**Media Engineering and Technology Faculty**
**German University in Cairo**

# Vehicle counting and classification

**Bachelor Thesis**

Author: Mahmoud Yassen Khodary
Supervisors: Prof Dr.HossamEldin Hassan
Submission Date: 29 Mai, 2023

**Media Engineering and Technology Faculty**
**German University in Cairo**

# Vehicle counting and classification

**Bachelor Thesis**

Author: Mahmoud Yassen Khodary
Supervisors: Prof Dr.HossamEldin Hassan
Submission Date: 29 Mai, 2023

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Mahmoud Yassen Khodary
29 Mai, 2023

# Acknowledgments

First of all, I want to thank god for everything I made. I am sincerely grateful to all those who assisted me in completing this thesis. Your help and assistance has been invaluable, and I deeply appreciate your contributions. I'd want to express my gratitude to Prof Dr Hossameldin Hassan for his outstanding contribution and unwavering support. His advice was invaluable to me during the research and writing of this thesis. For my Bachelor's degree, I could not have asked for a greater mentor. Furthermore, I express my deepest gratitude to my family for their unlimited support and boundless love. Their unwavering encouragement has been a constant source of inspiration and motivation for me. and finally, I'd to thank my friends for supporting me this semester and Encouraging me to keep going and offer the best results.

# Abstract

This bachelor's thesis explores the application of machine learning and deep learning in Vehicle counting and classification using Deep neural networks models. With the advancements in computer vision and machine learning, there are new possibilities for counting and classifying vehicles. This thesis aims to detect vehicles in images and videos, count them and classify them using models of object detection. These models operate on IDE such as Pycharm or google collab, capture the video or image, use the model of object detection, and apply this model on each image or frame in the video, after applying the model they detect the vehicles in the input, classify them, count them and at the end each model and its code gives you a resulting video after detection, counting, and classification. Vehicle counting and classification provides users with the number of vehicles in the input which can be image or video. By utilizing deep learning and Computer vision. the bachelor aims to provide users with methods that use these models which will help individuals to count the vehicles in video or image without using traditional counting methods.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

In the current era, (see Figure1.1) the proliferation of vehicles and extensive road networks in many cities has given rise to a pressing concern: traffic congestion. Addressing this issue requires the effective detection of vehicles in traffic images or videos, which serves as a fundamental task for urban traffic prediction. Consequently, the focus is on developing algorithms that possess real-time computational capabilities and can accurately detect vehicles. This project stems from the escalating need for efficient systems capable of counting and classifying vehicles.

Traditionally, manual vehicle counting has been a painstaking and time-consuming process prone to errors. Furthermore, it fails to provide comprehensive information about the vehicles, such as their type, speed, or direction of travel. However, with the rapid progress of computer vision and artificial intelligence technologies, researchers have extensively explored object detection algorithms based on deep learning. Among these approaches, computer vision-based vehicle counting and classification systems have gained prominence due to their ability to accurately count vehicles and provide supplementary data, including vehicle classification.

## 1.2 Aim of the project

This project aims to develop vehicle counting and classification system using computer vision techniques (see Figure1.2). The the ultimate goal is to provide a solution that can be deployed in real-world scenarios to improve traffic management and enhance the overall quality of life for urban residents. The system will be designed to operate on video and images. Counting and classifying vehicles is an essential task for traffic management, determining whether particular roads have traffic problems and whether they are congested, they can provide better real-time guides for drivers. Hence, traffic flow statistics have

