RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Électronique

End-of-study Project Dissertation for Obtaining the State Engineer's Degree in Electronic

Submitted in partial fulfillment of the requirements for the State Engineer Degree in Electronic Engineering

---

Camera-Based Automated Detection System for Speed Bumps and Road Defects

---

OULD ZMIRLI Amina

Supervised by :	**Mr. LARBES Cherif**	Pr	ENP
	Mr. KOBZILI Elhaouari	MCB	ESTA

Publicly presented and defended on the (24/06/2024)

Jury members :

President :	Mr.ADNANE Mourad	Pr	ENP
Examiner :	Mr. TAGHI Mohamed	MAA	ENP

ENP 2024

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Électronique

End-of-study Project Dissertation for Obtaining the State Engineer's Degree in Electronic

Submitted in partial fulfillment of the requirements for the State Engineer Degree in Electronic Engineering

---

Camera-Based Automated Detection System for Speed Bumps and Road Defects

---

OULD ZMIRLI Amina

Supervised by :	**Mr. LARBES Cherif**	Pr	ENP
	Mr. KOBZILI Elhaouari	MCB	ESTA

Publicly presented and defended on the (24/06/2024)

Jury members :

President :	Mr.ADNANE Mourad	Pr	ENP
Examiner :	Mr. TAGHI Mohamed	MAA	ENP

ENP 2024

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Électronique

Mémoire de Projet de Fin d'études

Pour l'obtention du diplôme d'ingénieur d'état en Électronique

Système de Détection Automatisé Basé sur des Caméras pour Dos d'âne et Défauts de la Route

OULD ZMIRLI Amina

Supervisé par :	**Mr. LARBES Cherif**	Pr	ENP
	Mr. KOBZILI Elhaouari	MCB	ESTA

Soutenu publiquement le (24/06/2024)

Membres du jury :

President :	Mr.ADNANE Mourad	Pr	ENP
Examinateur :	Mr. TAGHI Mohamed	MAA	ENP

ENP 2024

الملخص

جودة البنية التحتية للطرق ضرورية لضمان التنقل بأمان، ولكن هناك مخاطر مثل عيوب الطريق والمطبات التي تشكل تهديدات للمركبات والسائقين. يمثل نظام الكشف الآلي المعتمد على الكاميرات مبادرة تكنولوجية متقدمة تهدف إلى تعزيز السلامة على الطرق من خلال تقنيات رؤية الكمبيوتر المتقدمة. الهدف الرئيسي من هذا المشروع هو تطوير خوارزمية قوية قادرة على تحديد مختلف التشوهات في الطرق بدقة من خلال تحليل صور الكاميرات.

كلمات مفتاحية: مطبات السرعة، عيوب الطريق، كشف الأجسام، رؤية الكمبيوتر، نظام الكشف الآلي، التعلم الآلي، YOLOv8، السلامة على الطرق، الشبكات العصبية الالتفافية، المعالجة في الوقت الحقيقي

Résumé

La qualité des infrastructures routières est cruciale pour garantir des déplacements en toute sécurité, mais des dangers tels que les défauts de la route et les dos-d'âne représentent des menaces pour les véhicules et les conducteurs. Le Système de Détection Automatisée Basé sur les Caméras représente une initiative technologique de pointe visant à renforcer la sécurité routière grâce à des techniques avancées de vision par ordinateur. L'objectif principal de ce projet est de développer un algorithme robuste capable d'identifier précisément diverses anomalies routières en analysant les images des caméras.

Mots clés: Dos d'âne, Défauts de la route, Détection d'objets, Vision par ordinateur, Système de détection automatisé, Apprentissage automatique, YOLOv8, Sécurité routière, Réseaux neuronaux convolutionnels, Traitement en temps réel

Abstract

The quality of road infrastructure is important for ensuring safe travel, yet hazards like potholes and speed bumps pose threats to vehicles and drivers. The Camera-Based Automated Detection System represents a cutting-edge technological endeavor to strengthen road safety through advanced computer vision techniques. The primary objective of this project is to develop a robust algorithm capable of accurately identifying various road anomalies by analyzing camera footage.

Keywords: Speed bumps, Road defects, Object detection, Computer vision, Automated Detection System, Machine learning, YOLOv8, Road safety, Convolutional neural networks, Real-time processing

Acknowledgments

This dissertation is a significant milestone in my life, marking the end of a long educational journey and the start of new experiences. It represents the result of our hard work and dedication, supported by the invaluable help of those around me. I want to sincerely thank everyone who has contributed to the success of this work, directly or indirectly.

First and foremost, I am grateful to God for granting me the strength, perseverance, and wisdom to complete this dissertation. His guidance has been a constant source of inspiration and support.

I am especially grateful to my parents, sisters, and grandparents who have been with me, creating the best environment for my studies and personal growth. I also want to thank all my friends and classmates from school, particularly my best friends Ines, Maria, and Lilia, for their continuous support throughout my academic journey.

I deeply appreciate the guidance and unwavering support of my supervisors, Pr.Cherif LARBES and Dr.Elhaouari KOBZILI, throughout this endeavor. Their patience, motivation, and dedication have made this work meaningful. They have always been there for me whenever I needed assistance, and I am truly grateful for their mentorship.

Lastly, I would like to express my gratitude to all the teachers in the Department of Electronics who have supported me over the past three years, leading to the completion of this project.

Amina

Contents

List of Tables

List of Figures

List of Acronyms

- **AI** : Artificial intelligence
- **ANN** : Artificial neural networks
- **AP** : Average Precision
- **BCE** : Binary Cross-Entropy
- **CNN** : Convolutional Neural Network
- **CPU** : Central Processing Unit
- **CSPN** : Cross Stage Partial Network
- **DFL** : Distributional Focal Loss
- **DNN** : Deep Neural Network
- **DPM** : Deformable Part-based Models
- **FPN** : Feature Pyramid Networks
- **FCL** : Fully Connected Layer
- **GPU** : Graphic Processor Unit
- **GHL** : Gradient Harmonized Mechanism
- **HOG** : Histogram of Oriented Gradients
- **IoU** : Intersection over Union
- **mAP** : mean Average Precision
- **MLP** : Multi-layer perceptrons
- **NLP** : Natural language processing
- **NMS** : Non-max Suppression
- **ReLU** : Rectified Linear Unit
- **RGB** : Red Green Blue
- **R-CNN** : Region-based Convolutional Neural Network
- **SNN** : simulated neural networks
- **SSD** : Single Shot Multibox Detector

- **SPPF** : Spatial Pyramid Pooling with Feature Fusion
- **SVM** : Support Vector Machines
- **TanH** : hyperbolic tangent
- **TPU** : Tensor Processing Unit
- **YOLO** : You Only Look Once

General Introduction

"Neglecting the road ahead invites accidents instead"

Context and Statement of The Problem

The quality of roads is critical to ensuring the safe travel of vehicles. Potholes, speed bumps, and other pavement irregularities can cause vehicle damage and lead to hazardous traffic accidents. Therefore, detecting and characterizing these anomalies is vital for minimizing accident risks and vehicle wear. However, street images are inherently complex, with redundant and substantial information, and significant measurement noise, making the detection of road anomalies a challenging task.

Since the advent of high-speed roads, the scientific community has been dedicated to reducing road accidents by identifying surface defects and speed bumps. This effort gained momentum with advancements in sensor technology and the application of computer vision, combined with soft-computing approaches such as deep learning, for automated detection systems. These systems significantly streamline daily maintenance processes and reduce the loss of life and costs associated with traffic-related injuries.

However, one of the significant problems in object detection was reliance on traditional computer vision methods, which often needed help to detect and classify complex objects in varying environmental conditions accurately. Additionally, these methods usually lacked scalability and generalization capabilities, making them less effective for large-scale deployment across road networks and conditions. These limitations hindered the development of robust and reliable systems for detecting road defects and speed bumps, necessitating the exploration of more advanced approaches facilitated by AI and deep learning [10].

Motivation and Objectives

Traffic accidents resulting from road surface defects or unwanted objects continue to have severe consequences, causing fatalities, injuries, and substantial property damage. According to recent studies, including research by **Justo-Silva and Ferreira**[11], the global toll remains significant, with over **1.25** million lives lost and an annual injury count ranging from **20** to **50** million people. Projections still indicate that highway accidents are expected to rank as the fifth-highest cause of mortality by 2030.

A survey conducted in 2019 by **Chen et al**[12]. across approximately 166 countries estimated that road injuries would impose a cost of USD 1.8 trillion on the world economy from 2015 to 2030, equivalent to a **0.12%** annual tax on the global gross domestic product. This underscores the continued global impact of road accidents, now recognized as one of the top three predicted causes of death, posing a significant threat to both lives and economies. Regarding crash causes, the American Association of State and Highway Transportation Officials reports that roadway factors, including road defects and anomalies, contribute to approximately **34%** of reported incidents.

This project aims to enhance road safety by implementing advanced computer vision techniques. The project utilizes a deep learning-based real-time detection algorithm to detect and analyze speed bumps and road defects automatically.

The primary objectives include the development of algorithms to identify various road irregularities, such as potholes, cracks, and surface anomalies, using camera footage. The project strives to improve detection accuracy and reliability by minimizing false positives and negatives, ensuring the system can effectively differentiate between speed bumps and road defects.

Structure of The Thesis

Chapter 1: This section will offer a comprehensive literature review, delving into relevant studies and providing definitions of key terms while explaining various concepts pivotal to the research.

Chapter 2: Here, the focus will be on the database's design and acquisition crucial for training the detection model. It will encompass database design, data acquisition, and data annotation processes necessary for training preparation.

Chapter 3: This chapter will outline the overarching architecture of our work, elucidating the data collection and pre-processing procedures, the YOLOv8 model architecture, and the training methodology employed.

Chapter 4: Presenting the results derived from the trained YOLOv8 model, this section will thoroughly analyze its performance and effectiveness.

Lastly, we will explore the study's limitations, highlight potential areas for enhancement, and propose recommendations for future research.

Chapter 1

Literature Review

1.1 State-Of-The-Art

Detecting road anomalies is essential for ensuring road safety. Advances in computer vision, particularly Convolutional Neural Networks, have transformed this detection process. Convolutional Neural Networks have demonstrated significant potential in accurately identifying and classifying road defects from images.

In the early days of object detection, computer vision relied on fundamental techniques like edge detection and handcrafted features. Edge detection algorithms such as Sobel and Canny were pivotal for identifying road anomaly boundaries based on abrupt intensity changes. Meanwhile, feature-based methods like Hough Transform for line detection and descriptors like Histogram of Oriented Gradients and Local Binary Patterns were more suitable for capturing and analyzing road anomalies. These methods laid the groundwork for later developments by providing initial frameworks for detecting and classifying various anomalies.

The advent of machine learning, particularly with the rise of deep learning, revolutionized object detection capabilities. Histogram of Oriented Gradients coupled with Support Vector Machines became popular for their effectiveness in pedestrian detection and simpler road anomalies. The breakthrough came with Convolutional Neural Networks, where models like Faster Region-based Convolutional Neural Network, Single Shot MultiBox Detector, and You Only Look Once drastically improved detection accuracy and processing speed. Faster Region-based Convolutional Neural Network, introduced in 2015, introduced region proposal networks, while Single Shot MultiBox Detector and You Only Look Once focused on single-shot detection, enabling real-time anomaly detection applications. These methods integrated learning representations directly from images, surpassing traditional feature-based approaches in both accuracy and efficiency.

In the current decade, deep learning architectures continue to evolve, addressing challenges in object detection for road anomalies. Innovations include models like CenterNet, which emphasizes object center point detection, and DEtection TRansformer, a transformer-based approach that simplifies object detection into a set prediction problem. Vision Transformers have also emerged, adapting transformer architectures originally designed for natural language processing to handle image tasks, including object detection. EfficientDet and other variants focus on optimizing computational efficiency without sacrificing performance, crucial for deploying models in real-world scenarios.

Despite advancements, challenges persist in ensuring robustness across varying environmental conditions and road surfaces. Real-time processing remains a priority for applications like autonomous vehicles, necessitating further improvements in model efficiency and deployment on edge devices. Future research is likely to focus on enhancing the scalability, adaptability, and interpretability of object detection models, aiming to meet the demanding requirements of modern transportation systems and safety protocols.

1.2 Definitions

Common Road Defects

Road defects pose numerous challenges to the safety and reliability of road infrastructure, ranging from vehicle damage to accidents. Furthermore, they accelerate road degradation, potentially leading to more severe defects. Without prompt identification and repair, road defects can result in injuries, property damage, and even fatalities [13].

Common road defects that can lead to car accidents and vehicle damage include potholes, cracks, and uneven surfaces. Potholes are particularly hazardous as they can cause severe damage to a vehicle's suspension system, tires, and alignment. Drivers may also lose control of their vehicles when encountering large potholes, leading to accidents. Cracks in the pavement, while less dramatic than potholes, can expand over time and create similarly dangerous conditions. Uneven surfaces and road settlements can disrupt the smooth flow of traffic, causing vehicles to swerve unexpectedly and increasing the likelihood of collisions. Additionally, inadequate drainage can lead to water pooling on roads, creating slippery conditions that further exacerbate the risk of accidents. Addressing these common road defects is crucial for maintaining road safety and preventing vehicle damage.

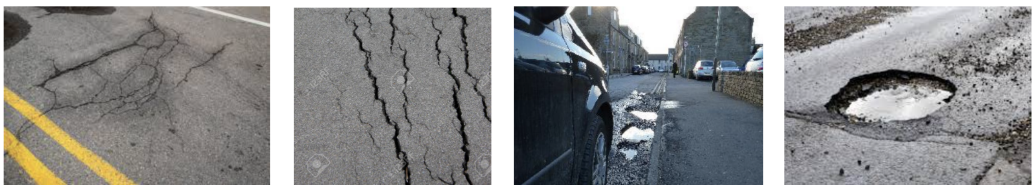
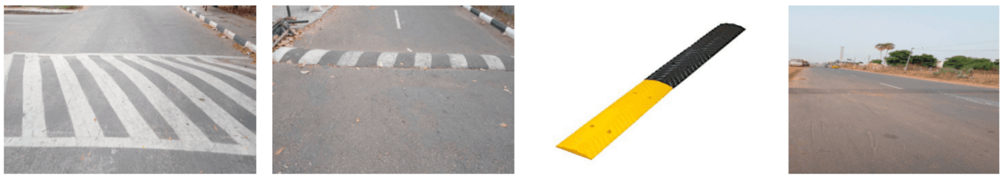


Figure 1.1: Most Common Road Defects

Speed Bumps

A speed bump typically features a distinctive profile designed to slow down vehicles effectively. It is commonly characterized by a rounded, elongated shape, resembling a gentle arc across the width of the road. The dimensions of a speed bump can vary depending on its intended purpose and location, but it typically has a height ranging from **7.6** to **10.2** centimeters and a length of several meters. To facilitate its detection, speed bumps often possess contrasting colors or reflective markings to enhance visibility, especially during low-light conditions or adverse weather. Additionally, some speed bumps may incorporate textural elements or patterns to provide tactile feedback to drivers, further alerting them to their presence and encouraging compliance with reduced speed requirements[14].



(a) Marked Speed hump (b) Marked Speed bump (c) Rumble strips (d) Unmarked hump/bump

Figure 1.2: Speed bump

Artificial Intelligence

According to **John McCarthy**, one of the pioneers in the field, AI is "the science and engineering of making intelligent machines"[15]. Artificial Intelligence is a computer science field that aims to create systems capable of performing tasks that would typically require human intelligence. However, AI is often regarded as a broad and multidimensional concept, challenging to precisely define due to its extensive and evolving nature. For instance, technologies ranging from simple recommendation algorithms used by YouTube, as per Forbes, to complex autonomous driving systems developed by companies like Tesla, are all considered AI. This diversity makes the field both fascinating and mysterious, with definitions evolving as technology progresses.

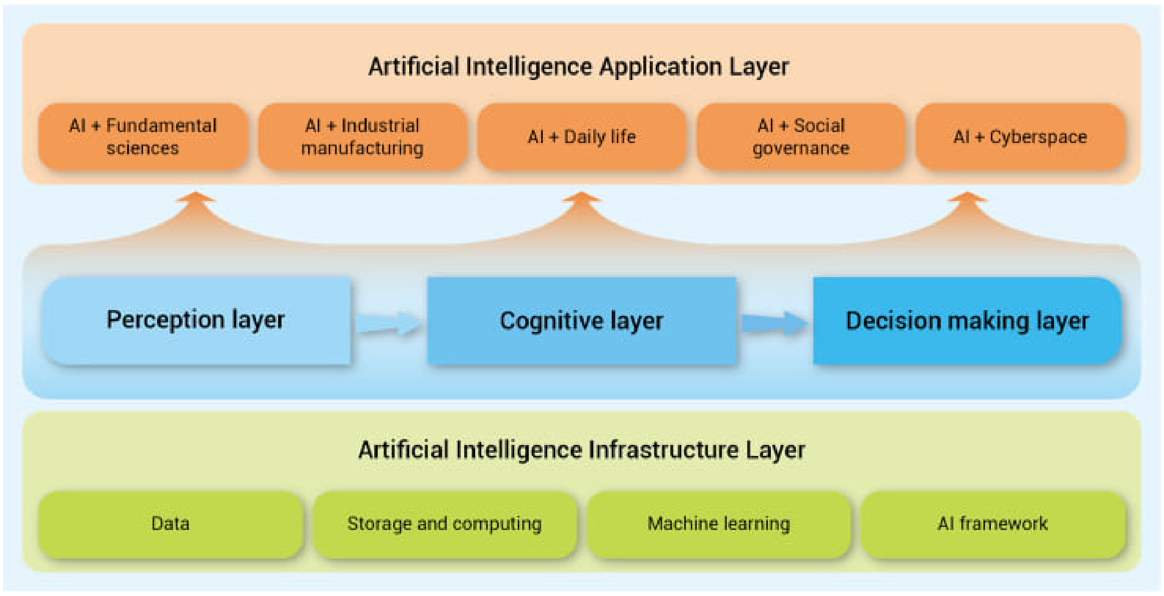


Figure 1.3: Artificial intelligence layers

AI systems operate by integrating extensive datasets with intelligent, iterative algorithms to discern patterns and features within the analyzed data. After each round of data processing, an AI system evaluates its performance and acquires additional expertise. Due to its continuous operation, AI can swiftly execute hundreds, thousands, or even millions of tasks, rapidly accumulating knowledge and proficiency in its designated tasks. it is important to recognize that AI transcends individual programs or applications, constituting an entire scientific discipline. The overarching aim of AI science is to develop computer systems capable of emulating human behavior and employing human-like cognitive processes to tackle intricate problems. To achieve this goal, AI systems employ a diverse range of techniques, processes, and technologies, with machine learning serving as a fundamental component [16].

Machine Learning

Machine learning algorithms empower computers to identify complex patterns and make informed decisions based on data without explicit programming. This iterative learning process enables AI systems to adapt dynamically to evolving situations and continuously refine their performance. Beyond machine learning, AI encompasses a vast array of methodologies, including natural language processing, computer vision, and robotics, all contributing to its multifaceted nature. By amalgamating various techniques and technologies, AI strives to replicate human-like intelligence, enabling computers to exhibit reasoning, problem-solving, and decision-making capabilities akin to human cognition.

There are many different components to an AI system, which we can think of as sub-fields of the overarching science of AI [17].

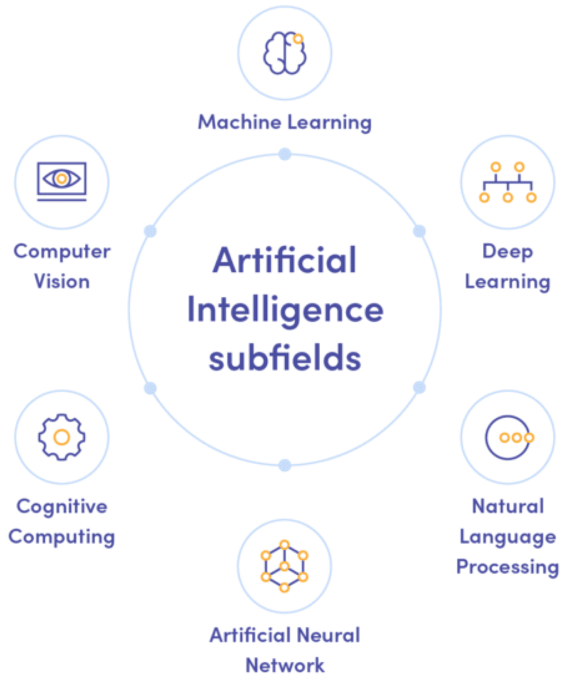

Machine Learning
Computer Vision
Deep Learning
Artificial Intelligence subfields
Cognitive Computing
Natural Language Processing
Artificial Neural Network

Figure 1.4: Artificial intelligence Subfields

Deep Learning

Deep learning is a specific type of machine learning that allows AI to learn and improve by processing data. It utilizes artificial neural networks, which mimic the structure and function of biological neural networks in the human brain, to extract meaningful insights, identify patterns, and generate outcomes through positive and negative feedback mechanisms.

Cognitive Computing

Cognitive computing is a key aspect of AI systems that aims to replicate human-machine interactions. By emulating the cognitive processes of the human brain, cognitive computing enables computer models to perform complex tasks such as text analysis, speech recognition, and image interpretation with human-like proficiency.

Natural Language Processing

Natural language processing (NLP) is a crucial component of AI technology that enables computers to understand, interpret, and generate human language, both written and spoken. NLP is essential for AI-driven systems that interact with humans, facilitating communication and enabling intelligent responses to text and speech inputs.

Computer Vision

Computer vision, a field of AI, enables computers to extract relevant information from digital images, videos, and other visual data to make decisions or formulate recommendations accordingly. While AI empowers computers to think, computer vision enables them to perceive, analyze, and understand their visual environment.

Operating similarly to human vision, computer vision nonetheless faces certain challenges. Unlike humans, who can train to discern objects, assess their distance, detect motion, and identify anomalies in an image, computers must perform these tasks much faster using cameras, data, and algorithms. These technologies replicate the functions of our eyes, optic nerves, and visual cortex.

Computer vision finds widespread application in various sectors such as energy, utilities, manufacturing, and automotive. Indeed, this field is experiencing steady growth and is projected to reach a value of **$31.3** billion by 2025, underscoring its increasing importance in our society and technological development.

Neuronal Network

A neural network is a type of machine learning model designed to emulate the decision-making process of the human brain. It comprises interconnected layers of artificial neurons, including an input layer, one or more hidden layers, and an output layer. Each neuron calculates its output based on inputs received from connected neurons, applying weights and thresholds to determine activation.

They learn from training data, adjusting weights and thresholds to improve accuracy over time. Once trained, they excel in tasks like speech or image recognition, processing data much faster than human experts. They form a crucial part of machine learning and are foundational to deep learning models.

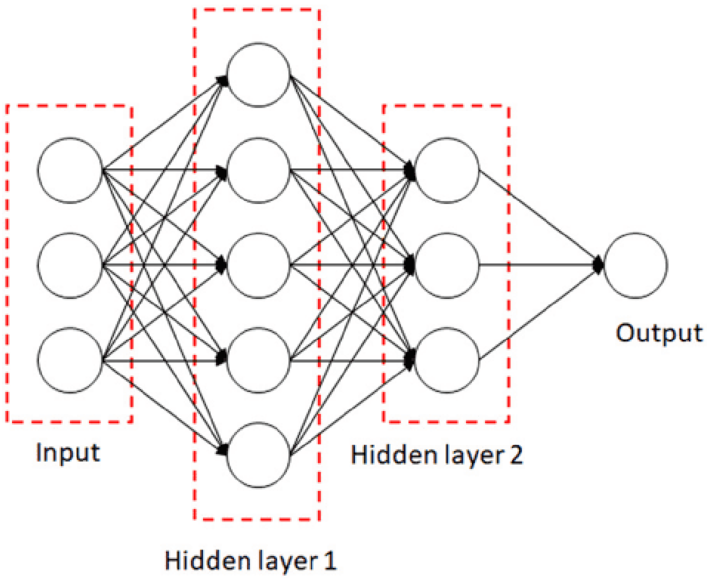


Figure 1.5: Neural networks architecture

Neural networks, encompassing various architectures such as ANNs and simulated neural networks (SNNs), form the backbone of modern machine learning systems. Multi-layer perceptrons stand out as a fundamental model among these architectures. MLPs are a specific type of neural network composed of interconnected layers of artificial neurons, each layer contributing to the network's ability to process and learn from data.

