural design, and methodology, giving a close examination of its impact on object detection.
We'll see how YOLO's efficient single-network approach is different from the old way of detecting objects, paving the way for real-time applications across various domains.

6.2 A revolution in object detection

YOLO is a state-of-the-art, real-time object detection algorithm and is extremely fast and accurate. Before YOLO's emergence in 2015, Systems used sliding window object detection and iterations such as RCNN, fast RCNN, and faster RCNN they segment an image into numerous regions and then run a classifier on each one, and high scores for that classifier would be considered detections in the image. This process required thousands of classifier runs and neural network evaluations per image to achieve detection. With YOLO, instead of looking at an image thousands of times to produce detection, You Only Look Once, YOLO only needs the image or video to pass one time through its network (*c.f* Figure 6.1), and that's why it is called YOLO.

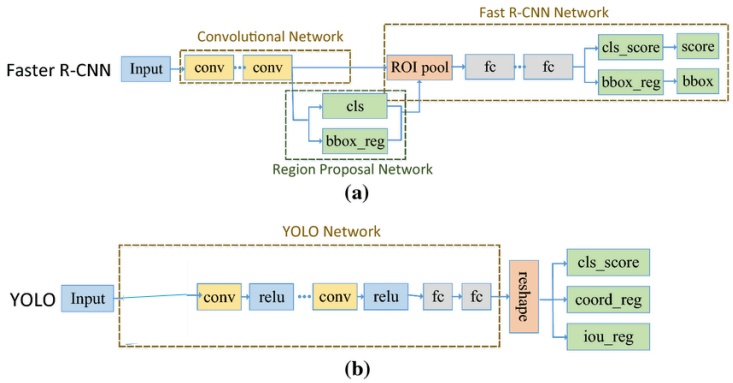


Figure 6.1: The network structure of a faster R-CNN and YOLO

However, YOLO outperformed most of the previous object detection algorithms (*c.f* Table 6.1).

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Table 6.1: Real Time systems on PASCAL VOC 2007 [36]

.

The primary methodology of YOLO is to resize the image, process it through a single convolutional network, and then refine the resulting detections based on the model's confidence thresholds (*c.f* Figure 6.2).

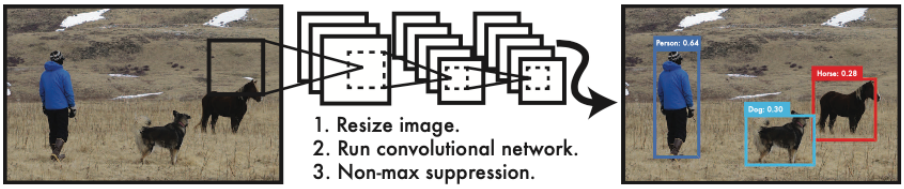


Figure 6.2: YOLO resizes the image, runs a single convolutional network on the image and then thresholds the resulting detections by the model's confidence [36]

.

6.3 Versions

YOLO has evolved through several versions, with each new version bringing advancements and betterment to the initial mechanism.

6.3.1 YOLO

YOLO [36] is the Original version known for speed, with lower accuracy. It resizes the image to 448 × 448, runs a single convolutional network, that has 24 convolutional layers followed by 2 fully connected layers, on the image and then thresholds the resulting detections by the model's confidence.
This version introduced the initial methodology: architecture, unified detection and training, which are described in the following sections.

Unified detection

Unified detection refers to the simultaneous prediction of bounding boxes and class probabilities across multiple scales within a single network. YOLO divides the image into an $S \times S$ grid. If the center of an object falls within a grid cell, that cell is responsible for detecting that object. Additionally, each cell predicts B bounding boxes and corresponding confidence scores. The confidence scores indicate the model's certainty that a box contains an object and the accuracy of the box prediction. Confidence is defined as $\Pr(\text{Object}) \times \text{IOU}_{\text{truth}}^{\text{pred}}$. If no object is predicted in a cell, the confidence score should be zero. Conversely, if an object is present, the confidence score should reflect how well the predicted box matches the real object.
Each bounding box consists of five predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the grid cell boundaries. The width and height (w, h) are predicted relative to the entire image. The confidence score reflects the accuracy of the predicted box compared to the actual object, calculated using the intersection over union (IOU) with the ground truth.
Furthermore, each grid cell predicts C conditional class probabilities $\Pr(\text{Class}_i \mid \text{Object})$. These probabilities are conditioned on the presence of an object within the grid cell. Only one set of class probabilities is predicted per grid cell, regardless of the number of bounding boxes B.
Figure 6.3 illustrates how the image is divided into an $S \times S$ grid, with each grid cell predicting B bounding boxes, the associated confidence scores, and C class probabilities. These predictions are encoded as an $S \times S \times (B \times 5 + C)$ tensor.

Architecture

YOLO implements a convolutional neural network, the initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. It has 24 convolutional layers followed by 2 fully connected layers and uses 1 × 1 reduction layers followed by 3 × 3 convolutional layers (*c.f* Figure 6.4).

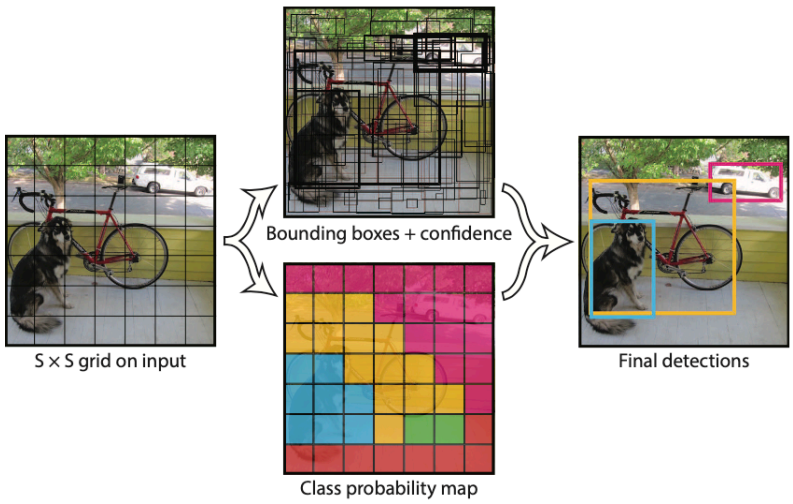


Figure 6.3: YOLO's Model [36].

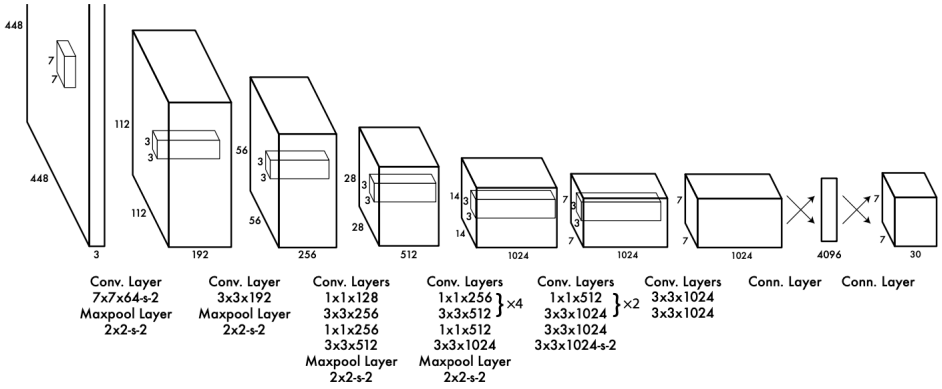


Figure 6.4: YOLO's Architecture [36].

Training

The training process started with pretraining the convolutional layers on the ImageNet 1000-class competition dataset. Specifically, the first 20 convolutional layers are utilized, followed by an average-pooling layer and a fully connected layer.

After approximately a week of training, the model achieves an impressive single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to models found in Caffe's Model Zoo. Subsequently, the architecture is adapted for object detection. To further augment its capabilities, the developers integrated four additional convolutional layers and two fully connected layers into the network. Consequently, they adjusted the input resolution (from 224×224 to 448×448) and refined the process of predicting class probabilities and bounding box coordinates. Also, to enhance accuracy, they normalized bounding box dimensions and introduced specialized predictors per object. This tailored approach maximizes detection performance and recall.

6.3.2 YOLOv2

YOLOv2 [36], an evolution of the original YOLO, builds upon the original YOLO architecture with enhancements aimed at improving accuracy and speed. YOLOv2 introduces several enhancements, including the use of batch normalization, high-resolution classifiers, anchor boxes and dimension clustering.