Figure 1.1: Vehicles on road

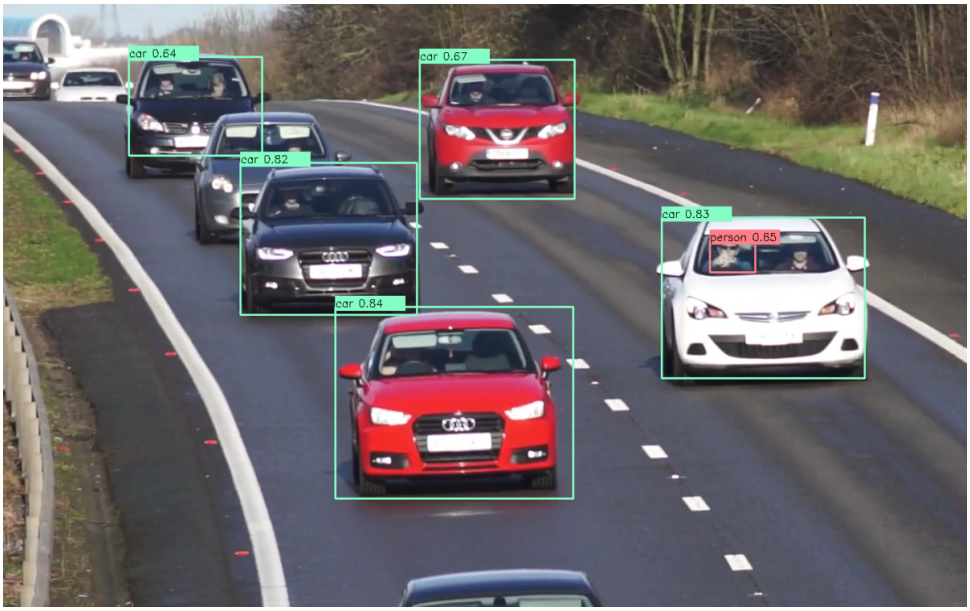



Figure 1.2: Example image

important applications. By automating this process, it will help save time and resources that would otherwise be spent on manual counting and classification. This project has the potential to be integrated into a smart city systems, enhancing the efficiency and safety of urban areas. In addition to the benefits of traffic management, the vehicle count and classification systems also have the potential to be applied in other areas such as security and surveillance. By using computer vision methods and techniques like object detection and classification, these systems can provide real-time data about the types and numbers

of vehicles in a given area. Accurate detection and counting of vehicles on the road play a crucial role in traffic information analysis, which in turn contributes to effective traffic control and management. These tasks are paramount for ensuring a safe and efficient transportation system. By reliably identifying and quantifying vehicles, authorities can make informed decisions and implement strategies to alleviate congestion, enhance traffic flow, and optimize overall road safety.

## 1.3 Thesis Outlines

This thesis consists of 7 chapters. The second chapter includes all the topics we required to reach the objective of this thesis. The third chapter was the approach taken to be able to construct the model. The fourth chapter talks about Hardware and GPU and its role in Computer vision. The fifth chapter consists of all the implementations of all the models built. The sixth chapter contains the results of these models used. The seventh chapter is the conclusion of the whole process and all suggestions for any future work regarding the same topic.

# Chapter 2

# Background

In general, there are various techniques and algorithms for performing Object Detection tasks and training the computer; however, in this chapter, an introduction to basics of the object detection that will help us to build a system that can be used to detect objects, as well as how these basics can be used to achieve the project's goal.

## 2.1 Brief Introduction

In the last decades, Object detection and classification algorithms are classified as an application of machine learning, deep learning, and other concepts and more details regarding these concepts will be further explained in the following part.

### 2.1.1 python

According to Machine Learning in Python [1], and [2], Python is a widely used programming language known for its simplicity and readability. It is a versatile programming language that has gained immense popularity in the technology industry [3]. According to [1]it is simple, which makes it easy to learn and understand, even for beginners. Python's versatility stems from its extensive library support and a broad range of applications. According to [4], in the field of web development, Python frameworks provide robust tools for creating dynamic and scalable websites. Python's simplicity and efficiency enable developers to build web applications quickly and maintain them effortlessly. Data Science and ML are other domains where Python is used to implement systems like Object detection. Its rich ecosystem includes libraries like NumPy, Pandas, and Scikit-learn which are used to empower data scientists to analyze and manipulate data efficiently. Python's exceptional flexibility renders it a highly suitable option for building machine learning models and implementing algorithms. Python's significance extends to the field of artificial intelligence (AI). With libraries such as TensorFlow and Keras. Python enables

developers to create AI applications, including natural language processing, computer vision and deep learning. Additionally, Python is used for scripting and automation tasks. Python libraries make it a good choice for writing scripts that automate repetitive tasks, simplifying workflows and saving time for developers. Python's influence can also be seen in network programming and cyber security. Its libraries, such as Scapy and Paramiko, allow developers to create powerful network tools and secure communication channels. Furthermore, Python is used in finance, game development, and system administration. The importance of Python lies in its ability to streamline development processes, enhance productivity, and foster innovation. Its extensive community support and active development ensure that Python remains up-to-date and relevant in the ever-evolving world of technology. In conclusion, Python's simplicity makes it a crucial programming language in technology fields like web development, AI, scripting, network programming and more. Its impact and significance continue to grow as it empowers developers to build innovative solutions efficiently.

### 2.1.2 python packages

According to [5], and [6] Python has a lot of powerful packages that enable users to create models efficiently. They provide pre-written code and functionalities that developers can easily integrate into their projects, saving time and effort. Packages enhance productivity, promote code reusability and enable developers to influence a specialized tools and libraries for various applications. According to [7], in the project we used some of them like num py which stands for (Numerical Python). According to [8], it is the analytical backbone of the Python programming language. It provides multidimensional arrays which are like 2D arrays, along with a large set of functions to operate a multiplicity of mathematical operations on these arrays. Arrays represent data blocks organized across multiple dimensions, effectively implementing mathematical vectors and matrices. Arrays serve a purpose beyond mere data storage, as they offer the advantage of expedient matrix operations through vectorization. According to [9]],the second one is Tensorflow package which is an open-source software library for dataflow programming across a range of tasks. The library is not only dedicated to symbolic math but also finds extensive utility in machine learning domains, particularly for tasks involving neural networks. The third one which is used a lot in the Computer Vision field is CV2 also known as OpenCV. It provides a comprehensive set of tools and functions for video and image processing, object detection and tracking, facial recognition, and more. With "cv2," developers can perform tasks like image manipulation, filtering and feature extraction. The package supports various image file formats and offers real-time computer vision capabilities. It is efficient and optimized for speed, making it well-suited for deployment in various environments, ranging from research settings to production environments. The extensive functionality of "cv2" make it an essential tool for developing computer vision applications in Python.