By leveraging input patterns and adjusting weights across hidden layers, MLPs excel in tasks such as image recognition and classification. Thus, while neural networks represent a broad concept in machine learning, MLPs offer a concrete implementation that harnesses the power of interconnected neurons to solve complex problems in various domains.

In an MLP, perceptrons are structured in interconnected layers. The input layer receives input patterns, while the output layer generates classifications or output signals based on these patterns. For instance, input patterns could entail various attributes of an image, with potential outputs representing different categories or labels associated with these attributes.

Hidden layers within the MLP iteratively adjust the input weights until the neural network minimizes its error. These hidden layers are believed to identify crucial features in the input data that contribute to predicting the outputs. This process, referred to as feature extraction, resembles statistical techniques like principal component analysis in its objective.

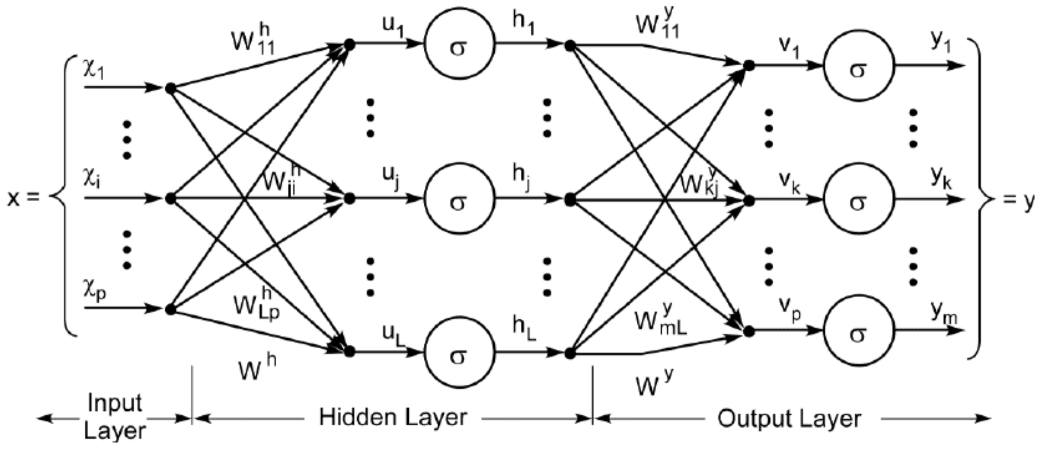


Figure 1.6: Multi-layer perceptrons architecture

Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is an advanced type of Deep Learning algorithm designed for processing images. It can analyze an input image, assigning significance to various features within it through the use of learnable weights and biases. This process enables the network to distinguish between different elements present in the image.

Unlike traditional classification methods that often require extensive pre-processing, ConvNets typically have lower pre-processing requirements. While older techniques necessitate the manual design of filters, ConvNets can autonomously learn these filter characteristics given sufficient training data [18].

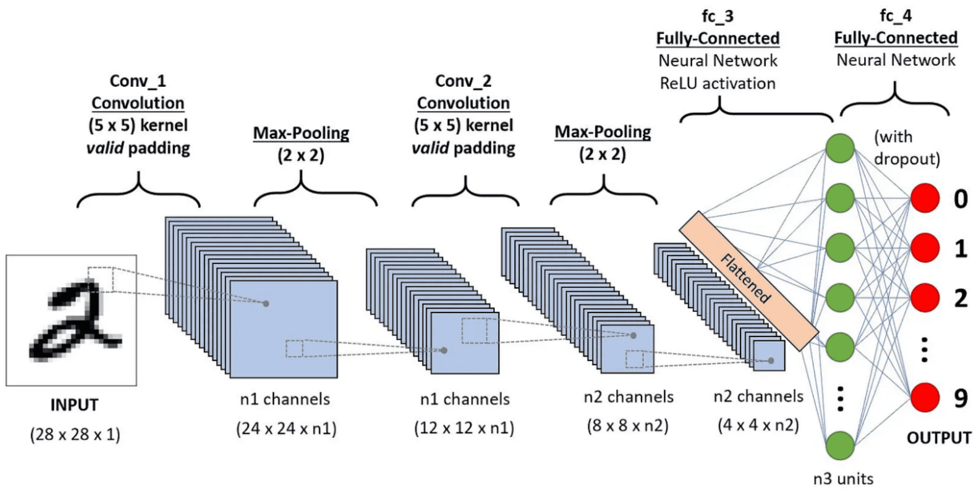


Figure 1.7: A CNN sequence to classify handwritten digits

Sensors

In the context of detecting road defects and speed bumps, various sensors can be used to gather relevant data. These include infrared cameras, which detect infrared radiation emitted by objects and convert it into thermal images. These cameras can capture temperature variations on the road surface, indicating potential defects such as cracks or unevenness. Thermal cameras for night vision operate similarly to infrared cameras but are specifically designed to detect heat signatures in low-light conditions, allowing vehicles to navigate safely at night.

LiDAR (Light Detection and Ranging) sensors utilize laser pulses to measure distances to objects and create precise 3D maps of the road environment. Ultrasound detectors emit high-frequency sound waves and analyze their reflections to detect objects and obstacles in the vehicle's surroundings. Laser sensors emit laser beams and measure the time it takes for the beams to bounce back from objects, providing accurate distance measurements.

Despite the versatility of these sensors, our work primarily relies on a visible camera embedded in the vehicle. This choice is driven by its ability to capture high-resolution, detailed visual data under natural lighting conditions, its cost-effectiveness, ease of integration with existing systems, and suitability for processing with established image analysis techniques.

Camera

A camera is an image capture device specifically used to gather visual data about the road surface and its surroundings. It is typically mounted on a vehicle equipped with advanced detection systems to capture high-resolution images of the road as the vehicle moves, allowing image processing and computer vision algorithms to detect and analyze roadway features such as speed bumps, cracks, potholes, and other defects.

Object Detection and Image Classification

Object detection and image classification are two fundamental tasks within the domain of computer vision, each serving distinct purposes in analyzing and understanding visual data. Object detection involves classifying and locating multiple objects within an image, and assigning specific labels or categories to each detected object. This task involves recognizing objects and precisely delineating their boundaries through bounding boxes. For instance, in the context of autonomous driving systems, object detection algorithms can identify pedestrians, vehicles, traffic signs, and other relevant entities on the road, providing critical information for navigation and decision-making [19].

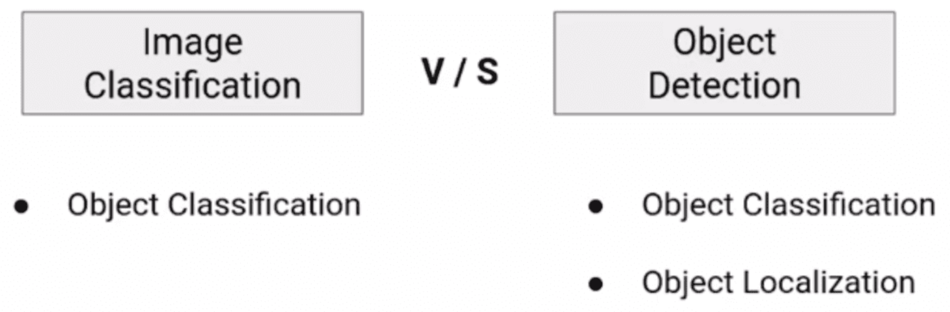


Figure 1.8: Object Localisation and Object Classification

On the other hand, image classification revolves around categorizing an entire image into a single predefined class or category based on its overall content. Unlike object detection, which focuses on localizing and labeling individual objects, image classification treats the entire image as a single entity. For example, given an image of a natural scene, an image classification model may classify it as depicting a forest, a beach, a mountain landscape, or any other relevant category. This task is particularly useful in applications where understanding the general context or content of an image is paramount, such as content-based image retrieval or organizing large image databases.

1.3 Object Detection Methods in Computer Vision

Methods based on deep neural networks, such as CNNs, are frequently employed for object detection. The approaches used for Real-time detection, such as **YOLO** (You Only Look Once), Faster R-CNN (Region-based Convolutional Neural Network), and SSD (Single Shot Multibox Detector), are at the forefront of research. The utilization of massive datasets for model training, image preprocessing techniques, and the integration of advanced deep learning algorithms contribute to enhancing detection accuracy [20].

Since the popularization of deep learning in the early 2010s, there's been a continuous progression and improvement in the quality of algorithms used to solve object detection.

One-Stage vs. Two-Stage Deep Learning Object Detection

Object detection methods can be categorized into One-stage vs. two-stage object detectors. In general, deep learning-based object detectors extract features from the input image or video frame.

An object detector solves two subsequent tasks :
Task #1: Find an arbitrary number of objects (possibly even zero)
Task #2: Classify every single object and estimate its size with a bounding box.

To simplify the process, we can separate those tasks into two stages. Other methods combine both tasks into one step (single-stage detectors) to achieve higher performance at the cost of accuracy.

In two-stage object detectors, the approximate object regions are proposed using deep features before these features are used for the image classification and bounding box regression for the object candidate. The two-stage architecture involves object region proposal with conventional Computer Vision methods or deep networks, followed by object classification based on features extracted from the proposed region with bounding-box regression. Two-stage methods achieve the highest detection accuracy but are typically slower. Because of the many inference steps per image, the performance (frames per second) is not as good as one-stage detectors. Various two-stage detectors include region convolutional neural network (RCNN), with evolutions Faster R-CNN or Mask R-CNN. The latest evolution is the granulated RCNN (G-RCNN). Two-stage object detectors first find a region of interest and use this cropped region for classification. However, such multi-stage detectors are usually not end-to-end trainable because cropping is a non-differentiable operation.

One-stage detectors predict bounding boxes over the images without the region proposal step. This process consumes less time and can therefore be used in real-time applications. One-stage object detectors prioritize inference speed and are super fast but not as good at recognizing irregularly shaped objects or a group of small objects. The most popular one-stage detectors include the YOLO and SSD. The latest real-time detectors are YOLOv8 (2023), YOLOv7 (2022), YOLOR (2021), and YOLOv4-Scaled (2020).

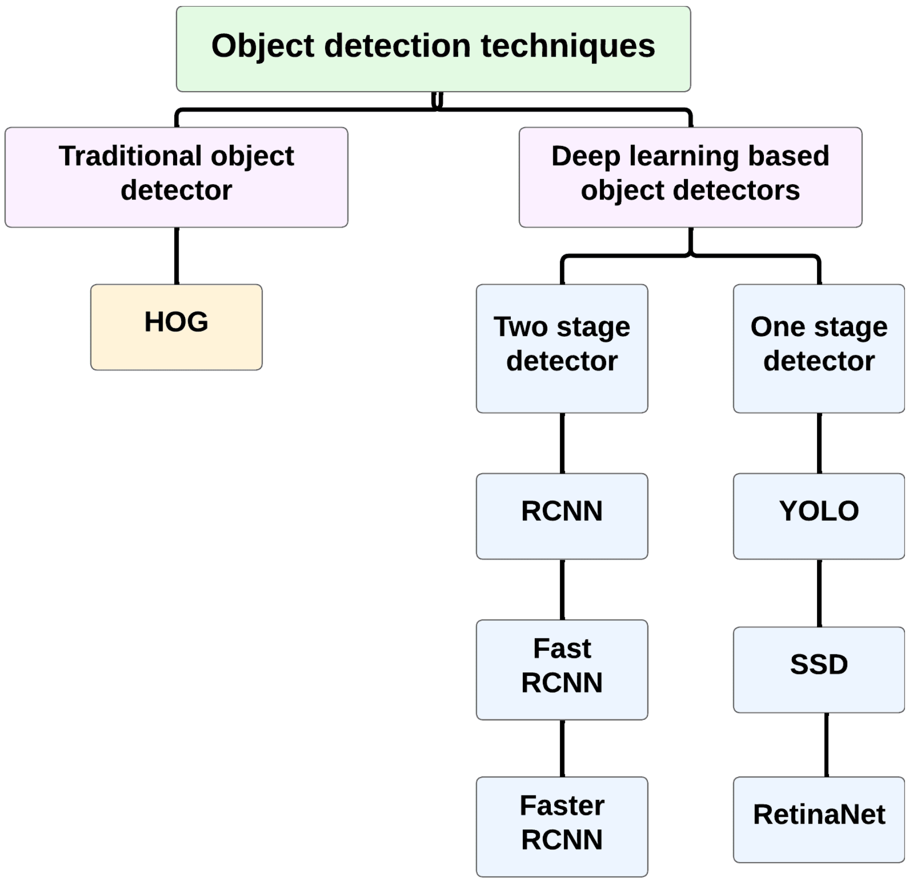

Object detection techniques
Traditional object detector
Deep learning based object detectors
HOG
Two stage detector
One stage detector
RCNN
Fast RCNN
Faster RCNN
YOLO
SSD
RetinaNet

Figure 1.9: Object detection techniques

1.3.1 Histogram of Oriented Gradients

The Histogram of Oriented Gradients is a classic method for object detection, initially introduced in 1986 but gaining significant traction in computer vision applications around 2005. It employs a feature extractor to discern objects in images by extracting essential information while filtering out extraneous details. The feature descriptor in HOG transforms image sizes into compact feature vectors, utilizing gradient orientation to pinpoint critical image components.

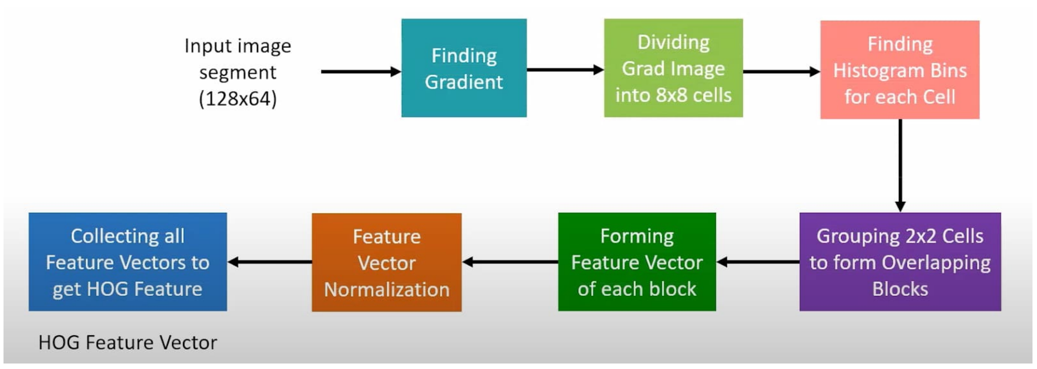

Input image segment (128x64)
Finding Gradient
Dividing Grad Image into 8x8 cells
Finding Histogram Bins for each Cell
Grouping 2x2 Cells to form Overlapping Blocks
Forming Feature Vector of each block
Feature Vector Normalization
Collecting all Feature Vectors to get HOG Feature
HOG Feature Vector

Figure 1.10: Architecture of HOG

In the architecture of HOG, the algorithm computes gradients for each pixel, generating histograms of gradient orientations to capture local image characteristics. This process involves dividing the image into small cells, calculating histograms within each cell, and then normalizing and concatenating the resulting feature vectors. Despite its effectiveness in creating useful feature descriptors and achieving high-accuracy object detection when combined with support vector machines (SVMs), HOG has its limitations.

HOG's drawbacks include its computational intensity, especially for complex images, and its diminished effectiveness in detecting objects in confined spaces. Despite these limitations, HOG remains valuable as an initial method for object detection, serving as a benchmark for evaluating the performance of other detection algorithms. It finds widespread use in various applications, including pedestrian detection and facial landmark recognition, owing to its decent accuracy and edge detection capabilities. Thus, while HOG may not be the optimal solution in all scenarios, it lays the foundation for further advancements in object detection algorithms.

1.3.2 Region-based Convolutional Neural Networks

Region-based CNN represents a groundbreaking approach to object detection by leveraging deep learning models. This methodology involves selecting proposed regions within an image, such as through anchor boxes, and subsequently labeling these regions with their respective categories and bounding boxes based on predefined classes. Once labeled, a CNN extracts features from each proposed region.

In the R-CNN framework, the input image is divided into numerous region sections, each undergoing individual CNN processing. This detailed approach, while effective, comes with significant time constraints, particularly during training. R-CNN handles classification and bounding box creation for each region separately, resulting in longer training times due to the sequential application of the neural network on each region.

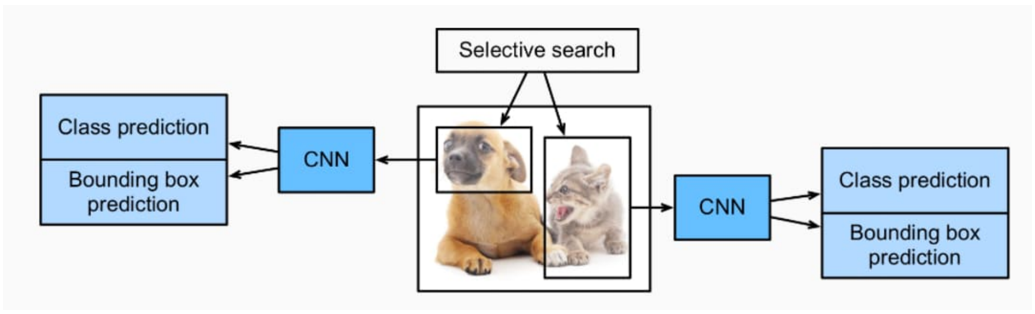


Figure 1.11: Architecture of R-CNN

This model faces several challenges. Firstly, while it effectively extracts features using pre-trained CNN models, extracting region proposals and selecting the best regions is slow. This results in prolonged training and high prediction times, necessitating substantial computational resources and making the architecture expensive. Additionally, the initial step of candidate selection can lead to poor choices, with limited opportunities for improvement, potentially impacting the performance of the trained model.

1.3.3 Fast R-CNN

Fast R-CNN is an advanced object detection model that builds upon the limitations of the original R-CNN, offering significant improvements in speed and efficiency while maintaining high accuracy. The primary innovation of Fast R-CNN is its ability to perform object detection and classification in a single, streamlined process, significantly reducing the computational overhead and time required for detection tasks.

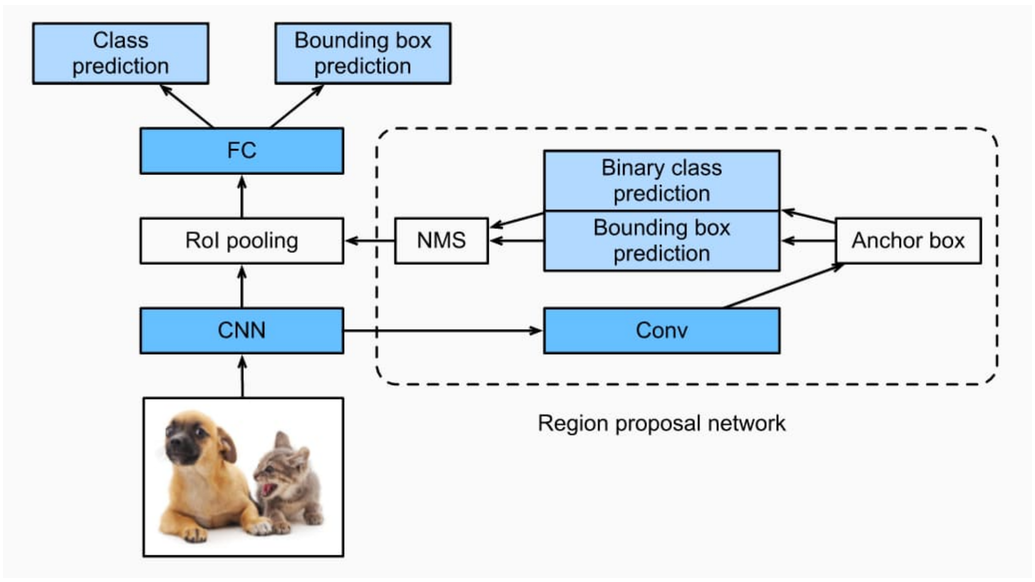


Figure 1.12: Architecture of Fast R-CNN

The process begins with an input image, which is passed through a Convolutional Neural Network (CNN) to generate a set of feature maps. Unlike the original R-CNN, which performs CNN processing on each proposed region independently, Fast R-CNN processes the entire image through the CNN once. This approach allows the network to capture essential features such as edges, textures, and other important details across the whole image, leading to more efficient and comprehensive feature extraction.

The Region Proposal Network (RPN) is a critical component of Fast R-CNN. The RPN works on the feature maps generated by the CNN to propose potential object regions within the image. It uses convolutional layers to process these feature maps and generate anchor boxes of various sizes and aspect ratios. The RPN performs two main tasks: binary classification to predict whether an anchor box contains an object or background, and bounding box regression to adjust the anchor boxes to better fit the proposed objects. This step is efficient and integral to reducing the number of regions that need further processing.

After the RPN proposes potential object regions, Non-Maximum Suppression (NMS) is applied to eliminate redundant and overlapping proposals. NMS ensures that only the most promising object regions, which best represent the actual objects in the image, are retained for further analysis. This step is crucial for reducing the number of false positives and focusing computational resources on the most relevant regions.

The selected regions of interest from NMS are then extracted from the feature maps and pooled to a fixed size using Region of Interest (RoI) pooling. RoI pooling ensures that all proposed regions, regardless of their original size, are resized to a uniform dimension. This step is essential for standardizing the input for the fully connected layers that follow, allowing the network to handle regions of varying sizes efficiently.

The pooled features are fed into fully connected layers for the final stages of object detection. During this phase, the model performs two tasks: class prediction and bounding box regression. Class prediction assigns a class label to each proposed region, identifying the object it contains (e.g., dog, cat, or background). Simultaneously, bounding box regression refines the coordinates of the bounding boxes to ensure they tightly enclose the detected objects. This dual-task approach enables Fast R-CNN to deliver precise object detection and classification results.[21].

1.3.4 Single Shot Detector

The Single Shot Detector offers a rapid solution for real-time object detection, outperforming Faster R-CNN by achieving nearly five times faster processing speed while maintaining high prediction accuracy. Unlike Faster R-CNN, this approach eliminates the need for a region proposal network and relies on multi-scale features and default boxes for efficient computation. Its architecture comprises three main components: feature extraction, detection heads, and non-maximum suppression layers, enabling efficient computation of bounding boxes without sacrificing accuracy. However, SSD may suffer from decreased image resolution and performance degradation for small-scale objects compared to Faster R-CNN.

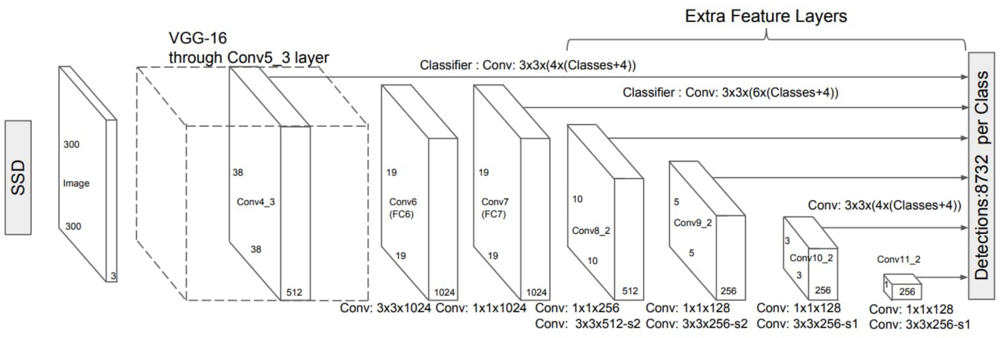


Figure 1.13: SSD architecture

1.3.5 RetinaNet

Introduced in 2017, RetinaNet marked a significant advancement in single-shot object detection, surpassing several established algorithms of its time. Its architecture outpaced YOLOv2 and SSD models in object detection capabilities while maintaining comparable speed. Additionally, RetinaNet demonstrated competitive accuracy similar to the R-CNN models.