1. Batch Normalization: By adding batch normalization to all of the convolutional layers in YOLO, there was more than 2% improvement in mAP.

2. High-Resolution Classifier: YOLOv2 was trained at 448×448 images for 10 epochs on ImageNet then it was fine-tuned. This helped increase of almost 4% mAP.

3. Anchor Boxes: Instead of predicting bounding boxes directly, YOLOv2 predicts anchor boxes, which are predefined boxes of various sizes and aspect ratios(input image of 416 and output feature of 13×13). These anchor boxes improve localization accuracy by providing better initial estimates for object-bound boxes. The results are 69.5 mAP with a recall of 81%. With anchor boxes, the model gets 69.2 mAP with a recall of 88%.

4. Dimension Clustering: YOLOv2 utilizes dimension clustering to identify the most representative bounding box dimensions in the dataset. By clustering the dimensions of ground-truth bounding boxes, YOLOv2 can learn to predict more accurate bounding boxes.

6.3.3 YOLOv3

Building upon YOLOv2, YOLOv3 [35] represents a significant advancement in real-time object detection. YOLOv3 introduced several improvements including Bounding Box Prediction, Class Prediction, Predictions Across Scales, Feature Extractor and Training.

1. **Bounding Box Prediction:** Following YOLOv2, YOLOv3 predicts bounding boxes using dimension clusters as anchor boxes. The network predicts four coordinates for each bounding box: t_x, t_y, t_w, and t_h. These coordinates correspond to the center of the box and its width and height. During training, the network uses the sum of squared error loss to compute the ground truth value for each coordinate prediction.

2. **Class Prediction:** Each box predicts the classes the bounding box may contain using multilabel classification. Instead of using a softmax, YOLOv3 uses independent logistic classifiers for class predictions. This approach is beneficial for complex datasets with overlapping labels, as it better models the data.

3. **Predictions Across Scales:** YOLOv3 predicts boxes at 3 different scales and extracts features from those scales using a concept similar to feature pyramid networks. The network then upsamples and merges the features from different layers to obtain more

	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 6.2: Darknet-53 [35]

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 6.3: Comparison of backbones [35]

meaningful semantic information and finer-grained details. This process allows YOLOv3 to predict boxes for the final scale, benefiting from prior computation and fine-grained features from early on in the network.

4. **Feature Extractor:** YOLOv3 uses a new network for feature extraction called Darknet-53, which is a hybrid approach between the network used in YOLOv2, Darknet-19, and residual network architecture. Darknet-53 has shortcut connections and is significantly larger, with 53 convolutional layers as shown in Table 6.2. Moreover, table 6.3 confirms that it performs similarly to state-of-the-art classifiers but with fewer floating point operations and more speed.

5. **Training:** YOLOv3 is trained on full images with multi-scale training, data augmentation, and batch normalization. The Darknet neural network framework is used for

training and testing.

6.3.4 YOLOv4

YOLOv4 [1] consists of:

1. Backbone: CSPDarknet53 which is a modified version of the Darknet architecture. The CSPDarknet53 backbone is designed to efficiently handle large-scale datasets and tasks, making it suitable for real-time object detection.

2. Neck: PANet which is utilized for instance segmentation due to its proficiency in maintaining spatial information, which is crucial for precise pixel localization during mask generation.

3. Head: based on YOLOv3 architecture, which is an anchor-based detection head. This architecture is optimized for high-quality and real-time object detection.

Moreover, it introduces several innovative concepts, including "Bag of Freebies" (BoF) and "Bag of Specials" (BoS) to improve both the backbone and detector components.

1. Bag of Freebies (BoF): YOLOv4 integrates several BoF techniques, including DropBlock regularization, Class label smoothing, and CIoU loss. These techniques are aimed at improving the accuracy and robustness of the detector without sacrificing speed.

2. Bag of Specials (BoS): YOLOv4 integrates several BoS techniques, such as Mish activation, SPP-block, SAM-block, PAN path-aggregation block, and DIoU-NMS. These techniques enhance specific attributes of the model, such as feature integration, attention mechanisms, and post-processing for screening model prediction results.

6.3.5 YOLOv5

YOLOv5 [29] represents a significant advancement in object detection technology, It grows on what came before while introducing several new features and improvements to enhance performance and usability.

1. Model Architecture: YOLOv5 maintains the foundational architecture of YOLO (You Only Look Once) models, employing a convolutional neural network (CNN) for object detection. However, it introduces refinements and optimizations aimed at enhancing both accuracy and efficiency.

2. Backbone Network: YOLOv5 embraces an innovative backbone network architecture, typically drawing inspiration from models like EfficientNet or CSPNet. These backbone networks offer superior feature extraction capabilities, empowering YOLOv5 to achieve heightened precision in object detection tasks.

3. Training Enhancements: YOLOv5 incorporates diverse training enhancements to bolster model convergence and performance. These include advanced techniques such as hyperparameter optimization, learning rate scheduling, and data augmentation strategies like CutMix and MixUp.

4. Experiment Tracking: YOLOv5 seamlessly integrates experiment tracking functionalities, enabling users to monitor and analyze training progress in real-time. This functionality proves invaluable for researchers and developers, facilitating a deeper understanding of model behavior and informed decision-making throughout the training process.

5. Automatic Export: YOLOv5 streamlines the deployment process by automatically exporting trained models to popular export formats, such as TensorFlow SavedModel or ONNX.

6. Usability Improvements: YOLOv5 introduces several enhancements geared towards improving user experience, including a user-friendly command-line interface (CLI) and comprehensive documentation.

Figure 6.5 illustrates the intricate design and network components of YOLOv5.

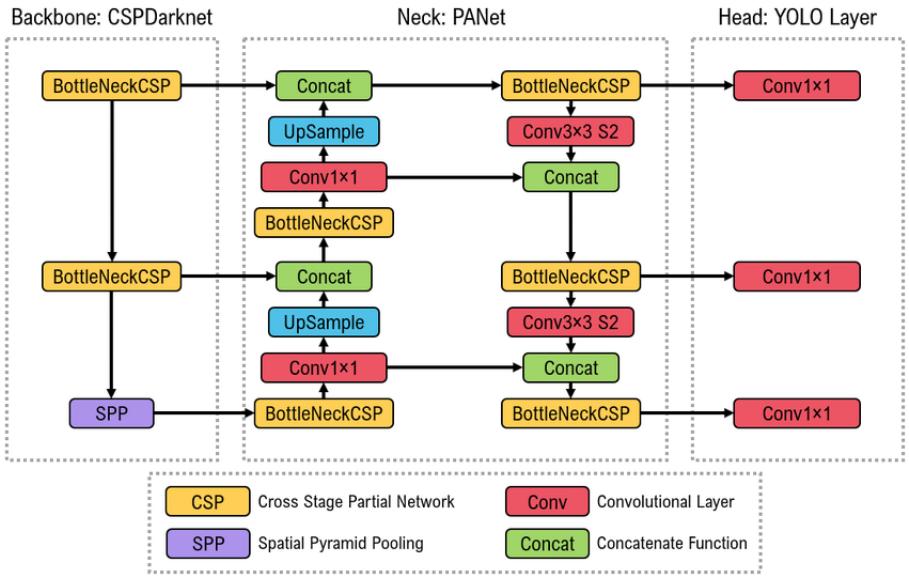
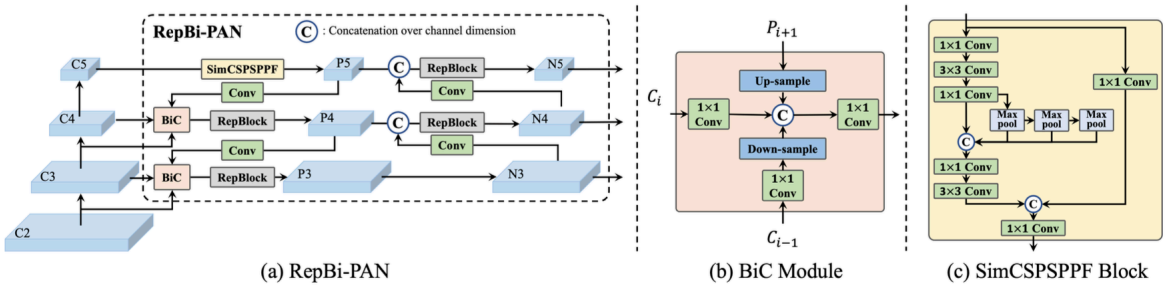


Figure 6.5: Architecture of the YOLOv5 model [24]

6.3.6 YOLOv6

YOLOv6 [21] introduces several enhancements to its architectural design and training methodology, including the implementation of a Bi-directional Concatenation (BiC) module, an anchor-aided training (AAT) strategy, and an improved backbone , neck design and Self-Distillation.



(a) RepBi-PAN (b) BiC Module (c) SimCSPSPPF Block

Figure 6.6: Overview of YOLOv6:(a) The neck of YOLOv6. (b) The structure of a BiC module. (c) A SimCSPSPPF block [21].

1. Bidirectional Concatenation (BiC) Module: The BiC module is introduced to provide more accurate localization signals in the neck of the YOLOv6 detector. It is applied to the top-down pathway of the Feature Pyramid Network (FPN) to improve performance without significant loss of efficiency. Also, The BiC module significantly boosts the performance of small object detection, particularly for YOLOv6-S and YOLOv6-L, improving the detection performance on small objects by 1.8%.

2. Enhanced Backbone and Neck Design: The YOLOv6 introduces an extended backbone and neck design, reinforcing it to achieve state-of-the-art performance on the COCO dataset at a high-resolution input. Also, feature integration at multiple scales is proven to be a critical and effective component of object detection, and the Feature Pyramid Network (FPN) is used to aggregate high-level semantic features and low-level features via a top-down pathway (*c.f* Figure 6.6).

3. Anchor-Aided Training (AAT) Strategy: The AAT strategy combines the advantages of both anchor-based and anchor-free paradigms. It introduces anchor-based auxiliary branches in the classification and regression head during training, which are removed at inference to boost accuracy performance without decreasing speed.

4. Self-Distillation: YOLOv6 introduces self-distillation to boost the performance of small models, where the heavier branch for DFL is taken as an enhanced auxiliary regression branch during training and is removed at inference to avoid marked speed decline.

6.3.7 YOLOv7

YOLOv7 [45] introduces an architecture of real-time object detectors and model scaling method. It includes model re-parameterization, and dynamic label assignment.

1. Model Re-parameterization: Model re-parameterization techniques merge multiple computational modules into one at the inference stage. This technique can be divided into module-level ensemble and model-level ensemble. At the module level, re-parameterization splits a module into multiple identical or different module branches during training and

integrates them into a completely equivalent module during inference. These new re-parameterization modules have been developed and designed for various architectures, enhancing their efficiency and performance.

2. Dynamic Label Assignment: Dynamic label assignment technology introduces new issues in the training of models with multiple output layers, particularly in assigning dynamic targets for the outputs of different branches. To tackle this issue, a new label assignment method called coarse-to-fine lead guided label assignment has been proposed to address this problem, aiming to provide effective solutions for handling dynamic label assignments across different branches.

3. Model Scaling: Model scaling involves scaling up or down an already designed model to fit different computing devices using factors such as resolution, depth, width, and stage. These Compound scaling methods have been proposed to maintain the properties of the model and its optimal structure during scaling.

6.3.8 YOLOv8

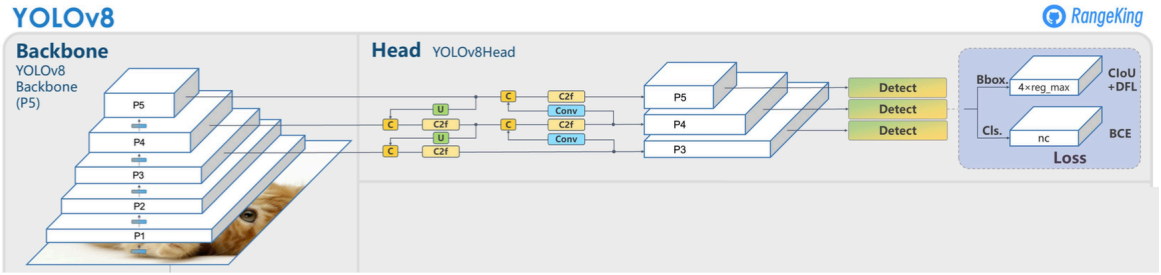


Figure 6.7: YOLOv8 Architecture [37].

YOLOv8 [37] builds on the success of previous versions, introducing new features and improvements for enhanced performance, flexibility, and efficiency. YOLOv8 supports various vision AI tasks, including detection, segmentation, pose estimation, tracking, and classification. This flexibility lets users apply YOLOv8's features in different areas and fields. YOLOv8 introduces numerous improvements compared to the earlier versions of YOLO such as a new neural network architecture (*c.f* Figure 6.7) that utilizes both Feature Pyramid Network (FPN) and Path Aggregation Network (PAN) and a new labeling tool called RoboFlow Annotate that simplifies the annotation process.

1. Feature Pyramid Network (FPN): The FPN operates by gradually reducing the spatial resolution of the input image while increasing the number of feature channels. This technique generates feature maps that are capable of detecting objects at different scales and resolutions.

2. Path Aggregation Network (PAN): The PAN architecture aggregates features from different levels of the network through skip connections. This enables the network to effectively capture features at multiple scales and resolutions, which is crucial for accurately detecting objects of different sizes and shapes.

3. RoboFlow Annotate: RoboFlow Annotate is used for image annotation and object detection tasks in computer vision. It makes it easier to annotate images for training the model and includes several features such as auto labeling, labeling shortcuts, and customizable hotkeys.

Figure 6.8 demonstrates the effectiveness of YOLOv8 in object detection and its potential for further improvements in the future.

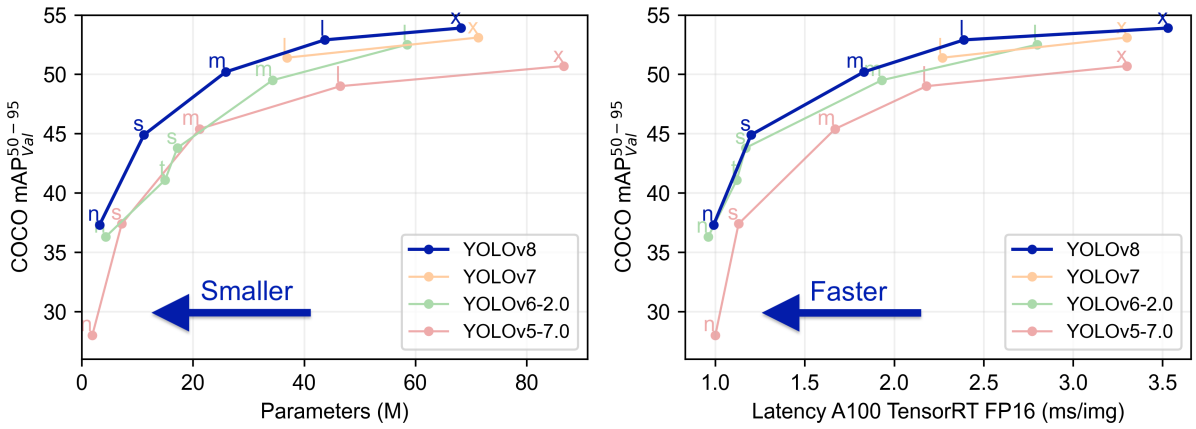


Figure 6.8: Comparison YOLOv8 with other YOLO versions [37].

Table 6.4 presents the performance metrics for different pre-trained models of YOLOv8. The table highlights the balance between model complexity, speed, and accuracy, providing insights into the trade-offs for various applications.

Model	Size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	Params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Table 6.4: Performance metrics for YOLOv8 models [44]

6.4 Applications in Computer Vision

Now we will see some real examples of YOLO systems applied in different sectors.

- **Agriculture:** In agriculture, YOLO-based systems are applied for monitoring crop health, detecting pests, and assessing plant growth. These systems provide farmers with valuable insights, helping them to optimize crop yield and reduce losses. As demonstrated in [25], YOLO is employed to detect pests such as true bugs in hazelnut orchards.The system achieves high precision, aiding in early pest detection.
- **Medical Imaging:** YOLO algorithms have significant applications in the healthcare sector, particularly in medical imaging. They assist in the detection and diagnosis of various diseases, including cancer, from medical images. According to [31], YOLOv8 is applied to medical images such as blood smear images for acute lymphoblastic leukemia, Pap smear images for cervical cancer, histopathological images for lung and colon cancer, and dermatoscopic images for skin cancer. By detecting and delineating tumor boundaries, YOLOv8 assists radiologists in identifying malignant areas more accurately.
- **Manufacturing:** YOLO algorithms also find applications in the manufacturing industry for quality control and anomaly detection. According to [41], YOLOv4 is used to detect three critical anomalies in bottles: dents on the bottle surface, skewed caps, and non-straight bottles. By leveraging computer vision and machine learning techniques, the system efficiently identifies defects in real-time, ensuring high-quality production.