In contrast to its predecessors, this algorithm replaces the traditional cross-entropy loss with focal loss, effectively addressing class imbalance issues commonly encountered in models like YOLO (with its first versions) and SSD. Comprising three key components, RetinaNet leverages the ResNet model, specifically ResNet-101, the feature pyramid network (FPN), and the focal loss. The feature pyramid network plays a crucial role in overcoming limitations present in prior architectures by amalgamating semantically rich features from lower-resolution images with semantically weaker features from higher-resolution images.

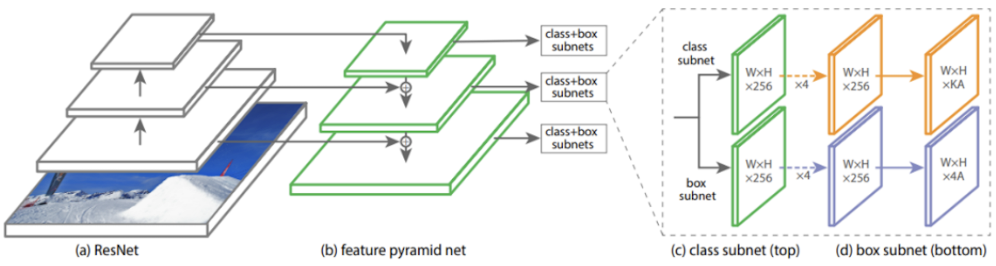


Figure 1.14: RetinaNet architecture

RetinaNet's output encompasses both classification and regression models, akin to other object detection methods. The classification network facilitates multi-class predictions, while the regression network deduces bounding boxes corresponding to the classified entities.

Moreover, it stands out as a top-tier method for object detection across various tasks. It is a robust alternative to single-shot detectors, delivering swift and precise results for diverse image-related endeavors. Notably, the RetinaNet object detection algorithm finds application in a broad spectrum of scenarios, excelling in tasks such as object detection in aerial and satellite imagery, showcasing its versatility and efficacy in real-world applications.

1.4 YOLO

1.4.1 Introduction

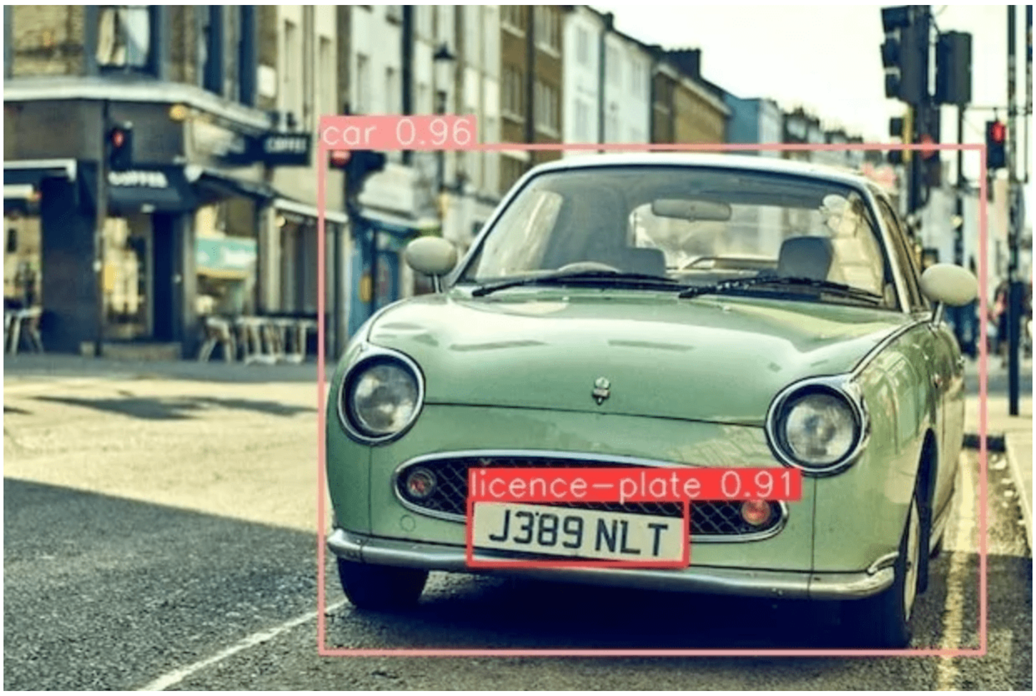


Figure 1.15: YOLO

In the realm of computer vision, humans possess a remarkable ability to swiftly identify objects in images, discern their locations, and infer their interactions. This rapid and accurate processing capability enables us to effortlessly navigate complex tasks such as driving with minimal conscious effort. The development of fast and precise algorithms for object detection holds immense potential, offering opportunities for autonomous vehicle navigation, real-time scene interpretation for assistive technologies, and the advancement of responsive robotic systems.

Traditional object detection systems often repurpose classifiers, evaluating them at various positions and scales across an image to identify objects. For instance, methodologies like deformable parts models (DPM) adopt a sliding window approach, systematically running classifiers over the entire image. Recent advancements, exemplified by approaches like R-CNN, employ region proposal methods to generate potential bounding boxes, subsequently subjecting these regions to classification and post-processing techniques to refine detections and eliminate redundancies. However, these multi-step pipelines present challenges in optimization due to the need to separately train each component.

A paradigm shift in object detection is evident in systems like YOLO, which reconceptualize the task as a single regression problem. This innovative approach bypasses the complexities of traditional pipelines by directly predicting bounding box coordinates and class probabilities from image pixels in a single pass. With YOLO, the detection process becomes streamlined: a single convolutional network generates multiple bounding box predictions along with corresponding class probabilities. By training on complete images, YOLO optimizes detection performance holistically, offering advantages over conventional methods [22].

The YOLO detection system operates with simplicity and efficiency. By applying a single convolutional network, and thresholding the resulting detections based on confidence scores, YOLO achieves accurate object localization and classification straightforwardly. This unified model architecture not only simplifies the detection process but also enhances performance, demonstrating the potential for advancing object detection capabilities in diverse applications.

Evaluation Metrics

Effective evaluation of object detection models relies on a comprehensive understanding of performance metrics, which serve as critical tools for assessing both the accuracy and efficiency of these models. These metrics provide valuable insights into how well a model can identify and localize objects within images, while also shedding light on its ability to manage false positives and false negatives. Given their importance, mastering these metrics is essential for evaluating and refining the performance of object detection models. In this guide, we will delve into various performance metrics associated with YOLOv8, elucidating their significance, interpretation, and practical implications[23].

Intersection over Union (IoU) : Is a widely used metric for assessing localization accuracy and determining localization errors within object detection models. To compute IoU, the intersecting region between the predicted bounding box and the ground truth bounding box corresponding to the same object is identified. Subsequently, the total area encompassed by both bounding boxes, referred to as the "Union," and the area of overlap between them, known as the "Intersection," are calculated. By dividing the area of intersection by the area of union, a ratio is obtained that represents the extent of overlap relative to the total area, this ratio serves as an effective measure of how closely the predicted bounding box aligns with the original bounding box.

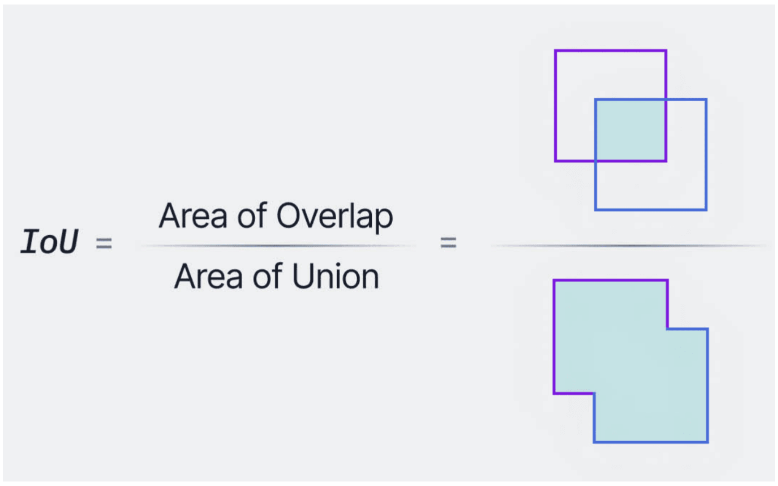


Figure 1.16: Intersection over Union

Precision and Recall: These are fundamental metrics for evaluating the performance of object detection models. Precision measures the proportion of true positives among all positive predictions, reflecting the model's ability to avoid false positives. Recall calculates the proportion of true positives among all actual positives, indicating the model's capability to detect all instances of a class. Balancing precision and recall is crucial for achieving high-performance object detection models.

$$Precision = \frac{TP}{TP + FP} \tag{1.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{1.2}$$

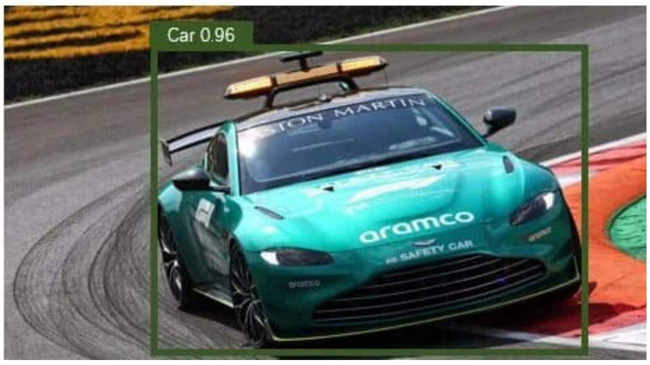
Average Precision (AP) : This key metric assesses the precision-recall performance of an object detection model. AP is calculated by computing the area under the precision-recall curve, depicting the trade-off between precision and recall at various classification thresholds. A higher AP value signifies superior performance in both precision and recall, indicating the model's effectiveness in accurately identifying objects while minimizing false positives and false negatives.

Mean Average Precision (mAP): Extending the concept of AP, mAP averages the AP values across multiple object classes. In multi-class object detection scenarios, mAP comprehensively evaluates the model's performance across different classes. By considering the average precision across all classes, mAP offers a holistic assessment of the model's overall effectiveness in object detection tasks.

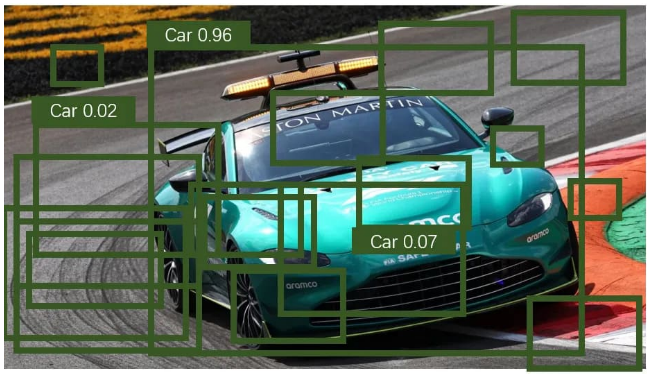
F1 Score: This metric, a harmonic mean of precision and recall, provides a balanced assessment of model performance. By incorporating false positives and false negatives into its calculation, the F1 Score comprehensively evaluates a model's effectiveness in object detection tasks. A higher F1 Score indicates better overall performance in terms of precision and recall, highlighting the model's ability to achieve accurate and reliable object detection results.

$$F1 - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{1.3}$$

Confidence: The number next to the predicted class in the output images, means how confident the model is that the object in the bounding box belongs to a category. Normally we set minimum confidence to display the predicted bounding box, otherwise, you can have way too many boxes in your prediction output.



(a) the model has a 96% confidence that the object in the box is a car



(b) set display confidence interval too low

Figure 1.17: confidence to display the predicted bounding box [2]

1.4.2 How YOLO Works

The integration of various object detection components into a single neural network streamlines the process. The network utilizes information from the entire image to forecast each bounding box, simultaneously predicting all bounding boxes across all classes within an image. Consequently, the network analyzes the entire image and its contained objects holistically.

YOLO facilitates end-to-end training and real-time speeds while maintaining a high level of average precision. In the system, the input image is segmented into an **S × S** grid. If an object's center falls within a grid cell, that cell is tasked with detecting the object.

Each grid cell predicts **B** bounding boxes and their corresponding confidence scores. These scores reflect both the model's confidence in the box containing an object and the accuracy of the predicted box. Specifically, confidence is calculated as **Pr(Object) * IOU truthpred**. Pr(Object) is a probability value assigned to each grid cell. It indicates how likely it is, according to the model's evaluation, that the grid cell contains an object of interest. If no object is present in the cell, the confidence scores are set to zero. Otherwise, the confidence score equals the intersection over union (IOU) between the predicted box and the ground truth.

Each bounding box comprises five predictions: **x, y, w, h,** and confidence. The **(x, y)** coordinates represent the box's center relative to the grid cell bounds, while width and height are relative to the entire image. The confidence prediction indicates the IOU between the predicted box and any ground truth box.

Furthermore, each grid cell predicts C conditional class probabilities, **Pr(Classi |Object)**, conditioned on the presence of an object in the cell. Regardless of the number of boxes B, only one set of class probabilities is predicted per grid cell.

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \tag{1.4}$$

During testing, the conditional class probabilities and individual box confidence predictions are combined to obtain class-specific confidence scores for each box. These scores reflect both the likelihood of the class appearing in the box and the predicted box's fit to the object [3].

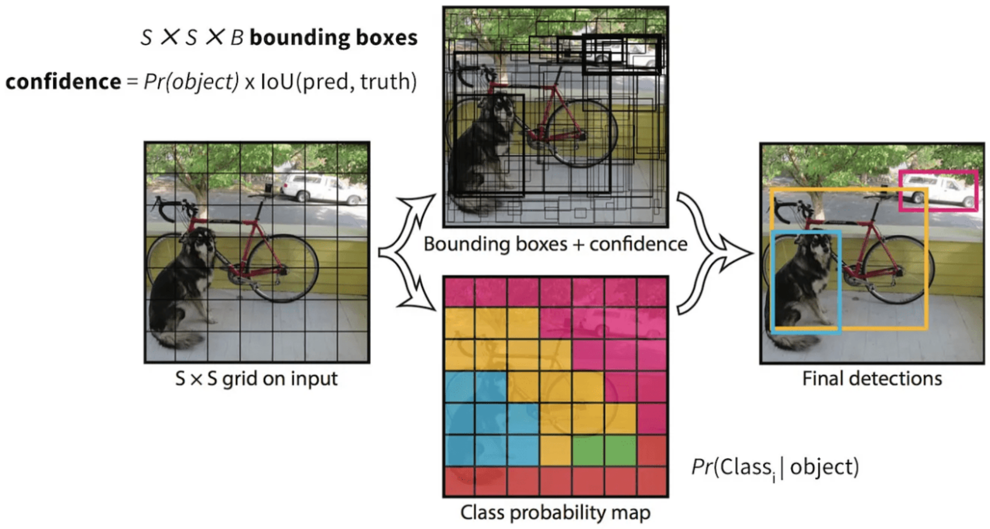


Figure 1.18: The functioning of the YOLO model [3]

1.4.3 Architecture

The model is implemented as a convolutional neural network (CNN) and evaluated using the **PASCAL VOC** [6] detection dataset. In this architecture, the initial convolutional layers are responsible for extracting intricate features from images, while the subsequent fully connected layers predict output probabilities and precise coordinates of detected objects.

Inspired by the GoogLeNet [24] model renowned for image classification, our network architecture comprises 24 convolutional layers followed by 2 fully connected layers. To achieve feature reduction and optimize computational efficiency, instead of using inception modules, we have integrated 1×1 reduction layers followed by 3×3 convolutional layers. This approach, pioneered by Lin et al [25], enhances feature extraction capabilities while maintaining computational efficiency.

Additionally, we have developed a faster variant of YOLO, termed "Fast YOLO," tailored specifically for rapid object detection. Fast YOLO streamlines the architecture by reducing the number of convolutional layers to 9 and optimizing filter sizes, thereby accelerating inference times without compromising detection accuracy. Importantly, despite these architectural modifications, all training and testing parameters remain consistent between the standard YOLO and Fast YOLO configurations, ensuring fair and comparable evaluations.

Initially pre-trained on the ImageNet classification task at reduced resolution, the convolutional layers are subsequently fine-tuned at full resolution to enhance object detection performance. The final output of the network is represented as a $7 \times 7 \times 30$ tensor, encapsulating predictions such as spatial grid attributes, bounding box characteristics, confidence scores, and conditional class probabilities for detected objects within the image [26].

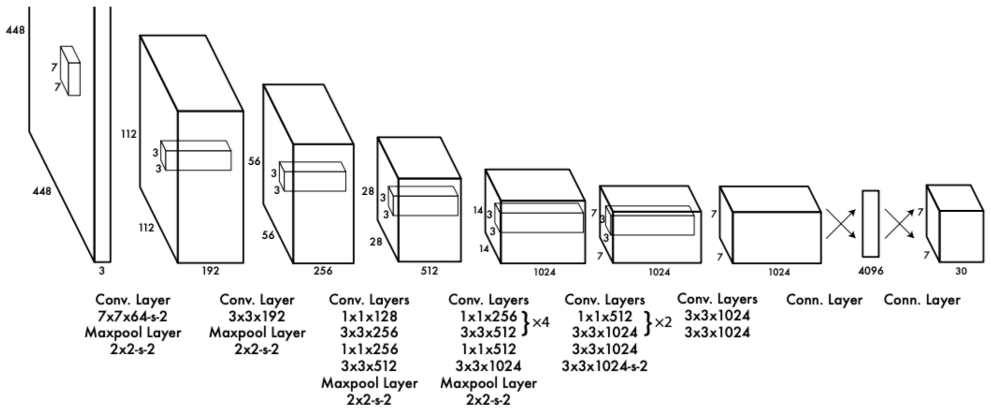

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 1.19: YOLO architecture [4]

1.4.4 Training

During the training process, convolutional layers were pre-trained on the ImageNet 1000-class competition dataset. Specifically, the initial 20 convolutional layers were utilized, along with an average-pooling layer and a fully connected layer. This training period lasted approximately one week, resulting in a top-5 accuracy of **88%** on the ImageNet 2012 validation set, which is comparable to models found in Caffe's Model Zoo.

Following pre-training, adjustments were made to the model to prepare it for detection tasks. This involved incorporating four additional convolutional layers and two FCLs, all with randomly initialized weights. To capture the fine-grained visual details necessary for effective detection, the input resolution of the network was doubled.

The final layer of the network was responsible for predicting both class probabilities and bounding box coordinates. To ensure consistency, the bounding box width and height were normalized by the image dimensions, restricting them to fall within the range of 0 and 1. Additionally, the bounding box coordinates were parametrized as offsets relative to a specific grid cell location, thus constraining them to values between 0 and 1.

In terms of activation functions, a linear activation function was applied to the final layer, while the rest of the layers utilized a leaky rectified linear activation function. This function incorporated a threshold, where values above 0 remained unchanged, while values below 0 were scaled by a factor of 0.1 [27].

$$\phi(x) = \begin{cases} x, & if \quad x > 0 \\ 0.1x, & otherwise \end{cases} \tag{1.5}$$

The optimization objective during training was a sum-squared error in the output of the model. Although this approach was relatively simple to optimize, it did not perfectly align with the primary goal of maximizing average precision. This discrepancy arose because the sum-squared error equally weighed the localization error and classification error, which may not be ideal.

During training, it was crucial to address instances where grid cells contained no objects. In such cases, the confidence scores tended to be pushed towards zero, potentially leading to model instability. To mitigate this issue, adjustments were made to increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that did not contain objects. Furthermore, the sum-squared error metric treated errors in large and small boxes equally.

However, in practice, small deviations in large boxes may have mattered less than in smaller ones. To partially rectify this, the square root of the bounding box width and height was predicted instead of the width and height directly.

The YOLO model predicted multiple bounding boxes per grid cell. A mechanism was implemented to assign responsibility based on the calculated Intersection over Union (IOU) with the ground truth to ensure that each object was associated with only one bounding box predictor during training.

The training process optimized a multi-part loss function, which included terms for bounding box coordinate predictions, confidence predictions, and classification errors. These terms were weighted based on whether an object appeared in a particular grid cell and which bounding box predictor was responsible for that prediction.

$$\begin{aligned}\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-an\sqrt{\hat{h}_i})^2] \\ +\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{noobj}(C_i-\hat{C}_i)^2+\sum_{i=0}^{S^2}1_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2\end{aligned} \tag{1.6}$$

where :
1_i^{obj} denotes if an object appears in cell i and 1_{ij}^{obj} denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

λ_{coord}: This parameter controls the importance of the localization (coordinates) errors of the bounding boxes containing objects. It is a scalar multiplier that adjusts the contribution of the localization errors to the total loss.

(x_i, y_i): Center coordinates of the predicted bounding box in the i-th grid cell.

$(\hat{x}_i, \hat{y}_i)$: Center coordinates of the ground truth bounding box corresponding to the predicted bounding box.

(w_i, h_i): Width and height of the predicted bounding box in the i-th grid cell.

$(\hat{w}_i, \hat{h}_i)$: Width and height of the ground truth bounding box corresponding to the predicted bounding box.

C_i: Confidence score (objectness) of the predicted bounding box in the i-th grid cell, indicating the presence of an object.

$\hat{C}_i$: Ground truth confidence score for the presence of an object in the i-th grid cell.

λ_{noobj}: Parameter that adjusts the weight of the objectness error for grid cells where no object is present. It helps in minimizing the impact of falsely predicting objects in empty grid cells.

$p_i(c)$: Predicted probability of the presence of class c in the i-th grid cell.

$\hat{p}_i(c)$: Ground truth probability of the presence of class c in the i-th grid cell.

The five terms that it sums represent respectively:

- The localization error of the bounding boxes responsible for detecting an object.
- The error on the dimensions of the bounding boxes responsible for detecting an object.
- The error on the objectness of the bounding boxes responsible for detecting an object.
- The error on the objectness of the other bounding boxes.
- The classification error.

Finally, the network underwent training for approximately 135 epochs on the training and validation datasets from **PASCAL VOC 2007 and 2012** [6]. To prevent overfitting, techniques such as dropout and extensive data augmentation were employed. Dropout, with a rate of 0.5, was applied after the first connected layer to prevent co-adaptation between layers. Data augmentation techniques included random scaling, translations, and adjustments to the exposure and saturation levels of the images.

1.4.5 Limitations of YOLO

YOLO imposes stringent spatial constraints on bounding box predictions by limiting each grid cell to predict only two boxes and one class. This constraint restricts the model's ability to predict nearby objects efficiently, particularly challenging for small objects occurring in clusters, such as flocks of birds.

Given that the model learns bounding box predictions from data, it faces difficulties generalizing to objects with unconventional aspect ratios or configurations. Additionally, the model relies on relatively coarse features for bounding box predictions due to multiple downsampling layers in its architecture.

Furthermore, although the model is trained using a loss function that approximates detection performance, it treats errors uniformly across small and large bounding boxes. However, a small error in a small box significantly impacts Intersection over Union (IOU), contrasting with the lesser impact of a similar error in a large box. As a result, incorrect localizations are the primary source of error in the model.

1.4.6 Comparaison of YOLO with Other Detection Systems

In the realm of object detection, several methods have been developed, each with its strengths and weaknesses. When considering real-time detection of speed bumps and road defects, YOLO stands out for several reasons.

YOLO's unified architecture presents a distinct advantage, enabling end-to-end optimization and significantly faster inference speeds compared to multi-stage approaches like R-CNN and Fast R-CNN. Its single-stage detection process and efficient design make it particularly well-suited for real-time applications. YOLO's adaptability further enhances its suitability, as it efficiently detects objects of various sizes and shapes, accommodating the diverse characteristics of speed bumps and road defects. Moreover, YOLO strikes a balance between speed and accuracy, ensuring real-time performance without compromising the quality of detection, which makes it a compelling choice for applications requiring swift and reliable identification of these road hazards.