6.5 Conclusion

In this chapter, we explored the evolution and significance of the YOLO algorithm in object detection. We examined its different versions, from the original YOLO to the latest YOLOv8, highlighting the architectural advancements and improvements in accuracy and speed. The unified detection approach, combined with continuous enhancements, has made YOLO a standard in the field of computer vision, enabling real-time applications across various domains such as agriculture, medical imaging, and manufacturing.

Chapter 7

Dataset

7.1 Introduction

In this chapter, we provide a detailed overview of the KITTI dataset, which is fundamental to our research in autonomous driving object detection. We will discuss the dataset's collection process, its structure, and the specific preparations required to utilize it effectively for training our model.

7.2 KITTI

The KITTI dataset, introduced by Geiger et al. in 2012, serves as a benchmark for object detection in autonomous driving research [13]. It has been recorded from a moving platform (*c.f* Figure 7.1) while driving in and around Karlsruhe, Germany by the Karlsruhe Institute of Technology (KIT) and the Toyota Technological Institute of Chicago (TTI-C) and it consists of 7,481 training images and 7,518 test images.

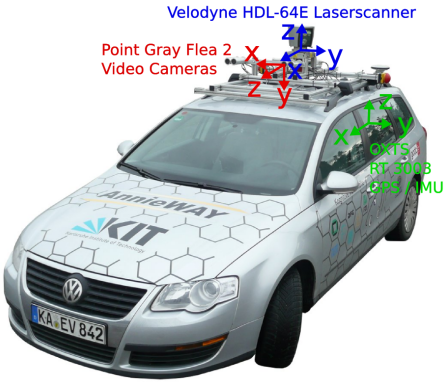


Figure 7.1: Recording platform: Volkswagen Passat B6 is equipped with four video cameras (two color and two grayscale cameras), a rotating 3D laser scanner and a combined GPS/IMU inertial navigation system [12].

We used the 2D object detection in KITTI dataset consisting of 7481 training images, labeled,

and 7518 test images, unlabeled, comprising a total of 80.256 labeled objects. All images are color and saved as png.

The dataset contains 9 labels, as shown in Figure 7.2: Car, Van, Truck, Pedestrian, Person_sitting, Cyclist, Tram, Misc (Miscellaneous): this label is used for miscellaneous objects that do not fit neatly into other categories, and Dontcare: this label is used to indicate regions of the image where objects are present but are not annotated.

Figure 7.3 shows a representative samples from the dataset.

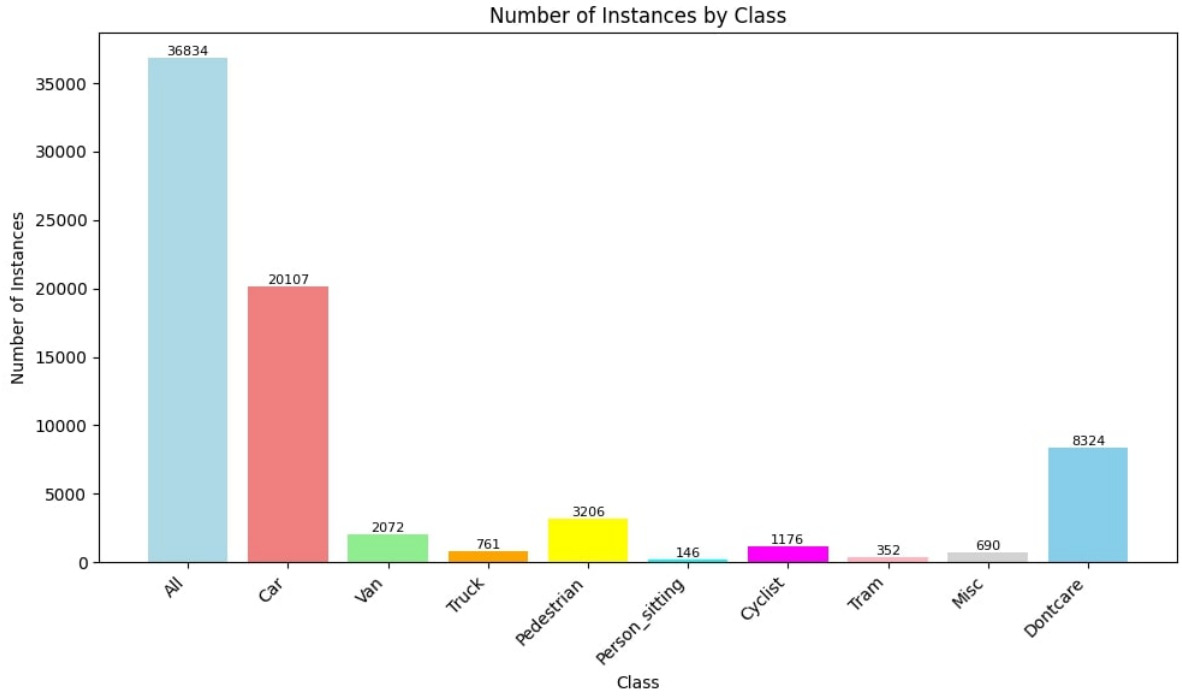


Figure 7.2: Training labels

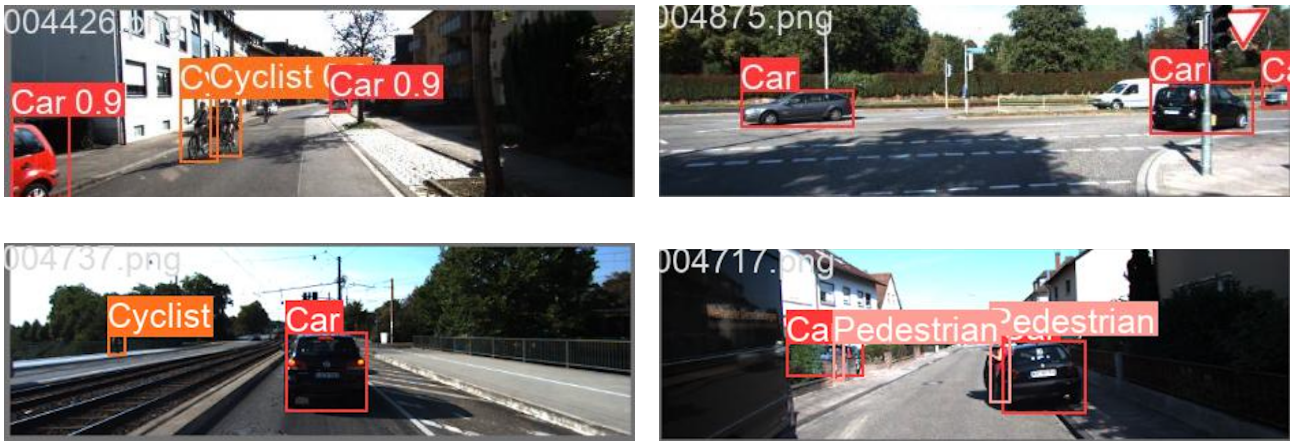


Figure 7.3: Representative samples from the dataset

7.3 Dataset Preparation

Due to the unavailability of ground truth annotations for the test set, our work will focus on the 7,481 training images. Following the train/test split suggested by Chen et al. [2], we divided the KITTI training dataset into 3,712 training images and 3,769 testing images. Our analysis primarily focused on the labels: Car, Pedestrian, and Cyclist.
YOLO requires the directory structure to include separate train, validation, and test data. We organized the data to be like this:

```
kitti
├── train
│   ├── images
│   └── labels
├── test
│   ├── images
│   └── labels
└── data.yaml
```

Initially, the labels were structured as follows:

- (*object_type*): Specifies the type of annotated object.
- (*truncation*): Represents the visible portion of the object as a float between 0.0 and 1.0, where 0.0 indicates full visibility and 1.0 signifies that the object is entirely outside the image frame.
- (*occlusion*): Indicates the level of occlusion with an integer value, where 0 means the object is fully visible and higher values indicate increased occlusion.
- (*alpha*): Measures the observation angle of the object in radians relative to the camera, specifically the angle between the object's heading direction and the camera's positive x-axis.
- $(left)$, (*top*), (*right*), (*bottom*): These are the 2D bounding box coordinates in the image, specifying the pixel positions of the top-left and bottom-right corners of the bounding box.
- (*height*), (*width*), (*length*): Provide the object's 3D dimensions (height, width, and length) in meters.
- (x), (y), (z): The 3D coordinates of the object's center within the camera coordinate system, measured in meters.
- (*rotation_y*): Describes the object's rotation around the y-axis in the camera coordinate system, expressed in radians.