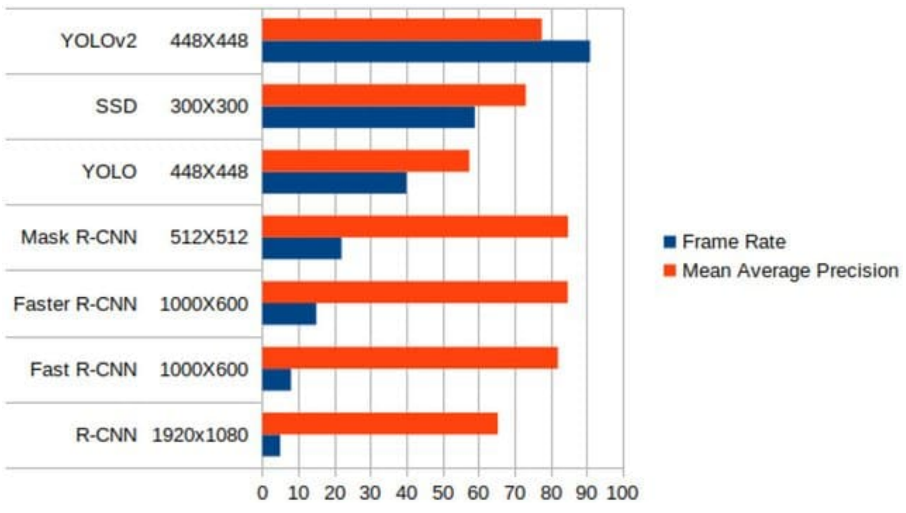


Figure 1.20: Comparison to other detection systems [5]

1.4.7 Comparaison of YOLO with Real-time Detection Systems

Various research endeavors have aimed to optimize standard detection pipelines for speed. Fast YOLO emerges as the fastest object detection method on PASCAL, showcasing a remarkable mAP of **52.7%**, more than doubling the accuracy of previous real-time detection methods. YOLO further advances mAP to **63.4%** while maintaining real-time performance. Additionally, training YOLO using VGG-16 yields a more accurate model, albeit significantly slower. While this model is useful for comparing with other systems relying on VGG-16, its slower speed directs the focus of the paper toward faster YOLO models. Despite notable advancements, the fastest DPM still falls short of real-time performance by a factor of two, although it effectively accelerates DPM without sacrificing much mAP.

R-CNN minus R improves upon its predecessor by replacing Selective Search with static bounding box proposals, yet it remains insufficiently fast for real-time applications. Similarly, Fast R-CNN accelerates the classification stage but still relies on selective search for bounding box proposals, resulting in speeds far from real-time. The introduction of Faster R-CNN replaces selective search with a neural network for proposing bounding boxes, achieving varying speeds and accuracies in different models. However, despite improvements, these models lag behind YOLO in terms of both speed and accuracy.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	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[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Figure 1.21: Real-Time Systems on PASCAL VOC 2007 [6], comparing the performance and speed of fast detectors [7]

On the VOC 2012 test set, YOLO achieves a mAP of **57.9%** which is closer to the original R-CNN using VGG-16. The system faces challenges with small objects compared to its closest competitors. For categories such as bottle, sheep, and tv/monitor, YOLO scores **8-10%** lower than R-CNN or Feature Edit. However, in other categories like cat and train, YOLO achieves higher performance. The combined Fast R-CNN + YOLO model emerges as one of the highest-performing detection methods. Fast R-CNN sees a **2.3%** improvement from the combination with YOLO, elevating it 5 spots up on the public leaderboard [7].

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Figure 1.22: PASCAL VOC 2012 [6] Leaderboard [7]

1.4.8 YOLO versions [1]

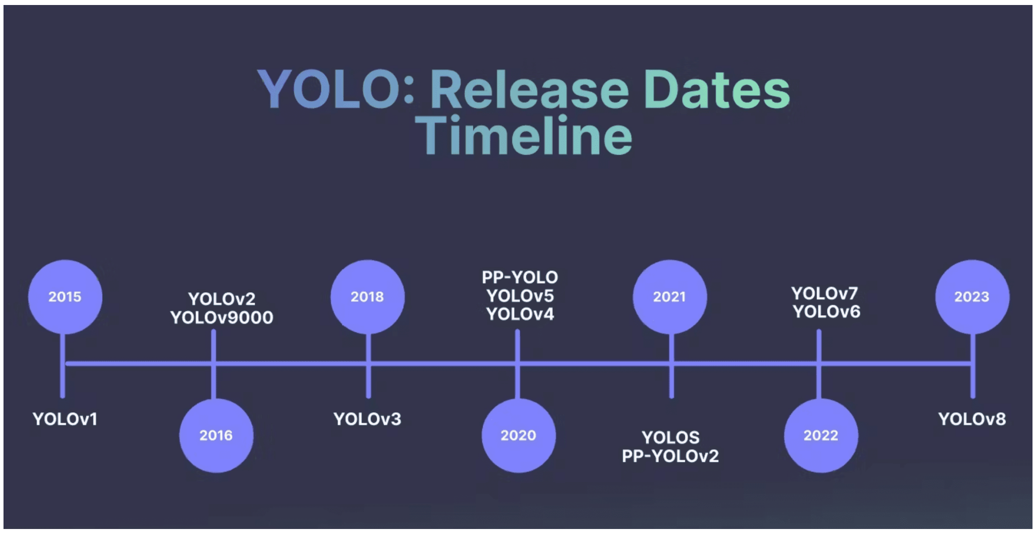


Figure 1.23: Release dates timeline

YOLOv2 : YOLOv2, also referred to as YOLO9000, emerged in 2016 as an evolution of the original YOLO algorithm, aiming for enhanced speed, accuracy, and broader object class detection capabilities. This iteration introduced several improvements, including adopting a different CNN backbone known as Darknet-19 [28], derived from the VGGNet [29] architecture but featuring simplified progressive convolution and pooling layers.

An essential enhancement in YOLOv2 lies in the incorporation of anchor boxes. These predefined bounding boxes, varying in aspect ratios and scales, assist in predicting final bounding boxes by combining them with predicted offsets. This mechanism empowers the algorithm to effectively handle a diverse array of object sizes and aspect ratios.

Furthermore, YOLOv2 integrates batch normalization, a technique aiding in model accuracy and stability. Additionally, it adopts a multi-scale training strategy, where the model is trained on images at various scales, followed by averaging the predictions. This strategy particularly enhances the detection performance for smaller objects.

Another pivotal aspect of YOLOv2 is the introduction of a new loss function tailored to object detection tasks. This loss function calculates the sum of squared errors between predicted and ground truth bounding boxes alongside class probabilities, optimizing detection accuracy.

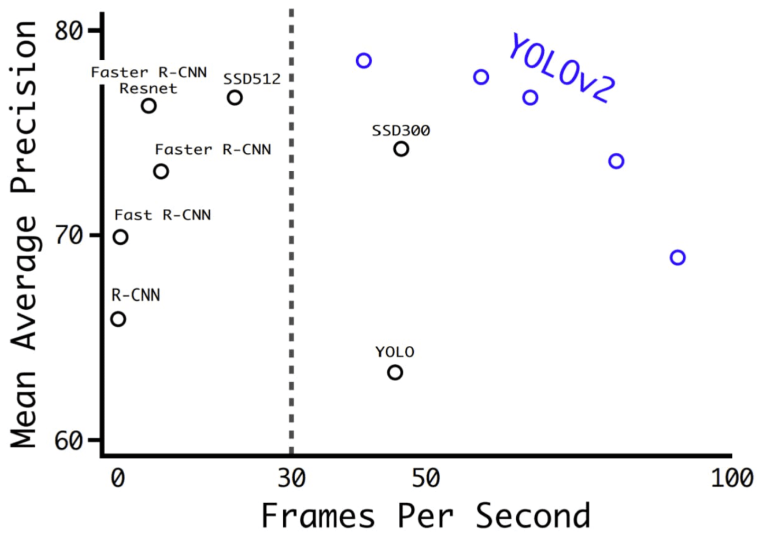


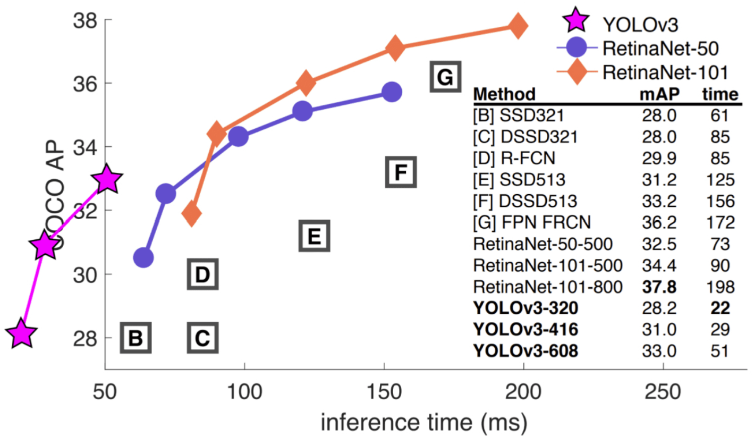
Figure 1.24: The results obtained by YOLOv2 compared to the original version and other contemporary models [7]

YOLOv3 : YOLOv3 debuted in 2018 with the primary objective of enhancing both accuracy and speed compared to YOLOv2. A marked enhancement in YOLOv3 is the adoption of a new CNN architecture known as Darknet-53. This architecture, a variant of the ResNet architecture, is specifically tailored for object detection tasks, boasting 53 convolutional layers and achieving state-of-the-art results across various object detection benchmarks.

Another key advancement in YOLOv3 is the utilization of anchor boxes with diverse scales and aspect ratios. Unlike YOLOv2, where anchor boxes were of uniform size, YOLOv3 employs scaled anchor boxes with varied aspect ratios, facilitating improved detection of objects with varying sizes and shapes.

YOLOv3 also introduces "feature pyramid networks" (FPN), a CNN architecture designed to detect objects across multiple scales. FPNs construct a feature map pyramid, with each level enabling object detection at different scales, thereby enhancing detection performance, especially for small objects observed at varying scales.

Furthermore, YOLOv3 exhibits enhanced versatility in handling a broader range of object sizes and aspect ratios, while also delivering superior accuracy and stability compared to its predecessors.



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

Figure 1.25: Comparison of the results obtained by YOLOv3 [8]

YOLOv4 : YOLOv4, revealed in 2020 by Bochkovskiy et al., marks the fourth version of YOLO, aimed at refining the capabilities of YOLOv3. A key enhancement in YOLOv4 lies in the adoption of the CSPNet (Cross Stage Partial Network) architecture, a variant of ResNet tailored specifically for object detection tasks. With a relatively shallow structure comprising 54 convolutional layers, CSPNet achieves state-of-the-art performance across various object detection benchmarks.

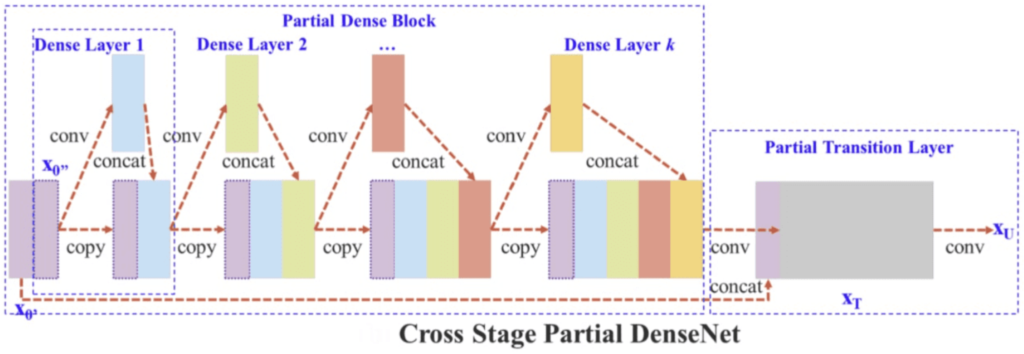


Figure 1.26: Architecture of CSPNet

Both YOLOv3 and YOLOv4 leverage anchor boxes with diverse scales and aspect ratios to better align with the size and shape of detected objects. YOLOv4 introduces a novel approach to anchor box generation termed "k-means clustering." This method involves clustering ground truth bounding boxes to determine anchor box centroids, enhancing alignment with detected objects' characteristics.

While employing a similar loss function to train the model, YOLOv4 introduces the "GHM loss," a variant of the focal loss function tailored to address imbalanced datasets, thereby enhancing model performance. Besides, YOLOv4 refines the architecture of the Feature Pyramid Networks (FPNs) utilized in YOLOv3.

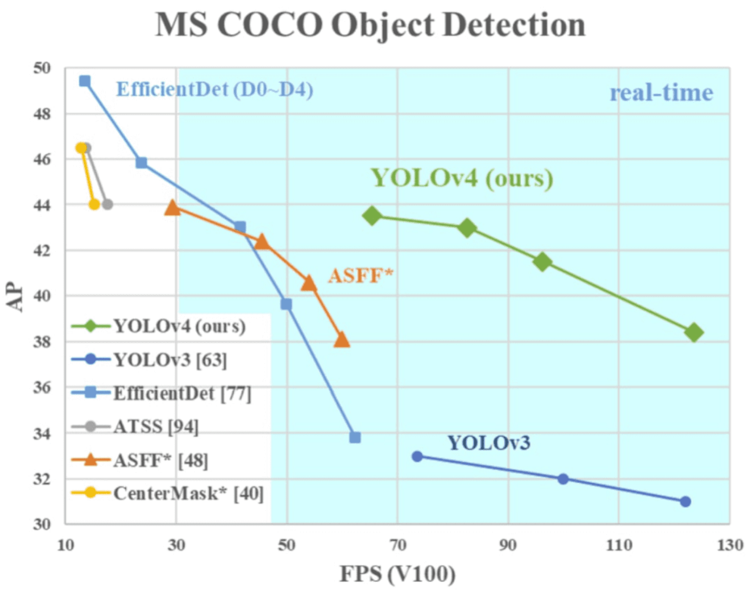


Figure 1.27: Comparison of the proposed YOLOv4 and other object detectors [8]

YOLOv5 : In 2020, YOLOv5 emerged as an open-source project developed by the original

YOLO algorithm team, now maintained by Ultralytics. It inherits the legacy of its predecessors while incorporating a slew of novel features and enhancements.

Diverging from its predecessor, YOLOv5 adopts the more intricate EfficientDet architecture, derived from the EfficientNet network architecture. This architectural upgrade empowers YOLOv5 to achieve heightened accuracy and enhanced generalization across a broader spectrum of object categories.

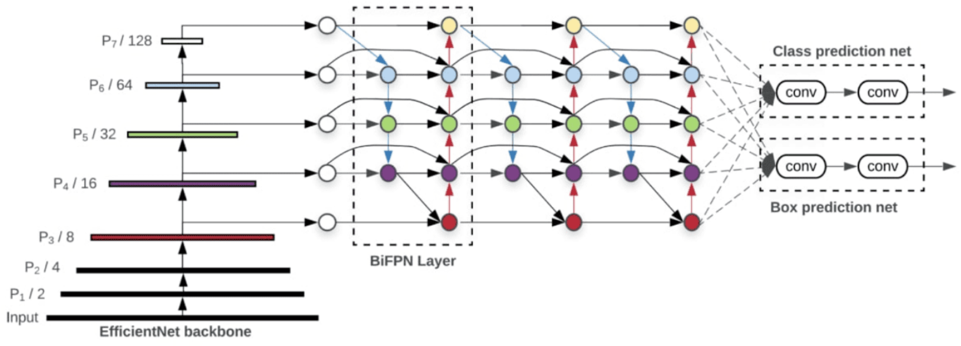


Figure 1.28: Architecture of the EfficientDet model

While YOLO relied on the PASCAL VOC dataset comprising 20 object categories for training, YOLOv5 harnesses a more expansive and diverse dataset dubbed D5, encompassing a total of 600 object categories.

YOLOv5 pioneers "dynamic anchor boxes" for anchor box generation, leveraging a clustering algorithm to group ground truth bounding boxes into clusters and utilizing their centroids as anchor boxes. This refinement ensures a closer alignment between anchor boxes and the detected objects' size and shape.

Introducing "spatial pyramid pooling" (SPP), YOLOv5 implements a pooling layer to diminish the spatial resolution of feature maps, enhancing detection performance on small objects by enabling multi-scale object observation. While YOLOv4 also employs SPP, YOLOv5 introduces several SPP architecture enhancements for superior outcomes.

Although YOLOv4 and YOLOv5 share a similar loss function for model training, YOLOv5 introduces "CIoU loss," a variant of the IoU loss function tailored to enhance model performance on imbalanced datasets.

YOLOv6 : In 2022, Li et al. [30] introduced YOLOv6 as an advancement over its predecessors. The adoption of a different CNN architecture is prominent among the disparities between YOLOv5 and YOLOv6. YOLOv6 opts for a variant of the EfficientNet architecture known as EfficientNet-L2. This architecture, distinct from EfficientDet utilized in YOLOv5, boasts fewer parameters and heightened computational efficiency while still achieving top-tier results across various object detection benchmarks. The schematic representation of the YOLOv6 model is depicted below.

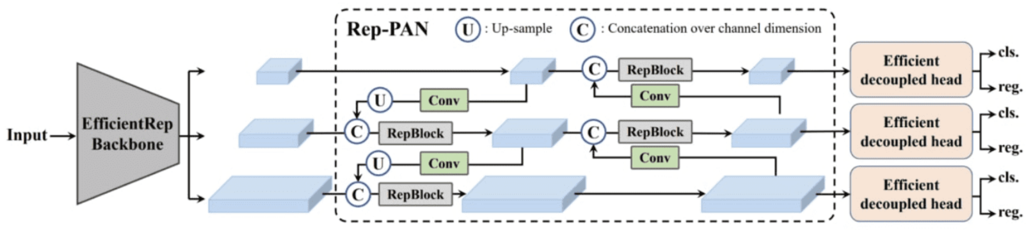
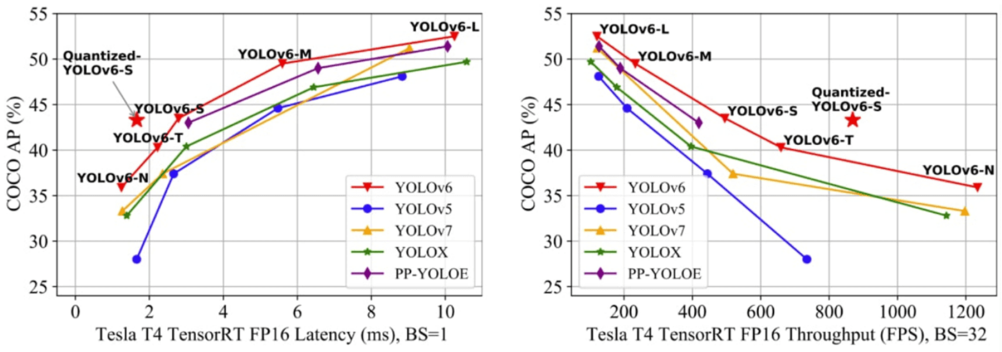


Figure 1.29: Overview of YOLOv6

Moreover, YOLOv6 pioneers a new method for anchor box generation termed "dense anchor boxes".



Comparison of state-of-the-art efficient object detectors. Both latency and throughput (at a batch size of 32) are given for a handy reference. All models are test with TensorRT 7 except that the quantized model is with TensorRT 8.

Figure 1.30: Results obtained by YOLOv6 compared to other state-of-the-art methods [8]

YOLOv7 : YOLOv7 introduces several enhancements, one notable improvement is its utilization of anchor boxes. Anchor boxes, comprising predefined boxes with varying aspect ratios, play a pivotal role in detecting objects of diverse shapes. YOLOv7 integrates nine anchor boxes, facilitating the detection of a broader spectrum of object shapes and sizes compared to earlier versions. This, in turn, aids in minimizing false positives.

A prominent advancement in YOLOv7 is the incorporation of a novel loss function dubbed "focal loss". Unlike previous iterations that employed a standard cross-entropy loss function, which often struggled with detecting small objects, focal loss addresses this challenge by down-weighting the loss for well-classified examples and prioritizing hard examples—objects that are inherently difficult to detect.

Moreover, YOLOv7 boasts a higher image resolution. Processing images at a resolution of 608 by 608 pixels surpasses the 416 by 416 resolution employed in YOLOv3. This heightened resolution enables YOLOv7 to detect smaller objects more effectively and enhances overall accuracy.

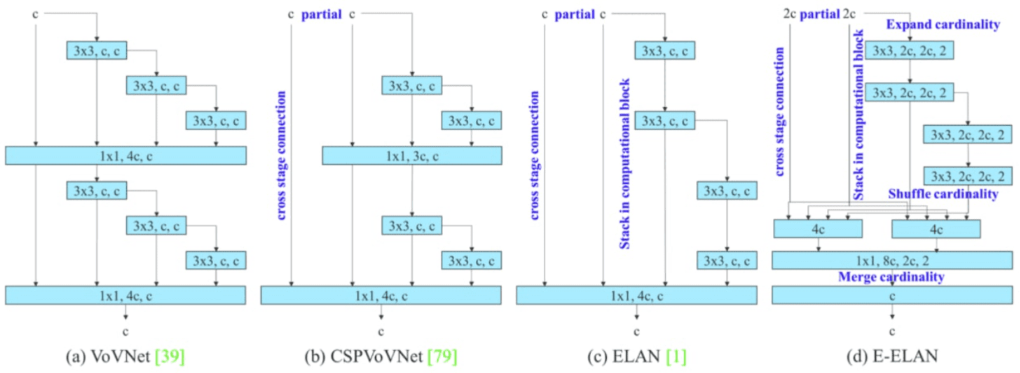


Figure 1.31: Change in the layer aggregation scheme of YOLOv7 for efficient object feature learning

Another noteworthy feature of YOLOv7 is its enhanced speed. With the capability to process images at a rate of 155 frames per second, YOLOv7 significantly outpaces other contemporary real-time object detection algorithms. Even the original baseline YOLO model operated at a maximum rate of 45 frames per second. This swift processing makes YOLOv7 well-suited for critical real-time applications such as surveillance and autonomous driving, where rapid processing is imperative.