However, for training a YOLOv8 model, only (*object_type*), $(left)$, (*top*), (*right*), and (*bottom*) were necessary. So, we removed all the rest of the labels. Since (*object_type*) is represented as

a string, we proceeded to map all labels, converting (*object_type*) into corresponding numerical identifiers. The mapping is as follows:

```
{
    "0": "Car",
    "1": "Pedestrian",
    "2": "Cyclist",
}
```

Then, we then normalized the coordinates ((*left*)(*top*)(*right*)(*bottom*)) using the equations:

$$
\begin{aligned}
x_{\text{center}} &= \frac{\text{left} + \text{right}}{2} \\
y_{\text{center}} &= \frac{\text{top} + \text{bottom}}{2} \\
\text{width} &= \text{right} - \text{left} \\
\text{height} &= \text{bottom} - \text{top} \\
x_{\text{center\_normalized}} &= \frac{x_{\text{center}}}{\text{image\_width}} \\
y_{\text{center\_normalized}} &= \frac{y_{\text{center}}}{\text{image\_height}} \\
\text{width\_normalized} &= \frac{\text{width}}{\text{image\_width}} \\
\text{height\_normalized} &= \frac{\text{height}}{\text{image\_height}}
\end{aligned}
$$

The `data.yaml` file contains the following configuration:

- **train:** ../train/images (path to the directory containing training images)
- **val:** (path to the directory containing validation images)
- **test:** ../test/images (path to the directory containing testing images)
- **nc:** 3 (number of classes)
- **names:** [”Car”, ”Pedestrian”, ”Cyclist”] (Class Labels)

7.4 Conclusion

In this chapter, we have provided an overview of the KITTI dataset, detailing its components, structure, and the specific preparations required for training our model. By understanding the dataset's intricacies and how to properly format and utilize it, we lay the groundwork for effective model training and evaluation. In the next chapter, we will delve into the experimentation and evaluation processes to assess the performance of our trained model.

Chapter 8

Experiments and Evaluation

8.1 Introduction

This chapter details the experimental setup and evaluation metrics used to assess the performance of our object detection models. We begin by outlining the configuration and hyperparameters. Subsequently, we introduce the metrics utilized to evaluate model performance. Additionally, we explore the Dynamic Scene Composition Technique (DSCT). The chapter concludes with a comprehensive analysis of the results, comparing our models with state-of-the-art alternatives to highlight their effectiveness and potential for real-world applications.

8.2 Experiment

To get the best model, we used the YOLOv8 pre-trained model, which is trained on the COCO dataset and includes 80 pre-defined classes: yolov8x.pt.
The model was initialized with the following configuration and hyperparameters:

- **Batch Size (batch):** This determines the number of images the algorithm processes within each sub-cycle into which an epoch is divided. During the experimentation, the model was executed with batch value of 32.

- **Learning Rate (lr0|lrf):** This is the step value in which weights are updated in each iteration to minimize the loss function. In this case, we adjusted the initial learning rate and final learning rate with a value of 0.0001.

- **Epochs (epochs):** An epoch is a complete cycle of processing all the data grouped in batches. Therefore, the number of epochs set for training equals the number of times each image is processed. During training, the epoch value was 100 epochs and we saved the model for each 10 epochs.

- **Image size (imgz):** the desired image size for training. Before being input into the model, all images are adjusted to this specified dimension. we tested on image size 640.

- **Device (device):** Indicates the hardware utilized for the training process. In our work, we used NVIDIA A40 GPU.
- **Data source (data):** Specifies the location of the dataset configuration file (for example, data.yaml). We used "kitti/data.yaml".

8.3 Metrics

To evaluate our model, we used several metrics, including:

- **Confusion matrix:** A confusion matrix is a useful tool for assessing the performance of a machine learning model, typically presented in a tabular format. It enables data scientists to gain detailed insights into their model's accuracy, including areas where it may be making errors or displaying weaknesses. By analyzing the confusion matrix, practitioners can make informed adjustments and enhancements to improve the model's predictive capabilities [6].

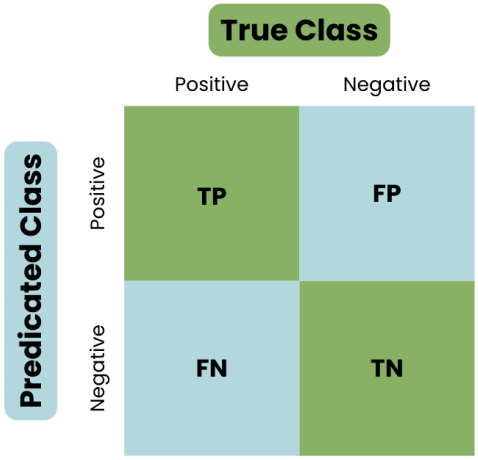


Figure 8.1: Example of confusion matrix

The confusion matrix contains:

  - **True Positive (TP)**: The model accurately predicted the positive class. For example, correctly identifying a Car as a Car.
  - **True Negative (TN)**: The model accurately predicted the negative class. For example, correctly identifying a Cyclist as not a Car.
  - **False Positive (FP)**: The model incorrectly predicted the positive class. For example, misidentifying a Cyclist as a Car.
  - **False Negative (FN)**: The model incorrectly predicted the negative class. For example, misidentifying a Car as a Cyclist.

- **AP** (Average Precision): Average precision (AP) is the area under the Precision-Recall curve. Precision measures the ratio of true positive predictions to the total number of positive predictions made by the model. It is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP):

$$P = \frac{TP}{TP + FP}$$

Recall quantifies the ability of the model to correctly identify all relevant instances. It is calculated as the ratio of true positives to the sum of true positives and false negatives (FN):

$$R = \frac{TP}{TP + FN}$$

AP is a metric used to evaluate the model's performance across different Intersection over Union (IoU) thresholds. Intersection over Union (IoU) measures the overlap between predicted and ground truth bounding boxes and is calculated as:

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

To test our models we used the object development kit provided by KITTI [13]. KITTI assesses detection for each class in three categories: easy, moderate, and hard, which are defined based on the levels of occlusion and truncation of objects [13].

- **Easy:** Minimum bounding box height of 40 pixels, with maximum of fully visible objects and a maximum truncation of 15%.
- **Moderate:** Minimum bounding box height of 25 pixels, with maximum of partially occluded objects and a maximum truncation of 30%.
- **Hard:** Minimum bounding box height of 25 pixels, with maximum difficult to see objects, and a maximum truncation of 50%.

In accordance with the standard KITTI setup, a Ground Truth (GT) instance is considered recalled when the Intersection over Union (IoU) exceeds 70% for cars, and 50% for pedestrians and cyclists.

8.4 Dynamic Scene Composition Technique

The Dynamic Scene Composition Technique (DSCT) involves merging four distinct images into one composite image, thereby creating a diverse and complex input for the model. This method introduces spatial and contextual variations, compelling the model to learn more robust features. By combining different images, DSCT simulates a variety of real-world scenarios within a single training example. The step-by-step process of DSCT includes several key stages:

Firstly, four images are randomly selected from the training dataset. These images are then combined into a single composite image, ensuring diverse object placements and backgrounds. Next, the bounding boxes and annotations are modified to accurately represent objects within the composite image. Finally, the composite image undergoes additional augmentation processes such as scaling, rotation, and color adjustments. This comprehensive approach enhances the model's ability to generalize across various scenarios, leading to improved performance in object detection tasks.

The DSCT is applied with a range of numbers from 0.0 to 1.0, indicating the probability of applying the augmentation. A probability of 1.0 means that the augmentation is applied to every image, while a lower probability, such as 0.1, indicates that for each image processed, there is a 10% chance that the mosaic augmentation will be applied and a 90% chance that it will not be applied.

Example of the DSCT [28]: The initial four images are shown in Figure 8.2.

Figure 8.2: Four images to be combined using DSCT

1. **Resize Each Image**: Each image is resized to 256x256 pixels, with bounding boxes adjusted accordingly (*c.f* Figure 8.3).

Figure 8.3: Resizing images

2. **Create Composite Image**: A new 512x512 image is created, and the resized images are placed in the four corners. The bounding boxes may initially be misaligned (*c.f* Figure 8.4).

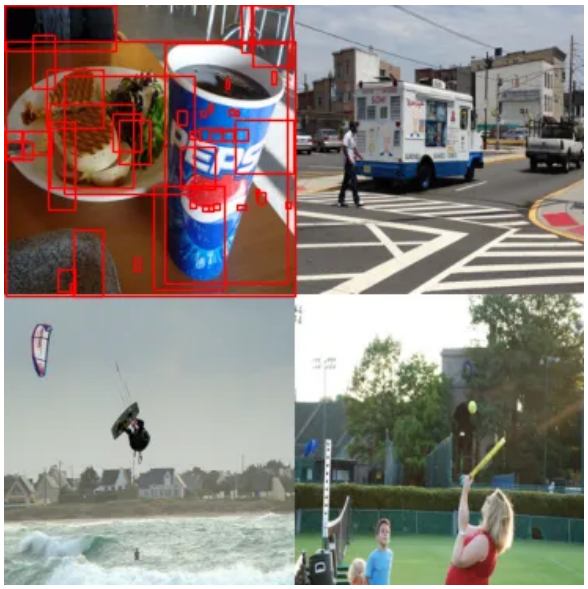


Figure 8.4: Creating the composite image

3. **Move Bounding Boxes**: Bounding boxes are repositioned as follows:

   - Top-left image boxes remain unchanged.
   - Top-right image boxes are moved 256 pixels to the right.
   - Bottom-left image boxes are moved 256 pixels down.
   - Bottom-right image boxes are moved 256 pixels to the right and 256 pixels down (*c.f* Figure 8.5).

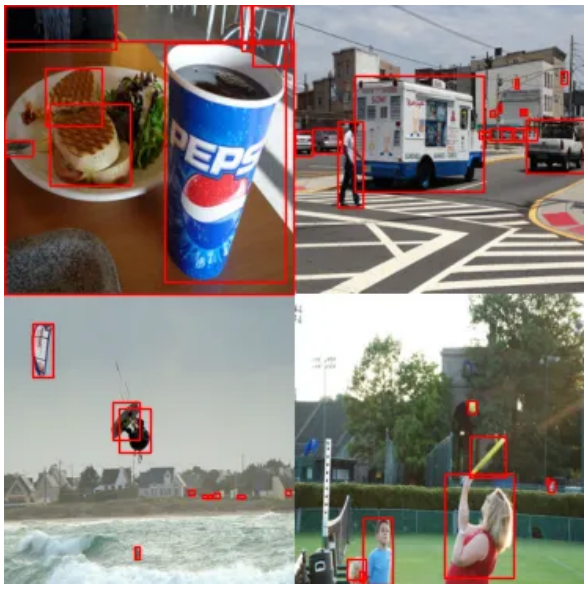


Figure 8.5: Adjusting bounding boxes

4. **Take Random Cutout**: A random 256x256 pixel section is selected from the composite image. This cutout is chosen to include parts of all four images to ensure diversity (*c.f* Figure 8.6).

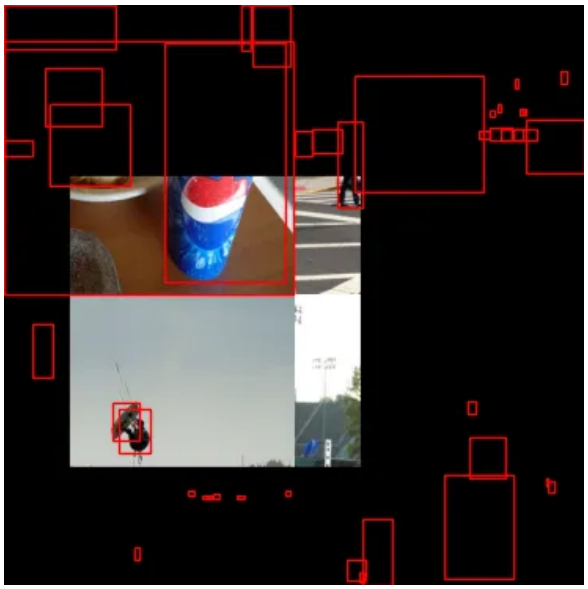


Figure 8.6: Taking a random cutout

5. **Remove Out-of-Bounds Boxes**: Bounding boxes outside the cutout are removed, while those partially inside are retained (*c.f* Figure 8.7).

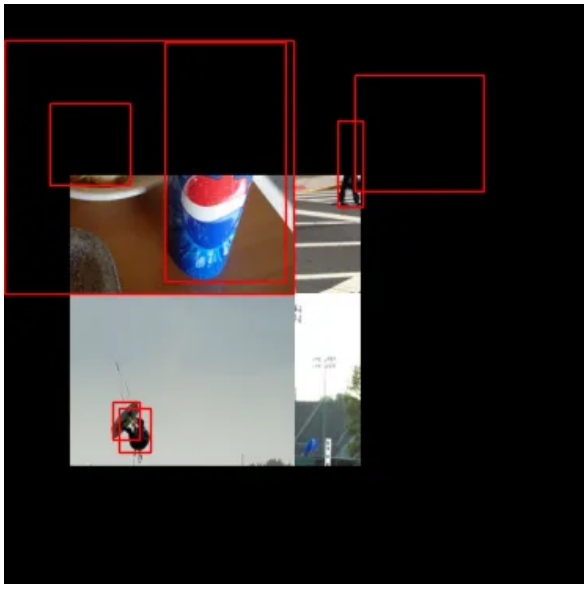


Figure 8.7: Removing out-of-bounds boxes

6. **Resize Remaining Boxes**: Remaining bounding boxes are resized to fit within the cutout, ensuring they are correctly proportioned (*c.f* Figure 8.8).

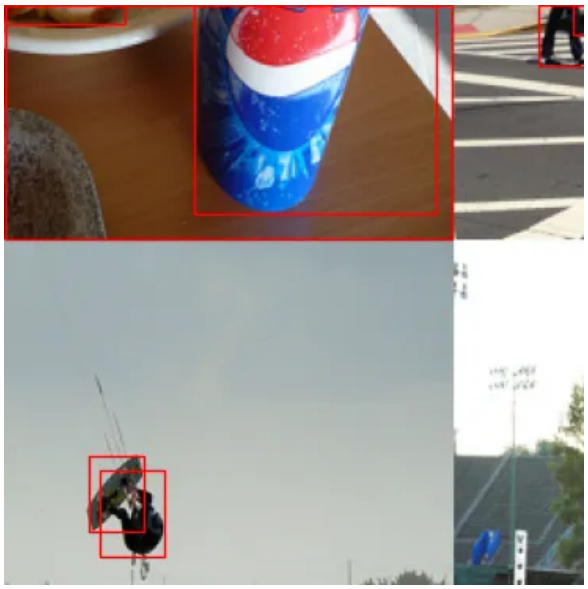


Figure 8.8: Final composite image with object detections

This method preserves the integrity of the bounding boxes and ensures the model is exposed to diverse features within a single image. The final composite image demonstrates the effectiveness of DSCT, maintaining even the smallest features, such as a sandwich in the top-left corner, which helps the model generalize better (*c.f* Figure 8.8).

8.5 Results

To evaluate the performance of our proposed method for object detection accuracy from images, we conducted two series of experiments. The first series aimed to assess the impact of varying the Dynamic Scene Composition Technique (DSCT) parameter values across different object categories. The second series compared the performance of our proposed method against several state-of-the-art models. The experiments were designed to ensure a rigorous evaluation, adhering to protocols established in leading research works. We set a confidence threshold of 0.15 for these experiments.

8.5.1 Initial series of experiments

In the initial series of experiments, we aimed to evaluate the performance impact of varying DSCT parameter values across different object categories. The methodology involved adjusting the DSCT values and measuring the model's performance in detecting cars, pedestrians, and cyclists under varying conditions.