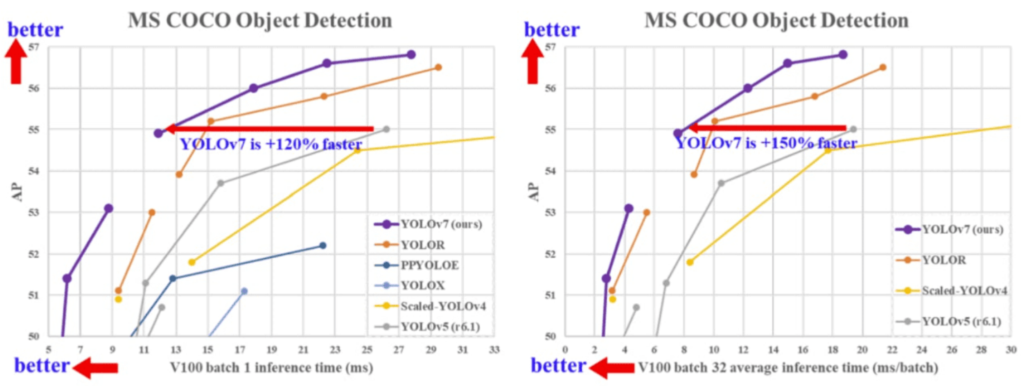
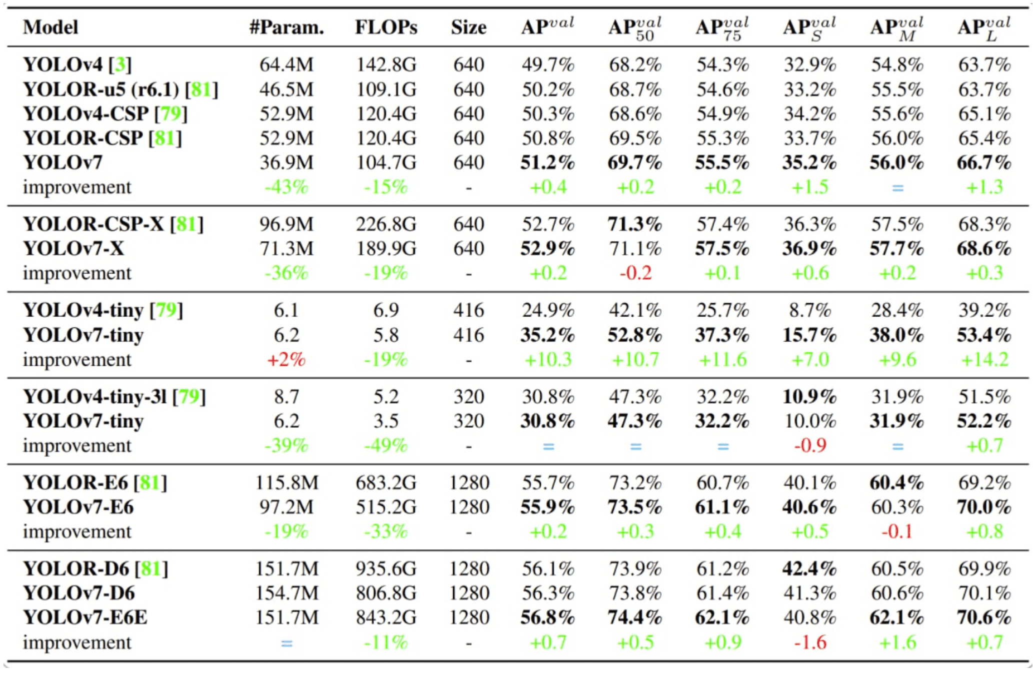


Figure 1.32: Comparison in performance and inference speed of YOLOv7 with contemporary state-of-the-art real-time object detectors [8]

In terms of accuracy, YOLOv7 exhibits commendable performance compared to other object detection algorithms. It achieves an average precision of **37.2%** at an IoU threshold of **0.5** on the widely used COCO dataset, placing it on par with other state-of-the-art object detection algorithms.



Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%	54.3%	32.9%	54.8%	63.7%
YOLOR-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%	54.6%	33.2%	55.5%	63.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%	54.9%	34.2%	55.6%	65.1%
YOLOR-CSP [81]	52.9M	120.4G	640	50.8%	69.5%	55.3%	33.7%	56.0%	65.4%
YOLOv7	36.9M	104.7G	640	**51.2%**	**69.7%**	**55.5%**	**35.2%**	**56.0%**	**66.7%**
improvement	-43%	-15%	-	+0.4	+0.2	+0.2	+1.5	=	+1.3
YOLOR-CSP-X [81]	96.9M	226.8G	640	52.7%	**71.3%**	57.4%	36.3%	57.5%	68.3%
YOLOv7-X	71.3M	189.9G	640	**52.9%**	71.1%	**57.5%**	**36.9%**	**57.7%**	**68.6%**
improvement	-36%	-19%	-	+0.2	-0.2	+0.1	+0.6	+0.2	+0.3
YOLOv4-tiny [79]	6.1	6.9	416	24.9%	42.1%	25.7%	8.7%	28.4%	39.2%
YOLOv7-tiny	6.2	5.8	416	**35.2%**	**52.8%**	**37.3%**	**15.7%**	**38.0%**	**53.4%**
improvement	+2%	-19%	-	+10.3	+10.7	+11.6	+7.0	+9.6	+14.2
YOLOv4-tiny-3l [79]	8.7	5.2	320	30.8%	47.3%	32.2%	**10.9%**	31.9%	51.5%
YOLOv7-tiny	6.2	3.5	320	**30.8%**	**47.3%**	**32.2%**	10.0%	**31.9%**	**52.2%**
improvement	-39%	-49%	-	=	=	=	-0.9	=	+0.7
YOLOR-E6 [81]	115.8M	683.2G	1280	55.7%	73.2%	60.7%	40.1%	**60.4%**	69.2%
YOLOv7-E6	97.2M	515.2G	1280	**55.9%**	**73.5%**	**61.1%**	**40.6%**	60.3%	**70.0%**
improvement	-19%	-33%	-	+0.2	+0.3	+0.4	+0.5	-0.1	+0.8
YOLOR-D6 [81]	151.7M	935.6G	1280	56.1%	73.9%	61.2%	**42.4%**	60.5%	69.9%
YOLOv7-D6	154.7M	806.8G	1280	56.3%	73.8%	61.4%	41.3%	60.6%	70.1%
YOLOv7-E6E	151.7M	843.2G	1280	**56.8%**	**74.4%**	**62.1%**	40.8%	**62.1%**	**70.6%**
improvement	=	-11%	-	+0.7	+0.5	+0.9	-1.6	+1.6	+0.7

Figure 1.33: The quantitative comparison of the performances [8]

However, it is worth noting that YOLOv7 falls short of the accuracy attained by two-stage detectors like Faster R-CNN and Mask R-CNN, which typically yield higher average precision on the COCO dataset albeit at the expense of longer inference times.

Despite its prowess, YOLOv7 possesses certain limitations:

1 - Challenges in detecting small objects : Similar to many object detection algorithms, YOLOv7 may struggle to accurately detect small objects, particularly in crowded scenes or when objects are distant from the camera.

2 - Difficulty in detecting objects at different scales : YOLOv7 may encounter difficulties in detecting objects that deviate significantly in size from other objects within the scene, whether exceedingly large or minuscule.

3 - Sensitivity to environmental conditions : YOLOv7's performance can be impacted by variations in lighting or other environmental factors, potentially posing challenges in scenarios where lighting conditions vary.

4 - Computational intensity : YOLOv7's computational demands may pose challenges for real-time implementation on resource-constrained devices such as smartphones or other edge devices.

YOLOv8

YOLOv8 is the latest version in the YOLO series of real-time object detectors, offering cutting-edge performance in terms of accuracy and speed. Building upon the advancements of previous YOLO versions, YOLOv8 introduces new features and optimizations that make it an ideal choice for various object detection tasks in a wide range of applications.

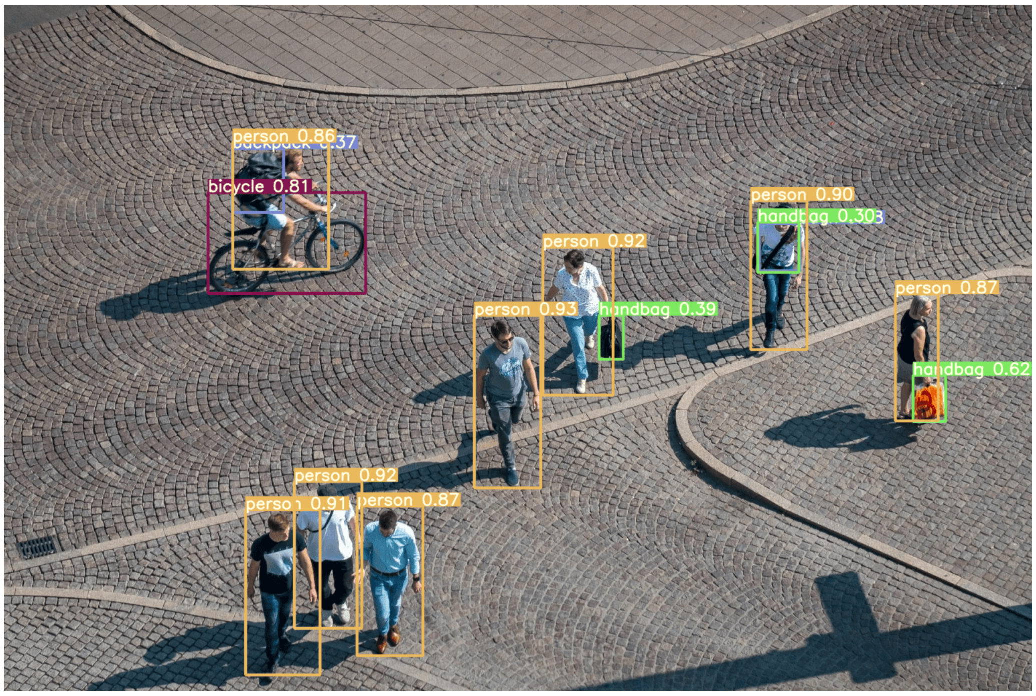


Figure 1.34: YOLOv8

YOLOv8 introduces significant enhancements across multiple fronts, including mosaic data augmentation, anchor-free detection, a C2f module, a decoupled head, and a refined loss function, all aimed at boosting performance and efficiency. Mosaic Data Augmentation, akin to YOLOv4, amalgamates four images to provide rich contextual information, with this strategy being halted in the final ten training epochs to optimize model efficacy. Anchor-free detection represents a pivotal shift from anchor-based methods, directly predicting an object's midpoint to accelerate Non-max Suppression (NMS) and improve generalization.

The integration of a C2f module in the model's backbone replaces the conventional C3 module, streamlining computational costs and enhancing gradient flow by concatenating outputs from all bottleneck modules. Moreover, the adoption of a Decoupled Head paradigm separates classification and regression tasks, alleviating potential loss misalignments. This is complemented by a task alignment score, derived from the Intersection over Union (IoU) metric, guiding the model in selecting top-k positive samples and computing classification and regression losses using Binary Cross-Entropy, Complete IoU (CIoU), and Distributional Focal Loss (DFL). While BCE loss addresses label prediction disparities, CIoU loss evaluates bounding box accuracy, and DFL optimizes boundary distribution, collectively enhancing YOLOv8's object detection capabilities [31].

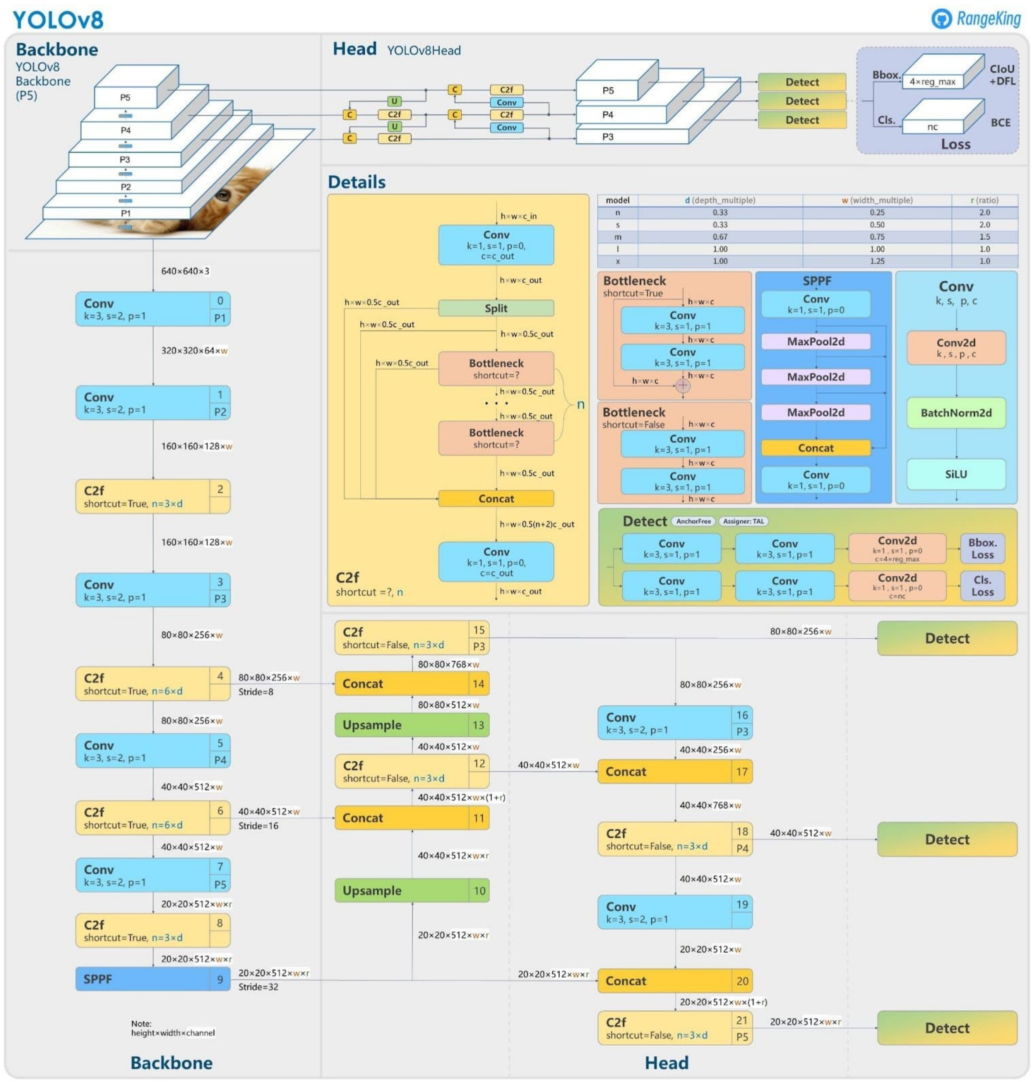


Figure 1.35: YOLOv8 architecture [9]

Performance-wise, YOLOv8 demonstrates a significant **1.2%** increase in average precision compared to its predecessor, YOLOv7, while simultaneously reducing the model's weight file size by **80.6 M**, rendering it more deployable in resource-constrained environments.

Additionally, Ultralytics YOLOv8 offers a diverse range of modes tailored to cover the entire machine-learning model lifecycle. These modes include Train, Val, Predict, Export, Track, and Benchmark modes, each serving specific purposes and providing flexibility and efficiency across various tasks and applications.

Train mode facilitates model training on custom datasets, while Val mode validates model performance post-training. Predict mode enables predictions on new data, while Export mode prepares the model for deployment in various formats. Track mode extends object detection models into real-time tracking applications, while Benchmark mode analyzes model speed and accuracy in different deployment environments [32].

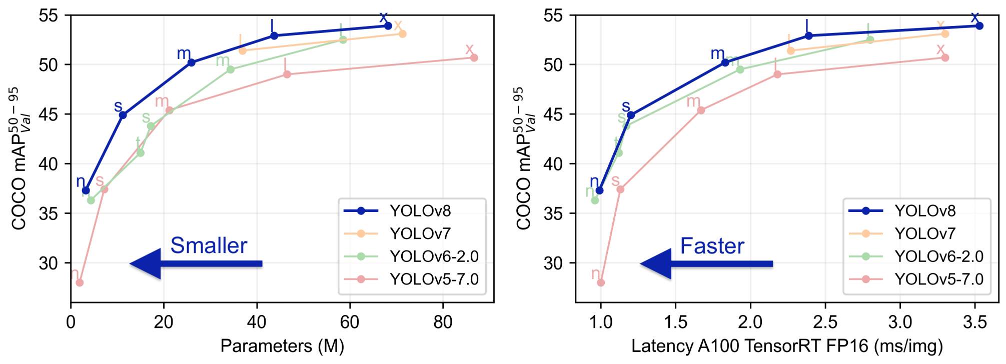


Figure 1.36: Comparison in parameters and latency of YOLOv8 with other versions [7]

YOLOv8 boasts versatility, finding applications across various real-world scenarios.

1- People Counting: Retail establishments leverage the model to monitor real-time foot traffic, gauge queue lengths, and optimize store layouts for enhanced customer experiences.

2- Sports Analytics: Analysts capitalize on the model to track player movements on sports fields, extracting valuable insights into team dynamics and player performance, and contributing to informed decision-making in sports management.

3- Inventory Management: The object detection capabilities facilitate inventory monitoring, enabling businesses to maintain optimal stock levels, anticipate demand trends, and streamline inventory operations for efficient supply chain management.

4- Autonomous Vehicles: Object detection models play a crucial role in the development of autonomous driving systems, empowering vehicles to recognize and respond to their surroundings accurately, and ensuring safe navigation on roads and highways.

Through its adaptability and precision, YOLOv8 proves instrumental in addressing diverse challenges across industries, paving the way for innovation and efficiency in various applications.

Chapter 2

Database Creation

2.1 Introduction

This chapter delves into the pivotal aspect of database creation, a foundational step in developing robust systems. It begins with exploring various datasets tailored to diverse computer vision tasks, providing essential insights into their significance and applicability across different domains. Subsequently, it delves into the meticulous process of designing the database, annotating it, and focusing on the compilation, organization, and augmentation of data essential for training detection models. The chapter delves into the intricacies of data annotation, elucidating the methodologies and standards adopted for accurate labeling, crucial for developing and evaluating detection models.

2.2 Datasets Overview

2.2.1 Ultralytics

In the realm of YOLOv8 object detection, Ultralytics is recognized as the organization responsible for developing and maintaining the YOLO model series. They have made substantial contributions to enhancing model architecture, refining training methodologies, and promoting open-source implementations. Ultralytics hosts repositories on GitHub, providing researchers and developers with access to the latest iterations of YOLO models, pre-trained weights, and tools essential for both training and deploying these models.

Ultralytics also offers extensive support for a wide array of datasets tailored to diverse computer vision tasks, encompassing detection, instance segmentation, pose estimation, classification, and multi-object tracking. These datasets are meticulously curated to meet specific task requirements and application domains. For instance, in object detection, datasets like COCO, LVIS, and OpenImagesV7 offer comprehensive annotations across various object categories, enabling robust detection algorithms. In instance segmentation, datasets such as COCO, Crack-seg, and Package-seg provide pixel-level annotations, facilitating precise object delineation in complex scenes.

For tasks like pose estimation, datasets like COCO and Tiger-pose annotate key points crucial for accurately determining object poses relative to the camera or world coordinate system. In image classification, datasets including CIFAR-10, CIFAR-100, and ImageNet provide vast collections of images spanning diverse categories, supporting effective categorization of visual content. Additionally, specialized datasets such as DOTAv2 and Argoverse cater to specific needs such as oriented bounding boxes and multi-object tracking, tailored for applications in aerial imagery analysis and urban environment monitoring.

As Ultralytics datasets lacked any records about speed bumps and road defects, we explored online repositories for suitable datasets. Our search led us to Roboflow, where we identified two datasets that met our requirements.

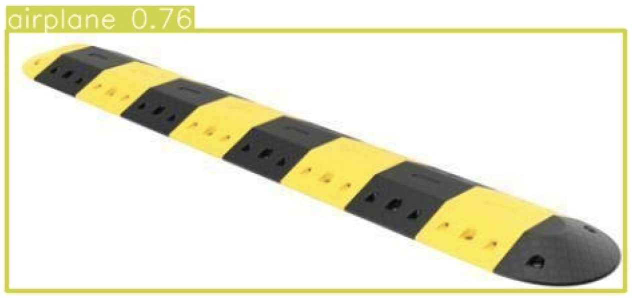


Figure 2.1: Model Fails to Detect Speed Bumps and Road Defects Without Specific Training

2.2.2 Roboflow Datasets

Roboflow is a platform designed to streamline the process of building, managing, and deploying computer vision models, providing tools and services that help developers and data scientists work more efficiently with image data for tasks like object detection, classification, and segmentation. It offers data annotation tools for labeling images, facilitating the creation of annotated datasets necessary for model training, and allowing for effective dataset management, including organizing, versioning, and sharing. Roboflow also provides preprocessing and augmentation options to clean and prepare data, enhancing model robustness. Users can train and deploy models directly on the platform, with integration support for popular machine learning frameworks such as TensorFlow, PyTorch, and YOLO, making dataset and model export seamless.

2.3 Designing the Database

To effectively train our model, we required a database containing both speed bumps and road defects, providing essential labeled data for learning and performance enhancement. This database is a crucial resource, offering diverse annotated images or videos that act as training data for the detection model. Through exposure to this varied dataset during training, the model learns to identify pertinent features and patterns associated with the objects of interest, facilitating its ability to recognize them during inference. Additionally, the comprehensive nature of the database aids in generalization, enabling the model to apply its learning to unseen data across different scenarios, lighting conditions, and environments. Furthermore, the database serves as a means for evaluating the model's performance, with separate validation or test sets allowing researchers to gauge its effectiveness on new data and identify areas for improvement. This iterative training, evaluation, and refinement process continues until the desired performance level is achieved.

Our primary objective is compiling a dataset containing images specifically tailored to detect speed bumps and road anomalies. To achieve this goal, it is imperative to adhere to precise specifications for the dataset aimed at training our model, images should be in standard formats to ensure compatibility with commonly used image processing libraries and tools for dataset preparation and model training. The dataset should cover diverse scenarios, including variations in lighting conditions, weather, road surfaces, and types of speed bumps and road defects. This diversity is essential for effective generalization across different real-world conditions. Furthermore, each image in the dataset must be meticulously annotated with bounding boxes around speed bumps and road defects to delineate the boundaries of each object for precise detection accurately.

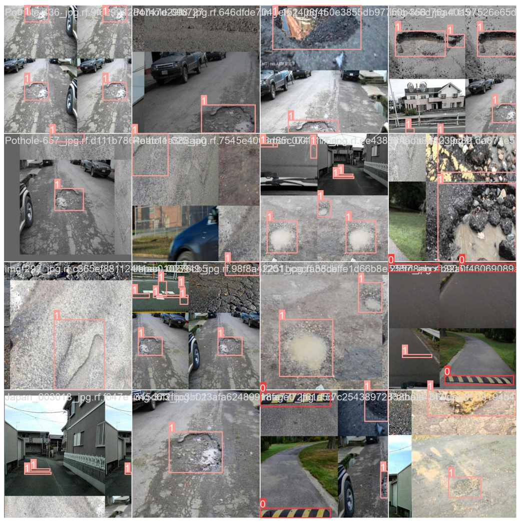


Figure 2.2: Batch of Database

The database is organized into two separate datasets, sourced from different locations found on Roboflow. The first dataset is specifically dedicated to speed bumps, with the following distribution: **86%** for the training set, consisting of **1927** images, **10%** for the validation set, comprising **217** images, and **4%** for the test set, containing **96** images. Images in this dataset underwent preprocessing, including auto-orientation and resizing to achieve a resolution of **640x640** pixels. Each training example was augmented with three outputs per image, and horizontal flipping was applied. The second dataset for road defects is divided into a training set comprising **80%** of the data with **3033** images, a validation set comprising **13%** with **491** images, and a test set comprising **7%** with **246** images. Preprocessing steps applied to the images include auto-orientation to ensure correct image alignment, resizing all images to a **640x640** pixel resolution by stretching, and modifying classes where seven classes were remapped and one was dropped. Each training example generates three outputs for augmentation, and horizontal flipping is applied to increase the dataset's variability and improve the model's robustness.

Overall, the entire dataset is partitioned into approximately **83%** for the training set, encompassing a total of **4960** images, **12%** for the validation set, comprising **708** images, and **5%** for the test set, containing **342** images. Notably, the road defects dataset underwent a modification of classes, simplifying the original three classes—' Drain Hole', 'Pothole', and 'Sewer Cover'—into a single class labeled 'road defect'. This decision was made after extensive experimentation with various datasets containing more detailed classifications of road defects. The results from these experiments were sub-optimal, largely due to the complexity introduced by the multiple classes. Consequently, the focus shifted to potholes, identified as the most dangerous defects compared to other types like cracks. This strategic focus aimed to enhance the model's performance and reliability in detecting the most critical road defects, thereby improving safety and usability.