DSCT	Car			Pedestrian			Cyclist			Overall		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
0.0	92.42	85.32	77.9	56.14	47.41	40.39	63.0	37.64	35.28	70.52	56.12	51.19
0.1	93.79	86.01	78.8	58.56	51.79	46.68	60.57	41.79	40.48	70.97	59.86	55.32
0.2	**94.36**	86.58	79.35	59.55	52.34	45.32	62.38	39.59	37.54	72.10	59.50	54.74
0.3	91.37	85.56	80.82	**64.02**	54.81	47.75	65.66	42.08	40.29	73.02	60.82	56.95
0.4	91.77	85.32	80.38	59.57	52.06	45.1	63.39	40.79	38.56	71.58	59.39	54.68
0.5	93.43	84.96	77.44	55.25	48.44	43.49	63.13	42.83	40.81	70.60	58.08	53.25
0.6	91.44	85.69	80.78	62.69	53.89	46.96	64.51	42.29	40.45	72.21	60.62	56.06
0.7	93.8	**88.52**	**81.38**	62.24	54.5	47.6	65.5	43.51	41.95	**73.85**	**62.18**	**56.98**
0.8	92.99	86.05	78.76	59.62	50.84	45.8	61.83	42.27	40.72	71.48	59.72	55.09
0.9	91.84	85.37	80.52	59.44	51.93	45.16	63.25	**44.95**	**42.88**	71.51	60.75	56.19
1.0	92.0	85.54	80.62	61.66	**52.9**	**47.97**	**66.04**	42.47	40.68	73.23	60.97	56.42

Table 8.1: Performance comparison across different DSCT numbers and categories

The results, shown in Table 8.1, highlights the effectiveness of our models across various DSCT values and object categories. Notably, DSCT 0.7 consistently delivers the best overall performance, demonstrating its importance and dominance:

- **Car Detection:** DSCT 0.7 achieves the highest performance in the Moderate (88.52%) and Hard (81.38%) categories, while maintaining competitive performance in the Easy category (93.8%). This showcases the robustness of DSCT 0.7 in handling car detection under various conditions.

- **Pedestrian Detection:** Although DSCT 0.3 performs best in the Easy category (64.02%), DSCT 0.7 delivers strong results across Moderate (54.5%) and Hard (47.6%) categories. This indicates DSCT 0.7's capability to effectively detect pedestrians in more challenging scenarios.

- **Cyclist Detection:** DSCT 0.7 shows excellent performance in the Moderate (43.51%) and Hard (41.95%) categories, while being competitive in the Easy category (65.5%). This highlights the versatility and effectiveness of DSCT 0.7 in cyclist detection tasks.

- **Overall Performance:** DSCT 0.7 stands out with the highest overall performance across all difficulty levels, achieving 73.85% (Easy), 62.18% (Moderate), and 56.98% (Hard). This underscores the dominance of DSCT 0.7, making it the most effective setting for robust and reliable object detection across diverse categories and conditions.

These results underscore the strong performance of DSCT 0.7 across different categories and difficulty levels. The comparison demonstrates the robustness and competitiveness of DSCT 0.7, highlighting its advanced capabilities and potential for further enhancement to achieve even higher accuracy and reliability in diverse and challenging environments.

8.5.2 Second series of experiments

To demonstrate the effectiveness of our model, we compare its performance with several advanced models including Regionlets, DPM-VOC+VP, Mono3D, LSVM-MDPM-sv, and Complexer-YOLO. The performance metrics are compared across three categories (Car, Pedestrian, Cyclist) and three difficulty levels (Easy, Moderate, Hard):

Method	Car			Pedestrian			Cyclist		
	Easy	Mod	Hard	Easy	Mod	Hard	Easy	Mod	Hard
Mono3D[3]	92.33	**88.66**	78.96	**80.35**	**66.68**	**63.44**	**76.04**	**66.36**	**58.87**
DPM-VOC+VP[32]	74.95	64.71	48.76	59.48	44.86	40.37	42.43	31.08	28.23
Regionlets[26]	84.75	76.45	59.70	73.14	61.15	55.21	70.41	58.72	51.83
LSVM-MDPM-sv[9, 10]	68.02	56.48	44.18	47.74	39.36	35.95	35.04	27.50	26.21
Complexer-YOLO[39]	55.63	49.44	44.13	19.45	15.32	14.80	28.36	23.48	15.60
Proposed Method (DSCT 0.7)	**93.8**	88.52	**81.38**	62.24	54.5	47.6	65.5	43.51	41.95

Table 8.2: Performance comparison across different methods and categories

The comparison, shown in Table 8.2, highlights the effectiveness of our proposed method (DSCT 0.7) relative to several advanced models:

- **Car Detection:** Our method achieves the highest performance in the Easy (93.8%) and Hard (81.38%) categories, and shows competitive performance in the Moderate (88.52%) category. This demonstrates the robustness of our method in handling car detection under various conditions.
- **Pedestrian Detection:** Our method delivers strong results across Moderate (54.5%) and Hard (47.6%) categories, while being competitive in the Easy category (62.24%). This indicates our method's capability to effectively detect pedestrians in more challenging scenarios.
- **Cyclist Detection:** Our method shows excellent performance in the Moderate (43.51%) and Hard (41.95%) categories, while being competitive in the Easy category (65.5%). This highlights the versatility and effectiveness of our method in cyclist detection tasks.

These results underscore the strong performance and dominance of our proposed method across different categories and difficulty levels. The comparison demonstrates the robustness and competitiveness of our approach, highlighting its advanced capabilities and potential for further enhancement to achieve even higher accuracy and reliability in diverse and challenging environments.

8.6 Conclusion

In this chapter, we have presented the detailed methodology and results of our experiments using the YOLOv8 model for object detection on the KITTI dataset. By leveraging advanced

techniques such as the Dynamic Scene Composition Technique (DSCT), we have demonstrated improvements in model performance, particularly in challenging scenarios. Our evaluation, which included metrics like the confusion matrix and Average Precision (AP), highlighted the robustness and accuracy of our models. Furthermore, the comparative analysis with state-of-the-art models underscored the competitiveness of our approach. These findings validate the effectiveness of our methods and provide a solid foundation for further enhancements and applications in autonomous driving and other computer vision tasks.

Chapter 9

Conclusions

9.1 Contributions

This final project has made significant contributions to the field of object detection for autonomous vehicles through the exploration and application of the YOLOv8 algorithm using the KITTI dataset. A thorough review of existing object detection methods and technologies was conducted, highlighting the evolution and advancements in the field, especially in the context of autonomous vehicles. Additionally, a novel approach called the Dynamic Scene Composition Technique (DSCT) was introduced to enhance the robustness of the detection model. This technique involves merging multiple images to create a diverse training dataset, which helps the model learn to handle a variety of real-world scenarios.
The results of the proposed YOLOv8 model demonstrated significant improvements in certain categories and conditions, particularly with the integration of DSCT. The performance of the YOLOv8 model was also benchmarked against other advanced models, showing competitive results and highlighting the effectiveness of the proposed methods. Moreover, the preparation and preprocessing of the KITTI dataset were meticulously documented, providing a clear framework for future research and applications in this domain.

9.2 Future Work

While this final project has achieved substantial progress, there are several avenues for future research and improvement. Future work could involve integrating data from additional sensors, such as LiDAR and radar, to enhance the accuracy and reliability of the object detection model. Additionally, utilizing more diverse and extensive datasets beyond KITTI can help in training a more robust model capable of handling various driving environments and conditions.
The techniques and methodologies developed in this thesis can be extended to other fields such as robotics, surveillance, and smart city infrastructure, where object detection plays a critical role. By building on the findings and contributions of this research, future studies can further advance the capabilities of object detection systems and their applications in various domains.

In conclusion, this thesis has laid a solid foundation for the application of advanced object detection algorithms in autonomous vehicles. The proposed methods and findings not only contribute to the academic knowledge base but also have practical implications for enhancing the safety and efficiency of autonomous driving technologies. Future research building on this work has the potential to make significant strides in the field of computer vision and autonomous systems.

Appendices

Appendix A

Code repository

In this appendix, we describe the code repository associated with the project and provide instructions on how to use it to replicate the experiments.

Directory Structure

- `DatasetPreparation`: Contains scripts for preparing datasets.
  - `keep_ness.py`
  - `map.py`
  - `normalization.py`
- `devkit_object-3`: Contains scripts and data for evaluation and mapping.
- `train.py`: Script for training the YOLO model.
- `test.py`: Script for testing the YOLO model.
- `data.yaml`: Configuration file for the dataset.
- `trainlist.txt`: List of training data files.
- `testlist.txt`: List of testing data files.
- `reverse.py`: Script to prepare and convert normalized labels to actual coordinates for testing.
- `keep.py`: Script to test the labels.

Getting Started

Prerequisites

- Python 3.10

- Install `ultralytics`:

```
pip install ultralytics
```

- Install `pillow`:

```
pip install pillow==10.3.0
```

Usage

Dataset Preparation

Prepare your dataset using the scripts in the `DatasetPreparation` directory. For example:

```
python DatasetPreparation/keep_ness.py
```

Training

Train the YOLO model using the `train.py` script:

```
python train.py
```

Testing

Generate the test results of the model using the `test.py` script:

```
python test.py
```

Model Evaluation

To evaluate the model, use the evaluation kit. You need to reverse the normalization and change the labels first. For missing values (occlusion, 3D coordinates, etc.), insert the ignored values (e.g., -1, -10).
To reverse the normalization and fill in the missing data use:

```
python reverse.py
```

To evaluate the model use:

```
python keep.py
```

Check this link for more information on testing using the KITTI evaluation kit.
GitHub link: LINK

Conclusion

By following the steps outlined above, you should be able to replicate the experiments conducted in this project. For any issues or questions, refer to the `README.md` file in the repository for additional information.