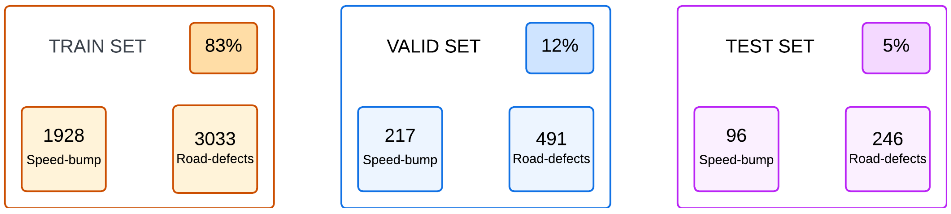
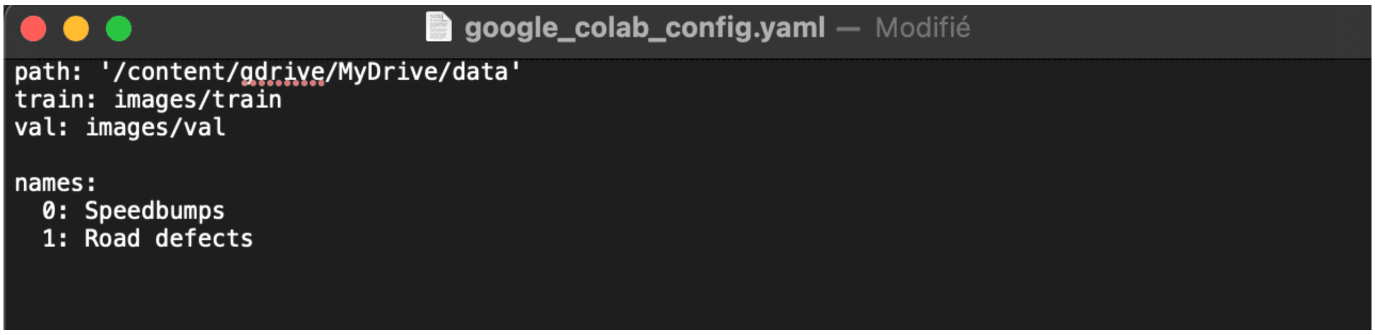


Figure 2.3: Dataset Overview Chart

To organize the data, we used a YAML file, a plain-text data serialization format that is both human-readable and machine-friendly. It is commonly used to structure data in a way that is easy to understand and edit manually, making it popular for configuration files, data storage, and communication protocols.

Our dataset resides within a specified directory on Google Drive, where we meticulously arrange both training and validation images alongside their corresponding labels. Adhering to the YOLOv8 annotation format, we assign specific labels to different classes. Notably, speed bumps are marked with the value 0, while road defects are assigned the value 1. This labeling convention ensures uniformity across annotations, supporting seamless integration with our model training pipeline. By adopting this systematic approach, we ensure that our data pipeline is coherent and transparent, promoting efficient model training and evaluation processes.



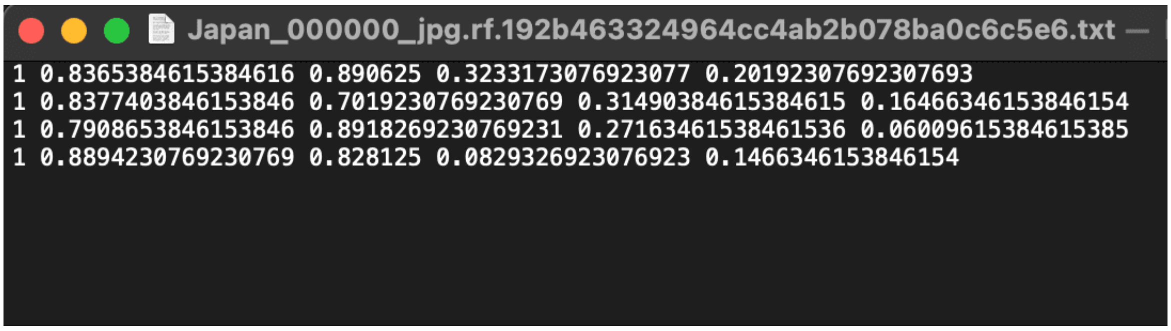
```
path: '/content/gdrive/MyDrive/data'
train: images/train
val: images/val

names:
  0: Speedbumps
  1: Road defects
```

Figure 2.4: Dataset organization

2.4 Data Annotation

The process of annotating collected data involves assigning tags or metadata to each data element to make them informative and usable for machine learning. In the specific case of our data collection for the YOLOv8 model, annotation follows a specific format for each image. Each annotation consists of five numerical values representing different attributes of the bounding box of the object. The format is as follows: [**class**] [**center_x**] [**center_y**] [**width**] [**height**]. Here, [**class**] represents the class label of the object being annotated. The subsequent four values [**center_x**]**,** [**center_y**]**,** [**width**], and [**height**] denote the normalized coordinates and dimensions of the bounding box relative to the image size. Specifically, [**center_x**] and [**center_y**] indicate the normalized coordinates of the center of the bounding box, while [**width**] and [height] represent the normalized width and height of the bounding box, respectively. This annotation format ensures compatibility with our model, allowing for accurate training and detection of objects within the images.



```
1 0.8365384615384616 0.890625 0.3233173076923077 0.20192307692307693
1 0.8377403846153846 0.7019230769230769 0.31490384615384615 0.16466346153846154
1 0.7908653846153846 0.8918269230769231 0.27163461538461536 0.06009615384615385
1 0.8894230769230769 0.828125 0.0829326923076923 0.1466346153846154
```

Figure 2.5: Bounding Box annotation format example

After identifying our data, we utilized the online application CVAT for the critical task of annotation. This tool enabled us to delineate objects of interest—specifically, speed bumps and road defects—using rectangular bounding boxes. Each object underwent manual annotation to ensure the accuracy and correctness necessary for effective model training. This meticulous approach was crucial given the limited reliability of automatic algorithms for this task.

Following annotation, we uploaded the annotated images in the YOLOv8 format, an option available within the CVAT application. This format choice was essential as it seamlessly integrated with our model's requirements, facilitating smooth incorporation into our training pipeline. Despite its challenges, this rigorous annotation process ensured the creation of high-quality data, leading to highly effective model training.

This workflow underscores that a data scientist's role extends beyond coding; meticulous data preparation is essential before software development begins. Accurate annotation serves as the foundation of model development by providing ground truth labels that guide the learning process. Consistent and precise annotations maintain uniformity across the dataset, mitigating biases during training and evaluation. They also enable effective model assessment, allowing researchers to gauge performance and identify areas for improvement.

In the context of YOLOv8, precise annotation is paramount for optimizing model training and performance. The accurate placement and classification of bounding boxes instruct the model to detect and classify objects accurately, enhancing its ability to generalize to new data. Ensuring rigorous and consistent annotation practices is critical for maximizing the efficacy of YOLOv8 models across various real-world applications.

2.5 Conclusion

In conclusion, establishing robust databases is crucial for developing proficient computer vision systems. This chapter has covered key aspects such as exploring diverse datasets designed for various computer vision tasks, underscoring their significance across different fields. It has also detailed the meticulous process of database design, including data gathering, organization, and enhancement essential for training detection models. Moreover, the chapter has elucidated the complexities of data annotation, highlighting its vital role in accurately labeling data to develop and assess detection models.

Platforms like Ultralytics and Roboflow play pivotal roles in supporting a broad spectrum of computer vision tasks, ensuring comprehensive training and evaluation of detection algorithms. Key datasets such as COCO, LVIS, and those available on Roboflow have provided annotated data crucial for tasks like object detection and instance segmentation. Despite challenges such as uneven dataset distributions and the need for precise annotation, these efforts emphasize the importance of meticulously curating datasets to achieve robust model performance.

Overall, the structured process of creating databases ensures that our YOLOv8 model, specialized in detecting speed bumps and road defects, is trained with high-quality data. This foundational step not only optimizes model training but also enhances its practical use in real-world situations, improving safety and usability across various applications.

Chapter 3

Methodology

3.1 Introduction

This chapter explores our real-time object detection system designed to identify speed bumps and road anomalies. It thoroughly details the customization process of the YOLOv8 model architecture, specifically focusing on adaptations made to its backbone, head, and various layers. The chapter comprehensively discusses diverse training strategies, including both training from scratch and leveraging pre-trained weights, accompanied by performance metrics that highlight the advantages of transfer learning in our context. Moreover, it emphasizes the critical role of data augmentation using the Albumentations library and rigorous data validation procedures, essential for enhancing the model's robustness and accuracy in real-world scenarios.

3.2 Data Collection and Preprocessing

The process initiates with a vehicle outfitted with an embedded camera navigating the road, where it records real-time images of the road surface. As the vehicle progresses, the embedded camera consistently captures images, providing a continuous visual feed of the surroundings. These collected images serve as the input for our model, which is then deployed in real-time. The trained model actively analyzes the live video feed, swiftly detecting the presence of speed bumps or road anomalies in real-time based on the acquired features and characteristics. Upon identifying an object, an integrated alerting system within the vehicle promptly notifies the driver of the obstacle's presence on the road, thereby bolstering road safety and driver awareness.

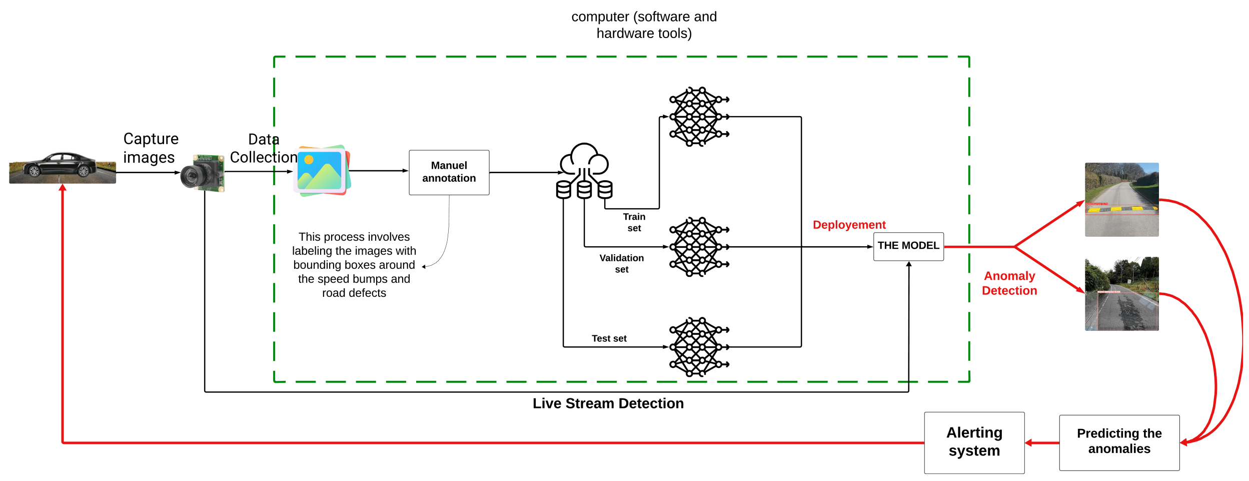


Figure 3.1: Our Real-time Object Detection System

it is important to consider the environmental conditions necessary for effective model deployment. Our model is optimized to perform reliably under specific conditions conducive to its functionality. Firstly, the model operates optimally during daytime conditions since our embedded camera utilizes standard imaging technology, which may experience limitations in low-light environments. Moreover, for real-time processing, it is essential to ensure that the vehicle's speed remains within a reasonable range to enable timely detection and response to road anomalies. Additionally, factors such as weather conditions, road surface quality, and camera positioning can also influence the model's performance and effectiveness in detecting speed bumps and road defects accurately.

3.3 Customization of the Adopted Model Architecture

3.3.1 Model Summary

The YOLOv8 model has been tailored to meet the requirements of our specific detection task, focusing on **2** classes instead of the default **80**. The model summary provides a detailed breakdown of its architecture, highlighting key components such as convolutional layers (Conv), C2f blocks, the upsampling module (Upsample), and detection layers (Detect). For instance, the model starts with a convolutional layer featuring **3** input channels and **32** output channels, followed by another convolutional layer increasing the channels to **64**. This is succeeded by a C2f block, maintaining **64** input and output channels, demonstrating the model's depth and complexity. As the architecture progresses, layers such as a convolutional layer with **64** input channels and **128** output channels, and C2f blocks further transforming **128** input channels into **128** output channels, highlight the hierarchical feature extraction. The inclusion of upsampling layers and concatenation operations, like combining **512** channels from an upsampling layer with **256** channels from a previous layer to form **768** channels, exemplifies the model's intricate design for feature fusion and resolution enhancement. Finally, the detect layer, integrating features from multiple scales, concludes the model with tailored detection capabilities. Additionally, **SiLU** (Sigmoid Linear Unit), an activation function used in various layers, enhances non-linearity and gradient smoothness, contributing to efficient model training and improved performance.

$$\text{SiLU}(x) = x \cdot \sigma(x) \tag{3.1}$$

Where :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

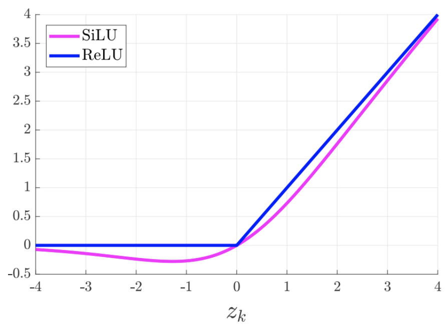


Figure 3.2: SiLU: Sigmoid-weighted Linear Unit

3.3.2 Model Statistics

The model consists of **225** layers, **11.1** million parameters, and operates with a computational complexity of **28.6** GFLOPs, optimizing for both accuracy and efficiency in object detection tasks. These statistics serve as critical indicators of the model's intricate complexity and computational capability. They are essential for evaluating the necessary resources and estimating the potential training performance of the model.

3.3.3 Detailed Layer Analysis and Functional Components

Understanding the architecture of our model is crucial for appreciating its functionality and performance. The model unfolds layer by layer, starting with initial convolutional layers (Conv) that process input images and extract hierarchical features, capturing both low-level details and high-level semantic information essential for accurate object detection. The model leverages convolutional blocks, particularly C2f blocks, to efficiently learn hierarchical features, enhancing its ability to discern between different object classes and background clutter.

The SPPF (Spatial Pyramid Pooling with Feature Fusion) module enriches the model's representation of objects at various scales and aspect ratios, enabling robust detection across diverse scenarios. Additionally, it enhances the model's ability to handle objects of varying sizes, such as speed bumps and road defects, by incorporating multi-scale contextual information into the detection process.

Upsampling operations (Upsample) increase the spatial resolution of feature maps, facilitating finer-grained predictions and improving localization accuracy. They play a pivotal role in refining the model's understanding of object boundaries and shapes, contributing to more precise detection outcomes.

Concatenation operations (Concat) fuse features from different network layers, enabling the model to comprehensively capture diverse object characteristics and enhance its discriminative power and detection accuracy.

3.4 Model Training

3.4.1 Google Colaboratory

Google Colaboratory, known as Google Colab, is a cloud-based platform offered by Google for writing and executing Python code within a Jupyter Notebook environment. It has garnered popularity among data scientists and researchers owing to its user-friendly interface and access to robust computational resources. One notable feature of Google Colab is its provision of free Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) resources [33].

Training and evaluating YOLOv8 models can be done through various platforms, but using Google Colab offers distinct advantages: it operates in the cloud, eliminating the need for complex environment setups, and allowing immediate model training by simply creating an account and writing code. Colab facilitates experimentation with different values and settings through parameter input forms. Its seamless integration with Google Drive simplifies data storage, access, and management, enabling direct storage and retrieval of datasets and models. Colab's support for markdown enhances documentation within notebooks, making it easier to organize and present work. Additionally, users can schedule notebooks to run at specified times, automating tasks and processes. The platform also supports adding extra functionality through third-party extensions and interactive widgets, enhancing the development experience.

We chose to train our model using Google Colab's **GPU T4** instead of the default CPU because of its substantial advantages that improve both training and inference processes. The **T4 GPU** significantly accelerates computation compared to CPUs or lower-end GPUs, making it essential for efficiently handling large datasets and complex models like YOLOv8. This capability reduces training times and enables faster experimentation and iteration. Its generous memory capacity surpasses that of CPUs, ensuring efficient processing of extensive image data and annotations without encountering memory constraints.

T4 GPUs excel in parallel processing, particularly for matrix operations and deep learning computations, thereby enhancing the performance of YOLOv8 in terms of training speed and inference accuracy. With support for CUDA, NVIDIA's parallel computing platform, T4 GPUs further enhance performance by leveraging GPU-accelerated computations crucial for compute-intensive tasks in deep learning. Moreover, Google Colab offers free access to T4 GPUs, making it a cost-effective option for individuals and small teams, democratizing access to powerful hardware for conducting deep learning experiments and deploying object detection models like YOLOv8.

3.4.2 Training a Model from Scratch

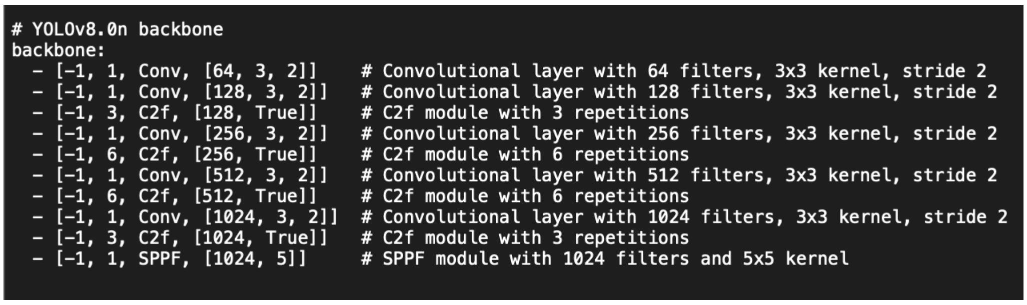
Configuring a YOLOv8 model via a YAML file entails meticulously crafting the model's architecture and parameters from scratch, devoid of pre-trained weights. This intensive process demands precise delineation of layer configurations, hyperparameters, and structural specifics. Training commences from ground zero, requiring a substantial amount of annotated data and extended training time to effectively learn object features.

Initially, we implemented the **"YOLOv8n.yaml"** model, constructed with the same architecture as the pre-trained model but without the pre-trained weights. This approach allows for a more meaningful comparison, especially considering that the architecture derived from our **"yaml"** file represents the most optimal parameters. Thus, using an alternative architecture would likely yield inferior results. Therefore, we opted for consistency by employing the same architecture, differing only in the presence or absence of pre-trained weights.

Backbone :

The backbone architecture of the YOLOv8 model outlines a sequence of layers, each meticulously defined by its **"source, repetition count, layer type, and parameters"** : **"source"** refers to the reference point or source layer from which the current layer receives its input. It specifies where the data flow originates within the neural network architecture, often denoted by a numerical index indicating the position of the preceding layer. The **"repetition count"** indicates how many times a particular layer or module is repeated within the architecture. This repetition is essential for capturing features at different scales or resolutions, allowing the model to extract increasingly complex information from the input data.

Initially, convolutional layers with diverse filter counts and a 3x3 kernel size are utilized to reduce the input image's dimensions by a factor of two, extracting fundamental features. Subsequently, specialized C2f modules, configured with specific settings, are iterated multiple times to capture features across varying resolutions. Progressively, the filter count within the layers increases, enabling the extraction of more intricate details from the input. Finally, an SPPF module is applied to capture features at different scales, leveraging 1024 filters and a 5x5 kernel size for comprehensive feature representation.



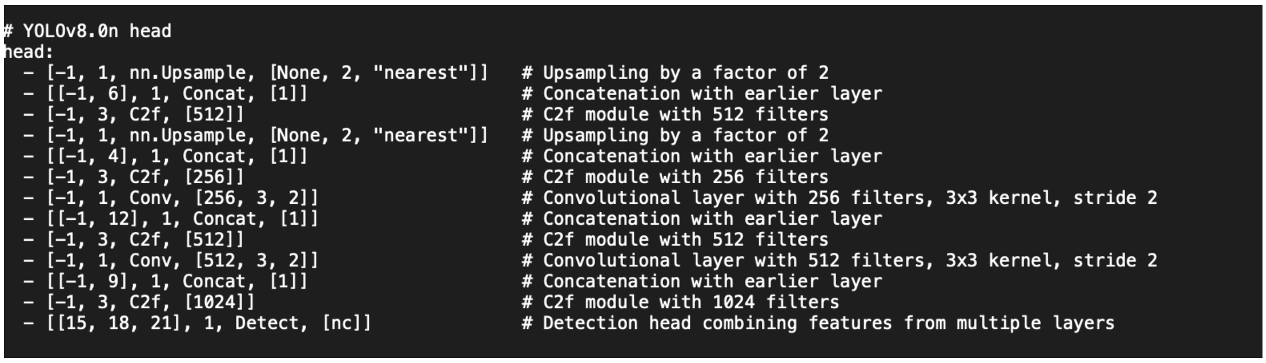
```
# YOLOv8.0n backbone
backbone:
  - [-1, 1, Conv, [64, 3, 2]]    # Convolutional layer with 64 filters, 3x3 kernel, stride 2
  - [-1, 1, Conv, [128, 3, 2]]   # Convolutional layer with 128 filters, 3x3 kernel, stride 2
  - [-1, 3, C2f, [128, True]]    # C2f module with 3 repetitions
  - [-1, 1, Conv, [256, 3, 2]]   # Convolutional layer with 256 filters, 3x3 kernel, stride 2
  - [-1, 6, C2f, [256, True]]    # C2f module with 6 repetitions
  - [-1, 1, Conv, [512, 3, 2]]   # Convolutional layer with 512 filters, 3x3 kernel, stride 2
  - [-1, 6, C2f, [512, True]]    # C2f module with 6 repetitions
  - [-1, 1, Conv, [1024, 3, 2]]  # Convolutional layer with 1024 filters, 3x3 kernel, stride 2
  - [-1, 3, C2f, [1024, True]]   # C2f module with 3 repetitions
  - [-1, 1, SPPF, [1024, 5]]     # SPPF module with 1024 filters and 5x5 kernel
```

Figure 3.3: Overview of the Backbone Section in YOLOv8n.yaml File

Head :

The head section describes how the features extracted by the backbone are transformed and utilized for final predictions.

The initial step involves enhancing the resolution of feature maps through **upsampling**, enhancing their suitability for precise object detection. Next, these upscaled features are combined with those from earlier layers via **concatenation**, enabling the model to harness multi-scale information effectively. After this fusion, the concatenated features undergo further refinement through **C2f** modules, which fine-tune filter configurations to extract more nuanced features. This iterative process includes multiple rounds of **upsampling** and **concatenation**, progressively merging information from diverse stages of feature extraction. Ultimately, the architecture converges at a detection head, where features from multiple layers are mixed to enable accurate object detection across a spectrum of scales and complexities.



```
# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]   # Upsampling by a factor of 2
  - [[-1, 6], 1, Concat, [1]]                    # Concatenation with earlier layer
  - [-1, 3, C2f, [512]]                          # C2f module with 512 filters
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]   # Upsampling by a factor of 2
  - [[-1, 4], 1, Concat, [1]]                    # Concatenation with earlier layer
  - [-1, 3, C2f, [256]]                          # C2f module with 256 filters
  - [-1, 1, Conv, [256, 3, 2]]                   # Convolutional layer with 256 filters, 3x3 kernel, stride 2
  - [[-1, 12], 1, Concat, [1]]                   # Concatenation with earlier layer
  - [-1, 3, C2f, [512]]                          # C2f module with 512 filters
  - [-1, 1, Conv, [512, 3, 2]]                   # Convolutional layer with 512 filters, 3x3 kernel, stride 2
  - [[-1, 9], 1, Concat, [1]]                    # Concatenation with earlier layer
  - [-1, 3, C2f, [1024]]                         # C2f module with 1024 filters
  - [[15, 18, 21], 1, Detect, [nc]]              # Detection head combining features from multiple layers
```

Figure 3.4: Overview of the Head Section in YOLOv8n.yaml File

Despite these efforts, the results fell short of our expectations. The table below illustrates the outcomes of our training process, comparing them to the results obtained using the pre-trained model.

Metrics	From Scratch	Pre-trained
Precision	0.717	0.778
Recall	0.633	0.721
mAP50	0.689	0.787
Time	2.733h	2.562h

Table 3.1: Comparison: Pre-trained vs. From Scratch Model

To address this, we turned to software reuse by utilizing the pre-trained YOLOv8.pt model. Training a model from scratch involves initializing all weights randomly, requiring a significant amount of labeled data and computational resources to learn features and patterns from the ground up. This process is time-consuming and often leads to suboptimal performance if the dataset is not sufficiently large or diverse.

In contrast, using a pre-trained model leverages transfer learning, where the model has already been trained on a large, generic dataset. This pre-trained model has learned to detect a wide variety of features that can be useful for different tasks. We fine-tuned this model by training it on our specific dataset of speed bumps and road defects, adjusting only the final layers to match our classes. This approach allowed us to achieve better performance more efficiently by building on the knowledge already acquired by the pre-trained model.

3.4.3 Sofware Re-use

Software reuse involves utilizing existing software artifacts to create new software components, thus preventing the need to repeatedly solve the same problems. This practice includes the use of libraries, design patterns, and frameworks, which can significantly streamline the development process. By coding with reuse principles in mind and leveraging existing artifacts, long-term time savings can be achieved. Essentially, reuse helps technologists avoid redundant efforts.

Reusing existing artifacts in software development offers several key benefits. It avoids redundant work, saving time and resources by preventing duplication of efforts across teams, allowing them to focus on higher-value tasks. This practice ensures consistency and predictability, as approved patterns, frameworks, and libraries used across systems provide uniform behavior and performance, facilitating easier problem-solving within the organization. It also reduces development time, as developers can incorporate existing code instead of creating new solutions from scratch, leading to faster innovation cycles. Additionally, using consistent architecture and design patterns with pre-existing components accelerates the construction of systems and features, ensuring a baseline level of security, performance, and reliability, thus allowing engineers to focus on high-value deliverables and complex problems.

3.4.4 Training

YOLOv8 models are available in various configurations, each tailored to specific trade-offs between speed, accuracy, and computational requirements. YOLOv8n is optimized for edge devices or applications with limited computational resources, prioritizing real-time processing over absolute accuracy. YOLOv8s strikes a balance between speed and accuracy, making it well-suited for tasks in moderate computational environments where a reasonable level of accuracy is crucial. YOLOv8m enhances accuracy further, suitable for applications with higher demands on computational resources. YOLOv8l is designed for scenarios where accuracy is paramount and sufficient computational power is available, ideal for server-side processing. YOLOv8x excels in accuracy-critical applications, leveraging substantial computational capacity typically used in detailed research or precise detection scenarios.

For our project, we selected YOLOv8s due to its appropriate balance of speed and accuracy, which aligns well with the dataset size. This model proves suitable for general object detection tasks without imposing excessive computational demands, particularly advantageous when working within Google Colab's environment. YOLOv8s efficiently processes datasets of this scale, effectively leveraging pre-trained weights to enhance training outcomes. Utilizing pre-trained models facilitates faster convergence and improves performance compared to training from scratch, leveraging prior knowledge to adapt more swiftly to specific dataset characteristics.

Model	mAPval 50-95	Speed A100 TensorRT (ms)	Params (M)	FLOPs (B)
YOLOv8n	37.3	0.99	3.2	8.7
YOLOv8s	44.9	1.20	11.2	28.6
YOLOv8m	50.2	1.83	25.9	78.9
YOLOv8l	52.9	2.39	43.7	165.2
YOLOv8x	53.9	3.53	68.2	257.8

Table 3.2: Comparison of YOLOv8 models based on mAP, speed, parameters, and FLOPs

3.4.4.1 Training Parameters

The model is configured for the object detection task (task=detect) and operates in training mode (mode=train), utilizing pre-trained weights from YOLOv8s.pt. The configuration details are stored in /content/gdrive/My Drive/google_colab_config.yaml on Google Drive. The training process spans 21 epochs, processing batches of 16 images each resized to 640x640 pixels. Data loading is optimized with eight workers, leveraging GPU acceleration for enhanced computational performance. The optimizer is automatically selected, and training is conducted in mixed precision mode (AMP) to maximize efficiency and speed.

3.4.4.2 Albumentations Augmentations

Albumentations, a versatile image augmentation library, enriches dataset variability essential for training machine learning models. It introduces a range of transformations to images based on specific probabilities. For example, the Blur augmentation applies a blur effect with a rare **1%** chance, varying kernel sizes between 3 and 7 to blur images while preserving edge details. Similarly, MedianBlur, also operating with a **1%** probability, reduces noise while maintaining image sharpness within the same kernel size range. Converting images to grayscale with a **1%** probability using ToGray promotes color invariance, aiding the model in focusing on shape and texture cues during training.

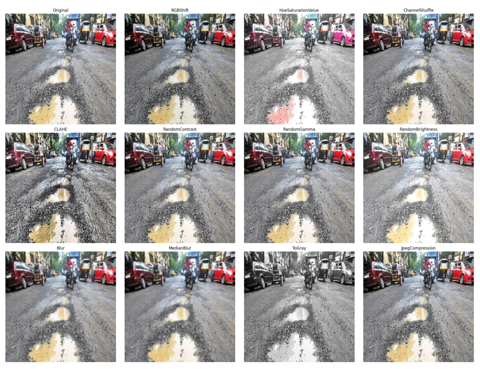


Figure 3.5: Albumentations Augmentations

Additionally, the application of CLAHE (Contrast Limited Adaptive Histogram Equalization) enhances image contrast by locally adapting histograms within smaller regions or tiles of an image, rather than globally. By applying CLAHE as an augmentation technique in machine learning models, it enhances the visibility of image features and details, contributing to improved model robustness and accuracy across diverse datasets and real-world scenarios.

3.4.4.3 Training Data Status

The preparation of training data begins with examining the cache file. This initial step ensures the validation and integrity of annotations associated with the training images. A total of 9476 images are processed, out of which 761 are categorized as background images. Importantly, no corrupted files are identified during this rigorous validation process. However, warnings prompt the removal of duplicate labels from specific images. This meticulous approach guarantees that each training image possesses unique and accurate annotations, which are essential for facilitating effective model learning and ensuring precise object detection throughout both the training and inference phases.

3.4.4.4 Validation Data Status

During the validation phase, the dataset's integrity is thoroughly verified to ensure robustness and reliability. This process involves scanning 1508 images, accompanied by scrutinizing 119 backgrounds and detecting zero corrupt files. Additionally, the validation routine includes identifying and rectifying duplicate labels within certain images, thereby preserving dataset coherence and minimizing potential errors during subsequent model training and evaluation. Visual representations of labeled datasets, such as the saved labels.jpg file, offer accessible insights into annotation accuracy and data distribution patterns, further enhancing dataset validation and preparation processes.

3.4.4.5 Optimizer Configuration

The optimizer configuration is crucial for optimizing model training efficiency. In this scenario, the optimizer=auto directive automatically selects the most suitable optimizer, learning rate (lr0), and momentum settings. Specifically, **AdamW** is chosen as the optimizer, offering effective management of weight decay. The selected learning rate is **0.001667**, with a momentum value of **0.9**, thoughtfully calibrated to achieve optimal convergence during the training process. Parameter groups are assigned distinct weight decay values, contributing to effective gradient descent optimization tailored to the model's requirements and objectives, as shown in the table below.

Optimizer	Learning Rate	Momentum
AdamW	0.001667	0.9

Table 3.3: Optimizer configuration settings

3.4.4.6 Training Setup

The setup phase prepares the model for effective training by standardizing input specifications and optimizing data loading processes. Specifically, training and validation images are resized uniformly to dimensions of 640x640 pixels, ensuring consistent input sizes throughout the training regimen. Data loading operations are streamlined by allocating two data loader workers, optimizing data retrieval and processing speed while efficiently utilizing available computational resources. Furthermore, comprehensive logging mechanisms document and store training results in designated directories, facilitating systematic analysis, troubleshooting, and performance evaluation throughout the **21 epochs** of planned model training.

3.4.4.7 Dataloader Mosaic

After 11 epochs are completed (only 10 epochs are left), the mosaic data loader is closed, indicating that the data loading process, specifically the mosaic augmentation technique, is being finalized or terminated. Mosaic augmentation is a method where four different images are combined into one, creating a diverse and complex training sample that can improve the robustness of the model. This technique is instrumental in enhancing the variability of the training dataset, which can lead to better generalization and performance of the trained model.

3.5 Conclusion

In summary, this chapter outlines the methodology employed to develop a real-time object detection system focused on identifying speed bumps and road anomalies. The process begins with a vehicle equipped with an embedded camera that continuously captures images of the road surface as it moves. These images serve as inputs for our model, which analyzes the live video feed to detect obstacles. Upon identifying an obstacle, an integrated alert system immediately notifies the driver, thereby enhancing road safety and driver awareness.

Critical environmental conditions conducive to optimal model operation were highlighted, emphasizing effective functioning during daylight and the importance of maintaining appropriate vehicle speeds. Factors such as weather conditions, road quality, and camera positioning were also discussed for their impact on model performance.

Customization of the YOLOv8 model architecture was detailed, focusing on its backbone, head, and specific layers such as convolutional blocks and feature fusion modules. The model was tailored to detect two classes (speed bumps and road defects) using Google Colab's GPU resources, significantly accelerating training and inference processes.

Training strategies included both training from scratch and utilizing pre-trained weights, with comparative performance metrics demonstrating the benefits of transfer learning. The use of Albumentations for data augmentation and meticulous data validation procedures ensured the robustness and accuracy of model training.

Chapter 4

Results and Analysis

4.1 Introduction

Before finding datasets online, we attempted to create our dataset by collecting images from various sources and manually annotating them using CVAT according to the YOLOv8 model. However, the results were unsatisfactory due to the limited data. Manually annotating 3000 images proved to be very difficult and time-consuming. We used available online datasets for our second attempt and introduced the YOLOv8 model from scratch. Training YOLOv8 from scratch involves configuring the model from a YAML file (YOLOv8n.yaml), defining the network architecture, adjusting the hyperparameters, and training the model on a specific dataset from randomly initialized weights. Despite these efforts, the results obtained with the model trained from scratch were not as good as we had hoped. Ultimately, we decided to use the pre-trained YOLOv8 model (YOLOv8s.pt) because it delivered the best results, leveraging prior learning on a large dataset to provide more accurate and robust predictions.

4.2 Performance Evaluation of Our Customized YOLOv8 Model

4.2.1 Evaluation Metrics

- **Train Box loss:** This metric measures the loss related to bounding box regression during model training. Bounding box regression focuses on predicting the coordinates (e.g., top-left corner, width, height) of bounding boxes around objects in images. Box loss quantifies the difference between the predicted and ground truth coordinates, helping the model to enhance its localization accuracy.

- **Train Class loss :** Class loss, or classification loss, quantifies the error in predicting the class labels of objects during model training. It assesses the model's ability to differentiate between various object classes. Lower values of classification loss signify higher classification accuracy.

- **Train Dfl loss :** Distributional focal loss is a specialized loss function used in object detection models to enhance performance by addressing two key challenges: class imbalance and object orientation. It combines focal loss, which focuses on harder-to-classify examples, with direction awareness, which enables the model to predict the orientations of objects accurately. This hybrid approach ensures that the model pays more attention to challenging cases and can accurately determine the angles and orientations of various objects, improving overall detection accuracy.

- **Precision (B) :** Precision assesses the correctness of positive predictions generated by the model. In object detection, Precision(B) specifically gauges the accuracy of predicted bounding boxes, indicating the proportion of predicted bounding boxes that accurately localize objects compared to all predicted bounding boxes.

- **Recall (B) :** Recall evaluates the model's capability to identify all pertinent instances of objects within the dataset. In object detection, Recall(B) specifically quantifies the recall of predicted bounding boxes, illustrating the proportion of ground truth bounding boxes that the model successfully detects.

- **mAP50 (B) :** Mean Average Precision at 50% IoU threshold for bounding boxes (mAP50(B)) computes the average precision across various object classes at a defined

IoU threshold of 50%. This metric offers a standardized assessment of detection accuracy, taking into account both precision and recall.

- **mAP50-95 (B) :** Mean Average Precision across a range of IoU thresholds (mAP50-95(B)) expands the evaluation to encompass multiple IoU thresholds, typically spanning from 50% to 95%. This holistic metric provides a deeper understanding of the model's performance across different degrees of object overlap.

- **Val Box loss, Val Class loss, Val Dfl loss :** These metrics quantify the losses incurred during the validation phase of model training: bounding box regression, classification, and direction-aware focal loss. They serve to evaluate how well the model performs on new data and inform refinements aimed at enhancing its ability to generalize.

- **Learning rate:** The learning rate is a crucial hyperparameter in machine learning, controlling the extent of updates to the model's weights during optimization. In complex models like YOLOv8, different parameter groups (PG0, PG1, PG2) may use distinct learning rates for optimal performance. PG0 typically includes the initial layers, PG1 the intermediate layers, and PG2 the final layers. Assigning unique learning rates to these groups stabilizes training and improves convergence. This fine-tuning ensures efficient learning across all layers, enhancing the model's robustness and accuracy in object detection tasks.

4.2.2 Results

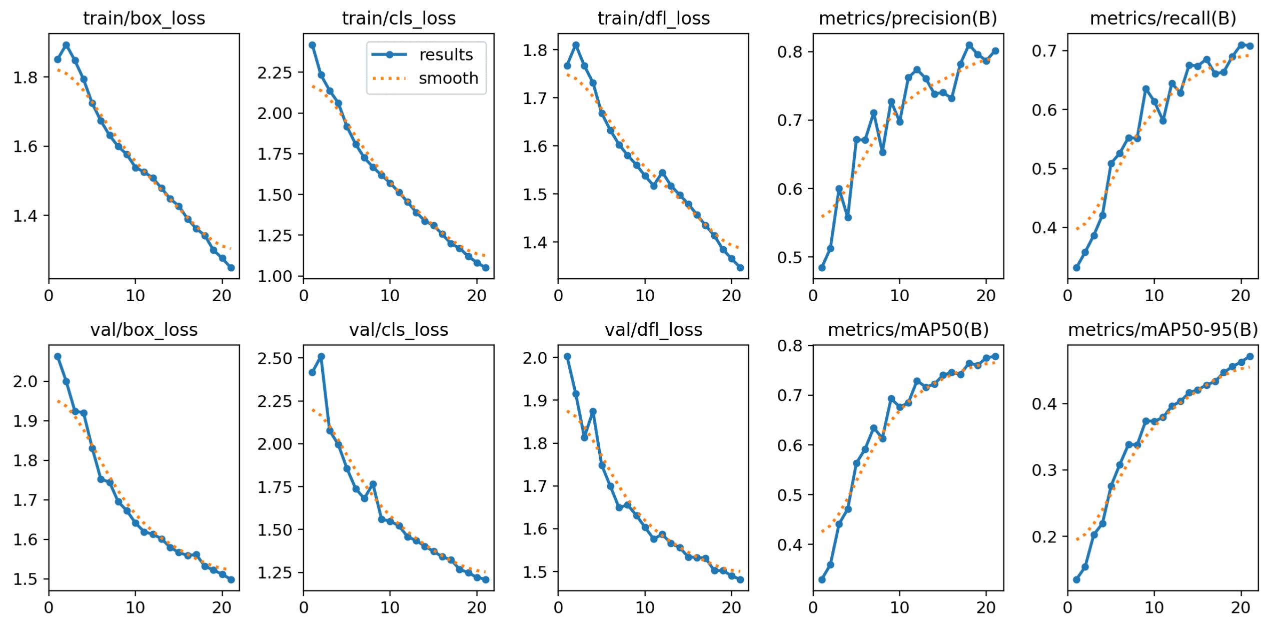


Figure 4.1: Performance Curves Over Training Epochs for Our Model

In the table below, we present the evolution of these metrics to analyze the performance of our YOLOv8 model comprehensively:

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	precision	recall	mAP50	learning rate
1/21	4.22G	1.8514	2.4203	1.7676	0.48447	0.33183	0.32963	0.00055476
2/21	4.29G	1.8942	2.2338	1.8109	0.51285	0.35818	0.35939	0.0010581
3/21	4.16G	1.8482	2.1382	1.7668	0.60014	0.38664	0.44093	0.001509
4/21	4.17G	1.7938	2.0622	1.7316	0.55756	0.42114	0.47121	0.0014312
5/21	4.17G	1.7243	1.9169	1.6682	0.67171	0.50893	0.56331	0.0013527
6/21	4.2G	1.6735	1.8092	1.6329	0.67072	0.52599	0.59167	0.0012741
7/21	4.16G	1.6316	1.7259	1.6031	0.71086	0.55241	0.6346	0.0011955
8/21	4.19G	1.5991	1.6682	1.5805	0.65335	0.5512	0.61344	0.0011169
9/21	4.18G	1.5762	1.618	1.5609	0.72754	0.63564	0.69306	0.0010383
10/21	4.15G	1.5384	1.5706	1.5379	0.69765	0.61357	0.67648	0.00095972
11/21	4.15G	1.5255	1.5136	1.5175	0.76199	0.58084	0.68463	0.00088113
12/21	4.15G	1.5079	1.4537	1.5454	0.77381	0.64439	0.72931	0.00080254
13/21	4.15G	1.4788	1.3908	1.5176	0.76102	0.62864	0.71649	0.00072395
14/21	4.17G	1.4479	1.3375	1.4982	0.73836	0.6755	0.72225	0.00064537
15/21	4.16G	1.4271	1.3098	1.4796	0.74026	0.67374	0.74049	0.00056678
16/21	4.14G	1.39	1.2566	1.4569	0.73183	0.68556	0.74567	0.00048819
17/21	4.14G	1.3618	1.199	1.4346	0.78215	0.6604	0.74193	0.00040961
18/21	4.14G	1.342	1.1706	1.4137	0.8103	0.66375	0.76449	0.00033102
19/21	4.15G	1.2993	1.1205	1.3846	0.79599	0.68996	0.76038	0.00025243
20/21	4.14G	1.2761	1.0824	1.3655	0.78688	0.71013	0.77509	0.00017384
21/21	4.15G	1.2486	1.0509	1.3469	0.80126	0.77835	0.74661	9.35E-05

Table 4.1: Performance Metrics Over Training Epochs for Our Customized YOLOv8 Model

Decreasing Loss :

The consistent decrease in overall loss across epochs implies that the model is undergoing effective adjustments, progressively enhancing its capacity to predict bounding boxes and classify objects accurately.

Metrics Evolution :

Notably, metrics like precision, recall, and mAP50 exhibit a positive trajectory, indicating an improvement in the model's accuracy over time. This suggests that the model is refining its ability to identify speed bumps and road defects with increasing precision and recall rates.

Learning Rate Adjustment :

A consistent decrease in the learning rate is observed throughout the training epochs, showcasing a strategic adaptation aimed at refining the model's parameters. This deliberate reduction in the learning rate is a preventive measure against overshooting and fosters smoother convergence toward an optimal solution. This adaptive adjustment plays a pivotal role in fine-tuning the model's learning process, thereby enhancing its performance and facilitating convergence as the training progresses.

Convergence Trend :

A convergence towards stability in both loss and metric values is observed as epochs advance. This leveling off may signify that the model is nearing its optimal performance, with further training yielding diminishing returns. Such convergence hints at a saturation point in the model's learning process.

Evaluation on Validation Data :

The conformity between the trends observed in loss and metrics on validation data and those on training data is indicative of effective generalization. This alignment suggests that the model is not overfitting to the training data and can generalize well to unseen data, reinforcing its reliability in real-world scenarios.

4.2.2.1 Overall Results

The table below provides an overview of the performance metrics for our detection model :

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
All	1552	3085	0.803	0.709	0.779	0.471
Speedbumps	231	259	0.842	0.846	0.874	0.538
Road defects	1202	2826	0.763	0.572	0.683	0.405

Table 4.2: Overall Performance metrics across different classes.

4.3 Analysis of Detection Results

4.3.1 Confusion Matrix

In object detection tasks, the confusion matrix plays a crucial role in evaluating the model's performance, by assessing its ability to detect multiple classes, including a designated 'background' class representing irrelevant regions in the image.

Including the 'background' class expands the matrix size to $\mathbf{(m+1)\times(m+1)}$, where m is the number of defined classes. Each cell **(i, j)** in this matrix represents a specific scenario where the model predicts the **jth** class as the **ith** class. The matrix also includes additional rows and columns: **(i, M)** for instances where the model wrongly identifies background as class **i**, and **(M, j)** for cases where the model misses detecting class **j**.

YOLOv8 employs two thresholds "confidence and Intersection over Union (IoU)" to classify predictions into this matrix. Predictions not meeting the confidence threshold are excluded, while valid predictions are organized into an m×n matrix based on IoU values. These predictions are further refined by sorting them by IoU, resolving duplications, and assigning labels based on IoU values to determine true positives or false positives.

This process is repeated for each image in the test dataset, constructing a detailed confusion matrix that offers insights into YOLOv8's performance across various classes and backgrounds, highlighting areas for enhancement in object detection tasks.

4.3.1.1 Theory of Decision

True Positive : A True Positive (TP) is registered when the model accurately identifies the presence of an object and correctly assigns its class. For instance, if the model identifies and labels a car in an image as a car, this counts as a true positive. In terms of bounding boxes, if the predicted bounding box overlaps adequately with the ground truth bounding box and accurately identifies the object inside it, it also qualifies as a true positive.

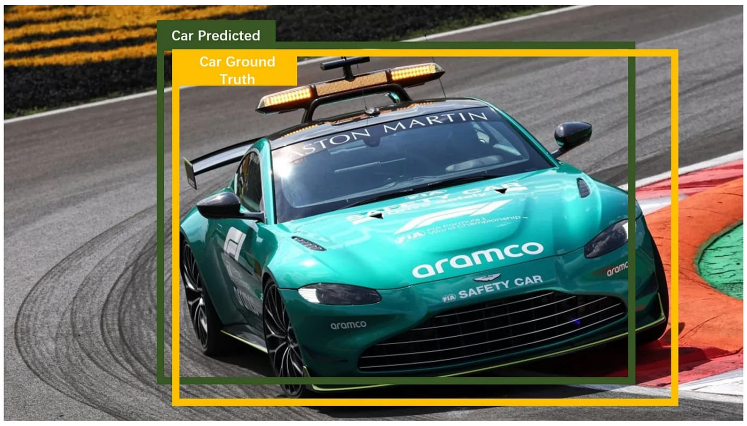


Figure 4.2: A True Positive Prediction

False Positive : A False Positive (FP) occurs when the model incorrectly predicts the presence of an object or misclassifies the object within the bounding box. For example, this happens when the predicted bounding box lacks sufficient overlap with the ground truth bounding box, or when it fails to overlap altogether. Such discrepancies often stem from localization inaccuracies.

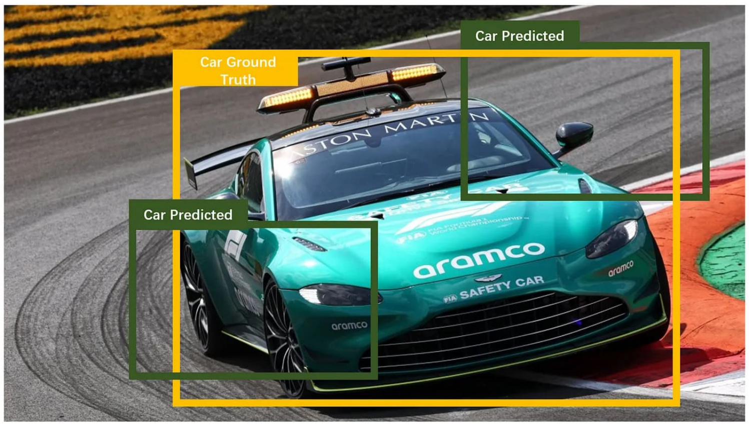


Figure 4.3: Two false positive predictions because of not enough overlapping region

Another instance occurs when the predicted bounding box overlaps adequately with the ground truth bounding box, but the model incorrectly classifies the object within. For example, if the predicted bounding box closely matches the ground truth but misidentifies the object's class (such as labeling a car as a truck), it results in a false positive.