Bibliography

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: (2020). URL: https://arxiv.org/pdf/2004.10934.pdf.

[2] Xiaozhi Chen et al. "3D Object Proposals for Accurate Object Class Detection". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015, pp. 424–432. URL: https://www.cvlibs.net/projects/autonomous_vision_survey/literature/Chen2015NIPS.pdf.

[3] Xiaozhi Chen et al. "Monocular 3D Object Detection for Autonomous Driving". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2147–2156. DOI: 10.1109/CVPR.2016.236.

[4] R. Y. Choi et al. "Introduction to Machine Learning, Neural Networks, and Deep Learning". In: *Translational Vision Science & Technology* 9.2 (Feb. 2020), p. 14. DOI: 10.1167/tvst.9.2.14.

[5] Musab Coşkun et al. "An Overview of Popular Deep Learning Methods". In: *European Journal of Technique* 7.2 (2017), pp. 165–176. URL: https://dergipark.org.tr/en/download/article-file/437659.

[6] DataCamp. *What is a Confusion Matrix in Machine Learning?* Nov. 2023. URL: https://www.datacamp.com/tutorial/what-is-a-confusion-matrix-in-machine-learning.

[7] P. Deepa and Rashmita Khilar. "A Report on Voice Recognition System: Techniques, Methodologies and Challenges using Deep Neural Network". In: (2021), pp. 1–5. DOI: 10.1109/i-PACT52855.2021.9697005.

[8] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: (2018). arXiv: 1603.07285 [stat.ML]. URL: https://arxiv.org/abs/1603.07285v2.

[9] Kai Fan et al. "Fast Second Order Stochastic Backpropagation for Variational Inference". In: *Advances in Neural Information Processing Systems 28 (NIPS 2015)*. 2015. URL: https://proceedings.neurips.cc/paper/2015/hash/fc3cf452d3da8402bebb765225ce8c0e-Abstract.html.

[10] Pedro F. Felzenszwalb et al. "Object Detection with Discriminatively Trained Part-Based Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645. DOI: 10.1109/TPAMI.2009.167.

[11] Kunihiko Fukushima. "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202. URL: https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf.

[12] A Geiger et al. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* (2013). DOI: 10.1177/0278364913491297. URL: https://doi.org/10.1177%2F0278364913491297.

[13] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[14] R. Girshick. "Fast R-CNN". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169. URL: https://ieeexplore.ieee.org/document/7410526.

[15] R. Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587.

[16] Meng-Hao Guo et al. "Attention Mechanisms in Computer Vision: A Survey". In: *Journal of LaTeX Class Files* 14.8 (Nov. 2021), pp. 1–20. URL: https://arxiv.org/abs/2111.07624.

[17] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385v1* (2015).

[18] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*. 2015, pp. 770–778. URL: https://papers.nips.cc/paper_files/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[19] Chao Huang et al. "Abnormal Event Detection Using Deep Contrastive Learning for Intelligent Video Surveillance System". In: *IEEE Transactions on Industrial Informatics* 18.8 (2022), pp. 5171–5180. URL: https://doi.org/10.1109/TII.2021.3122801.

[20] IBM. *Qu'est-ce que la Computer Vision ?* URL: https://www.ibm.com/ca-fr/topics/computer-vision.

[21] Katsamenis et al. "YOLOv6 v3.0: A Full-Scale Reloading". In: (2023). URL: https://doi.org/10.48550/arXiv.2301.05586.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: 1 (2012), pp. 1097–1105. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[24] Chuyi Li et al. "TraCon: A novel dataset for real-time traffic cones detection using deep learning". In: (2023).

[25] Martina Lippi et al. "A YOLO-Based Pest Detection System for Precision Agriculture". In: *2021 29th Mediterranean Conference on Control and Automation (MED)*. 2021, pp. 342–347. DOI: 10.1109/MED51440.2021.9480344.

[26] C. Long et al. "Accurate object detection with location relaxation and regionlets relocalization". In: *Proceedings of the Asian Conference on Computer Vision (ACCV)* (2014).

[27] Paula Mielgo Martín. *Procesamiento automático de imágenes y vídeos con técnicas de Deep Learning*. Tutors: Anibal Bregón Bregón, Jorge Silvestre Vilches. July 2022.

[28] Gabriel Mongaras. *YOLOX Explanation: Mosaic and Mixup for Data Augmentation*. 2021. URL: https://gmongaras.medium.com/yolox-explanation-mosaic-and-mixup-for-data-augmentation-3839465a3adf (visited on 06/17/2024).

[29] Upesh Nepal and Hossein Eslamiat. "Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs". In: (2022). URL: https://doi.org/10.3390/s22020464.

[30] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. "A Survey of the Usages of Deep Learning for Natural Language Processing". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.2 (2021), pp. 604–624. DOI: 10.1109/TNNLS.2020.2979670.

[31] N. Palanivel et al. "The Art of YOLOv8 Algorithm in Cancer Diagnosis using Medical Imaging". In: *2023 International Conference on System, Computation, Automation and Networking (ICSCAN)*. 2023, pp. 1–6. DOI: 10.1109/ICSCAN58655.2023.10395046.

[32] Bojan Pepik et al. "Multi-View and 3D Deformable Part Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.11 (2015), pp. 2232–2245. DOI: 10.1109/TPAMI.2015.2408347.

[33] Đorđe Petrović, Radomir Mijailović, and Dalibor Pešić. "Traffic Accidents with Autonomous Vehicles: Type of Collisions, Manoeuvres and Errors of Conventional Vehicles' Drivers". In: *Transportation Research Procedia* 45 (2020), pp. 161–168. DOI: 10.1016/j.trpro.2020.03.003. URL: https://www.sciencedirect.com/science/article/pii/S2352146520301654.

[34] Samira Pouyanfar et al. "A Survey on Deep Learning: Algorithms, Techniques, and Applications". In: *ACM Comput. Surv.* 51.5 (Sept. 2018), 92:1–92:36. DOI: 10.1145/3234150.

[35] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: (2018). URL: https://arxiv.org/pdf/1804.02767.pdf.

[36] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: (2015). URL: https://arxiv.org/pdf/1506.02640.pdf.

[37] Dillon Reis et al. "Real-Time Flying Object Detection with YOLOv8". In: (2023). URL: https://arxiv.org/pdf/2305.09972.pdf.

[38] Pierre Sermanet et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *arXiv preprint arXiv:1312.6229* (2014). URL: https://arxiv.org/pdf/1312.6229v4.

[39] Martin Simon et al. "Complexer-YOLO: Real-Time 3D Object Detection and Tracking on Semantic Point Clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019. URL: https://cs.brown.edu/people/pfelzens/papers/lsvm-pami.pdf.

[40] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks For Large-Scale Image Recognition". In: *CoRR Computing Research Repository* abs/1409.1556 (2014).

[41] Sai Lahari Sreerama et al. "Anomaly Detection Model for Bottles in a Manufacturing Unit". In: *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. 2023, pp. 1488–1493. DOI: 10.1109/ICIMIA60377.2023.10426102.

[42] D.C. Swanson et al. "Proposed Terms and Definitions for Power Quality: IEEE P1159.3/D9". In: *IEEE Transactions on Power Delivery* 14.1 (1999), pp. 123–129. DOI: 10.1109/61.736681. URL: https://ieeexplore.ieee.org/document/726791.

[43] Christian Szegedy et al. "Going Deeper with Convolutions". In: (2015).

[44] Ultralytics. *Ultralytics GitHub Repository*. 2024. URL: https://github.com/ultralytics/ultralytics.

[45] 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: (2022). URL: https://arxiv.org/pdf/2207.02696.pdf.

[46] R. Yamashita et al. "Convolutional neural networks: an overview and application in radiology". In: *Insights into Imaging* 9 (2018), pp. 611–629. DOI: 10.1007/s13244-018-0639-9. URL: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9.

[47] Mokri Mohammed Zakaria. *Classification des images avec les réseaux de neurones*. Université de Tlemcen. 2017. URL: http://dspace.univ-tlemcen.dz/bitstream/112/12235/1/Classification-des-images-avec-les-reseaux-de-neurones.pdf.

[48] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: (2014), pp. 818–833.