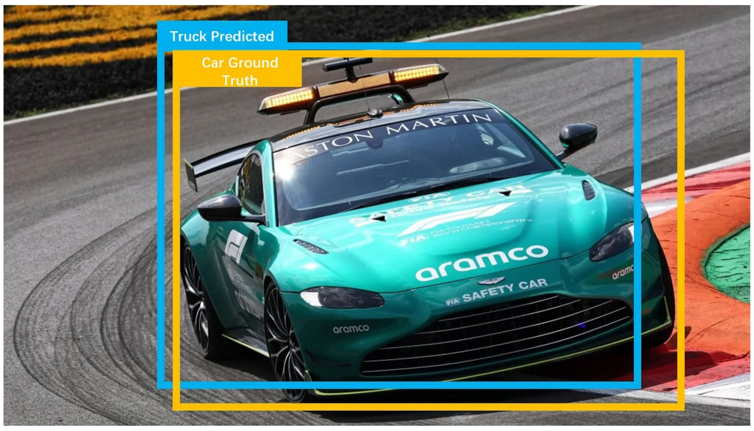


Figure 4.4: Misclassify a car as a truck. One False Positive in the image

False Negative : False Negatives (FN) happen when the model either misses detecting the presence of an object entirely or misclassifies the object within the bounding box. This occurs when the model fails to predict a bounding box for an object present in the image or when the predicted bounding box does not adequately overlap with the ground truth bounding box. In scenarios with multiple objects in an image, each undetected object contributes to the tally of false negatives.

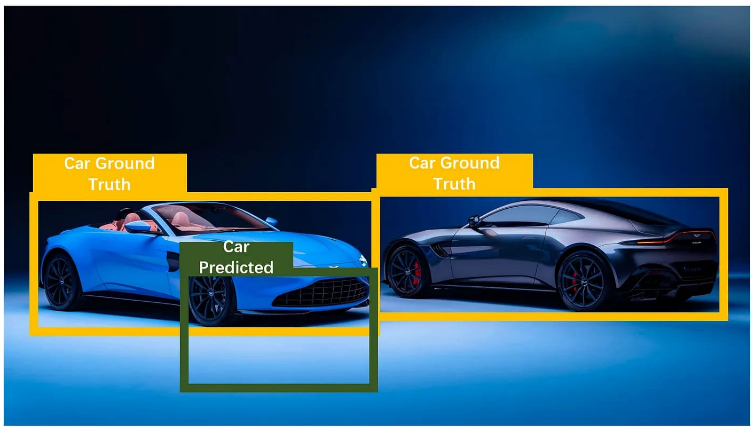


Figure 4.5: Two False Negatives and One False Positive

True Negative : True Negatives (TN) are implicit in object detection evaluations, as they represent all image areas not identified as containing objects by the model. These regions include correctly identified backgrounds or non-relevant areas within the image.

4.3.1.2 Results

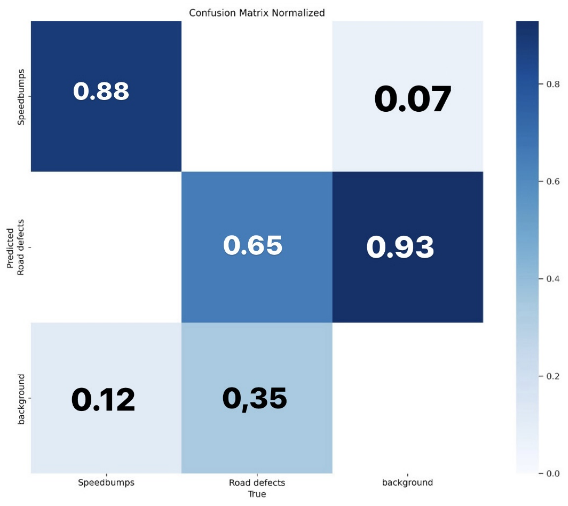


Figure 4.6: Confusion Matrix Normalized

The model's proficiency in detecting potholes shows an accuracy rate of **65%**, indicating it correctly identifies **65%** of potholes in test images. However, there is a **35%** margin where potholes are missed and mistakenly classified as background elements. In the analysis of background detection, approximately 700 instances of false positives were observed, with **93%** of background regions—encompassing road defects like cracking, crazing, and edge failures—incorrectly identified as potholes. While this high false positive rate might initially raise concerns, it underscores the model's heightened sensitivity to detecting any type of road defect.

This issue is a common challenge in road defect detection across various databases we've utilized, but it doesn't significantly diminish the efficacy of our model. Detecting a range of road defects, including those less critical than potholes, provides valuable insights for drivers and contributes to overall road safety and vehicle maintenance. Thus, what appears as a setback highlights the advantageous nature of our model's capability to detect diverse road hazards.

Additionally, our model excels in distinguishing speed bumps with an impressive accuracy rate of **88%**. It accurately identifies **88%** of speed bumps in the test data, with a **12%** margin where speed bumps are missed or misclassified as background elements. Moreover, the model demonstrates a low misclassification rate of **7%** for background areas erroneously identified as speed bumps, underscoring its effectiveness in differentiating non-pothole backgrounds from genuine speed bumps.

Overall, the model shows strong performance in identifying road defects, particularly excelling in speed bump detection. While its pothole detection accuracy is reasonable, its sensitivity to various road defects underscores its effectiveness in enhancing road safety and vehicle maintenance.

4.3.2 Evaluation Metrics Curves

4.3.2.1 Precision - Confidence

The precision-confidence curve illustrated in the graph provides insights into our detection model's performance across various confidence levels.

Analyzing the curves for different object classes allows us to gauge the model's performance characteristics. Ideally, as confidence levels increase, precision should remain high or even improve slightly, indicating that the model is confidently detecting objects with a high degree of accuracy. Conversely, lower confidence levels may lead to decreased precision, suggesting that while the model detects more objects, it might also include incorrect detections, impacting overall accuracy.

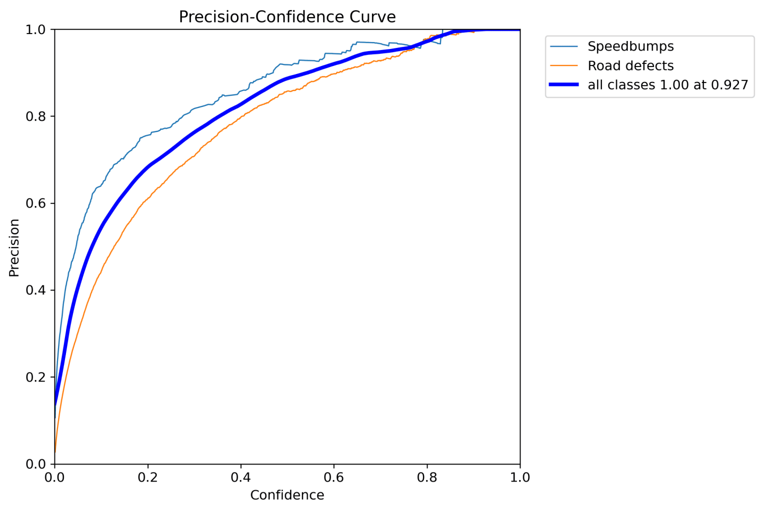


Figure 4.7: Precision-Confidence Curve

Our model displays high confidence in detecting speed bumps across all confidence thresholds, ensuring correct detections even at lower confidence levels. However, for road defects, particularly at lower confidence levels, the model appears less confident, potentially resulting in missed detections. The overall performance, represented by the curve for all classes, indicates a valid precision level, suggesting the model's effectiveness on average but potential challenges with road defects.

This observation is consistent with the notion that speed bumps possess more discernible visual characteristics than the broader spectrum of objects encapsulated within **"all classes"**, including general road defects.

4.3.2.2 Recall - Confidence

The Recall-Confidence Curve provides insights into the performance of the model in detecting speed bumps and road defects. This curve illustrates the relationship between recall (the ability to identify all relevant instances) and the confidence threshold (the model's certainty before making a prediction).

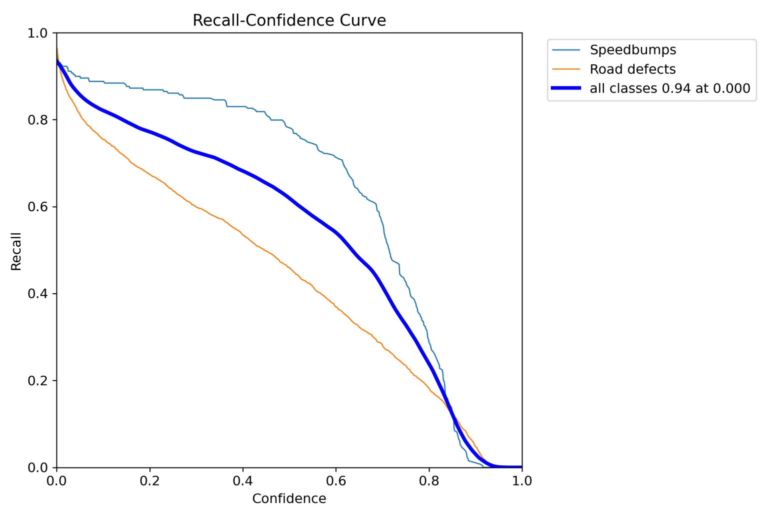


Figure 4.8: Recall-Confidence Curve

For speed bumps, recall starts high **0.9** at lower confidence thresholds, meaning the model can identify most speed bumps when it is less certain. However, as the confidence threshold increases, recall gradually decreases. This suggests that while the model is effective at detecting speed bumps at lower confidence levels, it starts to miss more true positives as it demands higher confidence for its predictions.

In the case of road defects, recall also begins high but drops more sharply compared to speed bumps as the confidence threshold increases. This indicates that the model stumbles more with detecting road defects, losing recall more quickly as it becomes more confident.

The combined performance for all classes shows a similar trend where recall starts high at low confidence thresholds and decreases steadily as the confidence threshold increases. This overall trend underscores the trade-off between recall and confidence: lower confidence thresholds result in higher recall but potentially more false positives, while higher confidence thresholds reduce recall but increase the model's precision.

4.3.2.3 F1 - Confidence

The curve is an F1-Confidence Curve, which plots the F1 score against the confidence threshold for different classes within a classification model. This type of curve is used to evaluate how the F1 score varies as the confidence threshold is adjusted, providing insights into the model's performance across different thresholds.

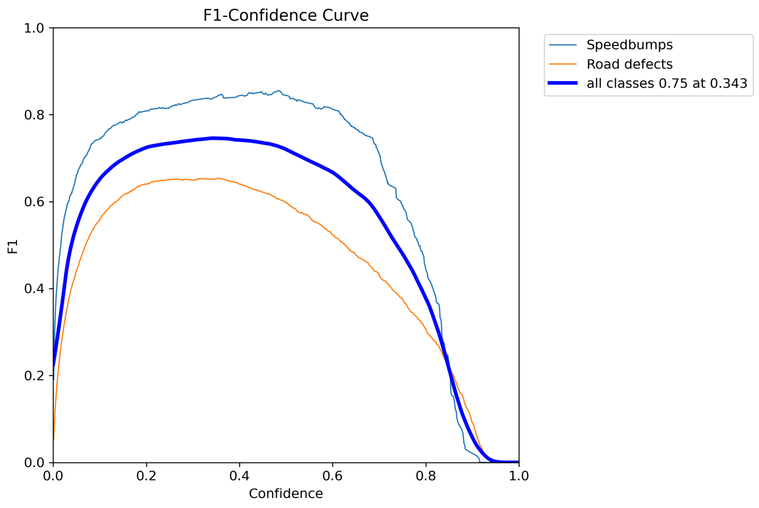


Figure 4.9: F1-Confidence Curve

The Speedbumps class peaks at a high F1 score close to 0.9 around a confidence threshold of 0.4 before it gradually decreases as the threshold increases. The Road defects class peaks at a lower value of around 0.7 and then gradually declines with higher confidence thresholds.

The F1 score for all classes combined indicates the overall performance of the classification model. The F1 score for this combined metric reaches its maximum value of 0.75 at a confidence threshold of 0.343, signifying the optimal threshold for the best overall F1 score across all classes.

4.3.2.4 Precision - Recall

When evaluating a classification model, the precision-recall curve becomes a secret weapon. It unveils the crucial trade-off between two key performance metrics: precision and recall.

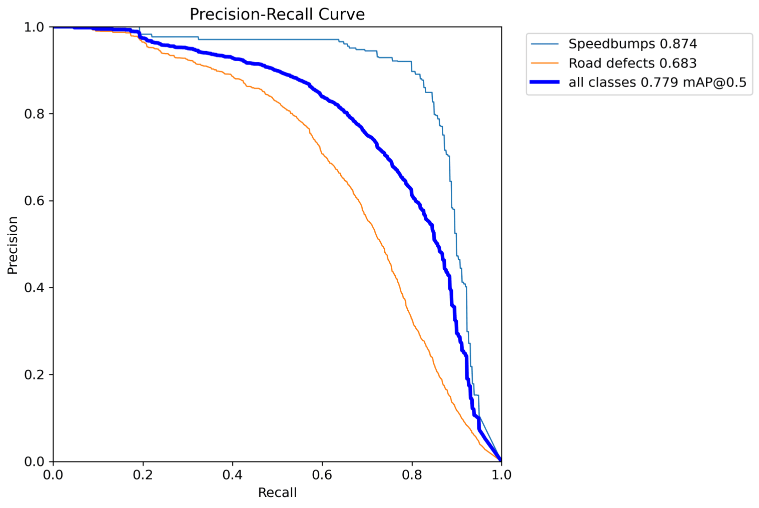


Figure 4.10: Precision-Recall Curve

In our scenario where we train a model to identify speed bumps and road defects, precision tells us how accurate the model is in its identifications – are the bumps it finds truly bumps, or is it flagging other things? Recall, on the other hand, focuses on how well the model catches all the actual speed bumps and road defects – does it miss any important ones?

The precision-recall curve elegantly captures this balancing act. It plots precision on the vertical axis and recall on the horizontal axis. Ideally, we want a curve that starts high in precision and stays elevated even as recall increases. This signifies a model that excels at correctly classifying both speed bumps and road defects, minimizing missed detections.

The curve for the Speedbumps class demonstrates that the model maintains a high level of precision across various recall levels, achieving an average precision (AP) of 0.874. This indicates that the model is highly effective at identifying speedbumps while minimizing false positives.

The curve for the Road defects class shows a more moderate level of precision, with an AP of 0.683. Precision for this class decreases more rapidly as recall increases, suggesting that the model can accurately distinguish road defects, but is less effective than speed bumps.

4.4 Efficiency and Performance of YOLOv8 in Real-Time Object Detection

Type	**Preprocess (ms)**	**Inference (ms)**	**Postprocess (ms)**	**Input Shape**
Images	4.3	663.7	1.2	(1, 3, 640, 640)
Video 01	4.6	404.9	0.8	(1, 3, 384, 640)
Video 02	5.0	424.3	0.8	(1, 3, 384, 640)
Video 03	2.6	419.5	0.7	(1, 3, 384, 640)
Video 04	2.3	430.6	0.7	(1, 3, 384, 640)
Video 05	2.4	425.0	0.7	(1, 3, 384, 640)

Table 4.3: Processing Times for YOLOv8

The YOLOv8 model exemplifies its efficiency through distinct phases of processing: preprocessing, inference, and postprocessing.

Preprocessing involves preparing input data by resizing and standardizing it to a form suitable for the model—typically (1, 3, 640, 640) for images and (1, 3, 384, 640) for videos. This step ensures uniformity and optimal data compatibility before feeding it into the model.

Inference, the core computational stage, applies the model's learned parameters to the preprocessed data to detect objects within the images or frames. YOLOv8 showcases impressive speed in this phase, ranging from approximately 419.5 to 663.7 milliseconds per image, depending on the input type and size.

Postprocessing refines the model's predictions, filtering and refining outputs like bounding boxes and class probabilities. YOLOv8's performance benchmarks illustrate its strength in real-time object detection across both images and videos, offering competitive processing times compared to other detection methods like Faster R-CNN or SSD, while maintaining high accuracy and efficiency in detecting and localizing objects within visual data.

4.5 Conclusion

In conclusion, this chapter provides a detailed evaluation of our real-time object detection system centered around the customized YOLOv8 model. Throughout 28 training epochs, we meticulously analyzed essential metrics, witnessing continuous improvement across all parameters. The strategic adjustment of learning rates proved pivotal in stabilizing training and enhancing convergence, contributing to overall performance gains.

Analysis of detection results, including the confusion matrix underscored the model's strengths in accurately detecting speed bumps while revealing areas for improvement in broader classifications like road defects. Comparative analysis with alternative detection approaches suggested avenues for future research, while practical testing validated our method's efficacy in enhancing road safety and infrastructure maintenance.

General Conclusion

The study identified several limitations in our detection model. Primarily, the model struggles under poor lighting conditions at night, and its accuracy is influenced by vehicle speed, with higher speeds potentially compromising its effectiveness. Additionally, while the model assists drivers in prediction, it does not replace the need for driver vigilance. Notably, our work exhibits an imprecision rate of **0.20**, indicating potential instances of missed detections.

The model demonstrates robust performance metrics, including a precision of **0.803** and a recall of **0.709**, signifying its adeptness in accurately identifying positive cases and capturing a substantial portion of actual positives. Furthermore, achieving a mAP50 score of **0.779** underscores its capability in precisely localizing objects with an IoU threshold of **0.50**.

Overall, the study contributes significantly by advancing the technological frontier in road hazard detection using the YOLOv8 CNN model. Our research provides empirical evidence of the model's effectiveness and reliability in real-world applications by addressing key challenges such as varying road defect shapes and environmental impacts on detection accuracy.

Implementing the YOLOv8 model in practical road safety applications carries profound implications. Real-time detection of road defects and speed bumps equips drivers with crucial information to navigate safely and respond proactively to potential hazards, thereby reducing accidents and minimizing vehicle damage. Moreover, enabling real-time monitoring and identification of road defects supports efficient infrastructure maintenance, facilitating timely repairs and enhancing overall road conditions and public safety.

Future research should prioritize several key areas to further advance road defect and speed bump detection using AI technologies. Practical integration of the YOLOv8 model into vehicle-compatible hardware systems, optimizing for real-time processing efficiency, energy conservation, and robustness across diverse environmental conditions, is crucial. Exploring multi-sensor integration, such as LIDAR and radar, could significantly bolster detection accuracy, especially in challenging conditions like adverse weather or low visibility. Additionally, expanding the dataset diversity for model training to encompass a broader spectrum of road defect scenarios will enhance its applicability across various geographic and road-type contexts.

Human-centered design principles should guide the development of user interfaces, ensuring seamless integration of detection alerts into driver interfaces for enhanced usability and effectiveness. Longitudinal studies evaluating the model's long-term performance and reliability in real-world deployments will provide valuable insights for iterative improvements and adaptations.

In conclusion, this work lays a solid foundation for advancing academic knowledge and practical implementation of AI-driven solutions in enhancing road safety by effectively detecting critical road defects and speed bumps. By addressing these research avenues, future studies can build upon this groundwork, contributing to safer and more efficient road networks globally.

Bibliography

[1] Aquila Data. Livre blanc "les évolutions de yolo". https://www.aquiladata.fr/wp-content/uploads/2021/10/Livre-Blanc-22Les-volutions-de-YOLO22-par-Charles-Franchomme-Aquila\protect\@normalcr\relax-Data-Enabler.pdf, 2021. Accessed: 2024-03-20.

[2] Zihao Geng. How to evaluate an object detection model: Iou, precision, recall, and map, 2024. Accessed: 2024-04-08.

[3] Touraya El Hassani. You only look once: Un réseau de neurones pour la détection d'objets. https://blog.octo.com/you-only-look-once-un-reseau-de-neurones-pour-la-detection-dobjets, 2022. Accessed: 2024-03-07.

[4] Amir Tosson. The way to a smarter community: Exploring and exploiting data modeling, big data analytics, high-performance computing, and artificial intelligence techniques for applications of 2d energy-dispersive detectors. https://www.researchgate.net/publication/343653793_The_way_to_a_smarter_community_Exploring_and_Exploiting_Data_Modeling_Big_Data_Analytics_High-Performance_Computing_and_Artificial_Intelligence_Techniques_for_Applications_of_2D_Energy-Dispersive_Detectors, 2020. Accessed: 2024-03-25.

[5] R. Divya and J. Dinesh Peter. Smart healthcare system - a brain-like computing approach for analyzing the performance of detectron2 and posenet models for anomalous action detection in aged people with movement impairments. https://www.researchgate.net/publication/349960212_Smart_healthcare_system-a_brain-like_computing_approach_for_analyzing_the_performance_of_detectron2_and_PoseNet_models_for_anomalous_action_detection_in_aged_people_with_movement_impairments/figures?lo=1&utm_source=google&utm_medium=organic, 2021. Accessed: 2024-03-25.

[6] R Vijayabhaskar. PASCAL VOC 2007 and 2012. https://www.kaggle.com/datasets/vijayabhaskar96/pascal-voc-2007-and-2012, 2020. Accessed: 2024-03-28.

[7] Saad Mohammad Alkentar, B. Alsahwa, A. Assalem, and D. Karakolla. Practical comparison of the accuracy and speed of yolo, ssd, and faster r-cnn for drone detection. *Journal of Engineering*, 2021. Accessed: 2024-03-30.

[8] V7 Labs. Yolo object detection: The most popular ai model for computer vision, 2023. Accessed: 2024-04-01.

[9] Ultralytics. Ultralytics x paperspace: Advancing object detection capabilities through partnership, 2023. Accessed: 2024-04-03.

[10] Author(s). Title of the article. *Applied Sciences*, 13(14):8349, 2023.

[11] Federal Highway Administration. Roadway safety factsheet. https://highways.dot.gov/sites/fhwa.dot.gov/files/2022-06/factsheet.pdf, 2022. Accessed: 2024-04-04.

[12] The Lancet Public Health. Global, regional, and national burden of road injuries and fatalities: 1990–2017. *The Lancet Public Health*, 4(8):e402–e410, 2019. Accessed: 2024-04-05.

[13] IKO Group. Road defects: the most common types. https://ikogroup.co.uk/news-advice/road-defects-the-most-common-types/, 2021. Accessed: 2024-04-07.

[14] ScienceDirect. Artificial intelligence in road safety: A review. *Procedia Computer Science*, 151:876–883, 2019. Accessed: 2024-04-08.

[15] ResearchGate. What is artificial intelligence? *ResearchGate*, 2009. Accessed: 2024-04-10.

[16] National Center for Biotechnology Information. Artificial intelligence in healthcare: A review. *Frontiers in Artificial Intelligence*, 4:56, 2023. Accessed: 2024-04-11.

[17] DeepAI. Machine learning glossary and terms: Convolutional neural network. https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network. Accessed: 2024-04-12.

[18] IBM. Computer vision. https://www.ibm.com/fr-fr/topics/computer-vision. Accessed: 2024-04-14.

[19] Analytics Vidhya. A basic introduction to object detection. *Analytics Vidhya Blog*, 2022. Accessed: 2024-04-15.

[20] Neptune AI. Object detection algorithms and libraries: A comprehensive guide, 2023. Accessed: 2024-04-16.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.02640*, 2015. Accessed: 2024-04-17.

[22] Viso AI. Yolov8 guide. https://viso.ai/deep-learning/yolov8-guide/. Accessed: 2024-04-17.

[23] Ultralytics. Ultralytics documentation. https://docs.ultralytics.com/, 2023. Accessed: 2024-04-20.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. Accessed: 2024-04-18.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. Accessed: 2024-04-18.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. Accessed: 2024-04-19.

[27] Keylabs AI. Training yolov8 models: Tips for success, 2023. Accessed: 2024-04-19.

[28] Joseph Redmon. Darknet-19. https://paperswithcode.com/method/darknet-19, 2015. Accessed: 2024-04-16.

[29] Siddhesh Bhobe. Vgg net architecture explained. https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f, 2019. Accessed: 2024-06-16.

[30] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022. Accessed: 2024-04-18.

[31] Ultralytics. Detect: Export. https://docs.ultralytics.com/tasks/detect/#export, 2023. Accessed: 2024-04-20.

[32] Viso.ai. Object detection: Techniques, methods, and applications, 2023. Accessed: 2024-04-22.

[33] Ultralytics. Integrations: Training yolov8 using google colaboratory. https://docs.ultralytics.com/integrations/google-colab/#training-yolov8-using-google-colaboratory, 2023. Accessed: 2024-04-24.