### 2.1.3 Introduction to Machine learning

According to [10], Machine learning (ML) is a branch of artificial intelligence (AI) and a method of data that automates analytical model build. It allows software applications to become accurate when they predict outcomes, earn from data, identify patterns and make decisions with minimal human intervention. According to [11], [12], and [13], machine learning helps intelligent machines by giving them the knowledge to support their functionalities. Essentially, machine learning algorithms are inserted into machines and data streams gave so that information and data are obtained and fed into the system for faster and more efficient management of processes. According to [14], Machine learning has important roles in industry such as applications in manufacturing and production systems. The term "machine learning" was introduced by Arthur Samuel in 1959. He described ML as a paradigm that enables computers to acquire knowledge and learn without explicit programming. This groundbreaking concept opened up new possibilities and raised intriguing questions. Instead of relying on the manual crafting of data-processing rules by programmers, could computers autonomously learn these rules by analyzing data? These questions opened the door to a new programming paradigm called Machine learning. In classical programming, we have a box where the user inputs rules (a program) to the model or the program and data to be processed according to these rules, and outcome answers (see Figure2.1) shows an example diagram of traditional programming, but with

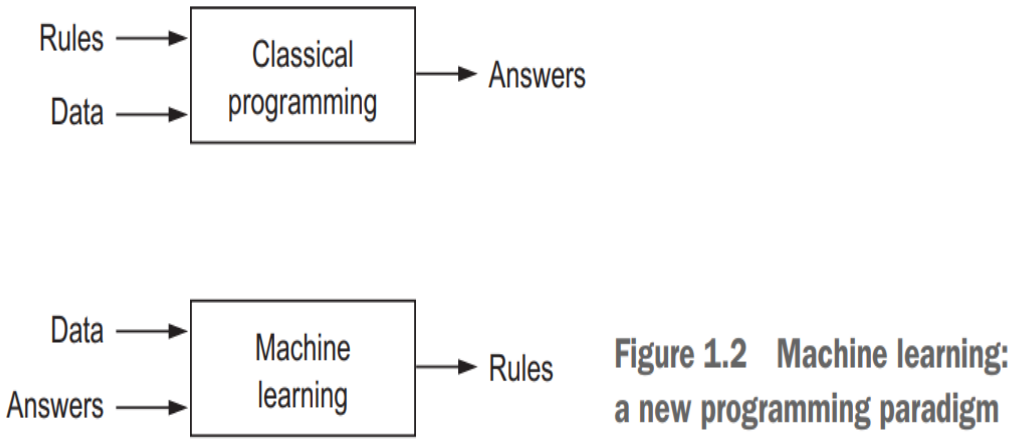



Figure 2.1: Machine Learning vs Classical programming

machine learning user inputs data, also the answers expected from the data, and out come to the rules. These rules could be applied to unseen data to produce original answers. Figure2.1 shows the machine learning programming example and this shows how it's different from traditional programming. Machine learning is a trained system rather than explicitly programmed. It's presented with many examples relevant to the task you

want to perform, and it finds statistical structure in these examples that Ultimately, this enables the system to generate automated rules for accomplishing the task. For instance, if you want to detect vehicles in the pictures, you could present a machine-learning system with many examples of pictures like 10 thousand already have vehicles init, and the system would learn statistical rules for associating specific pictures with specific tags and these examples of pictures called Dataset.

### 2.1.4 Machine learning approaches

According to [15] and [16] ML algorithms are divided into supervised and unsupervised (see Figure2.2) . The contrast between these two primary classes is the presence of

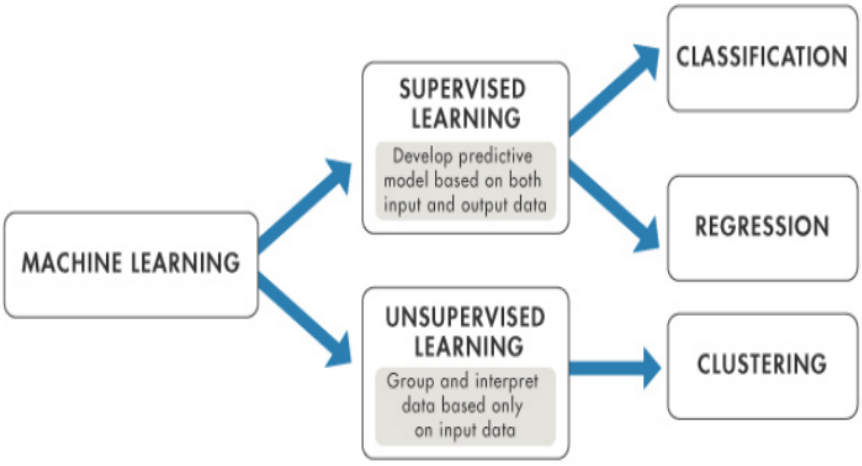



Figure 2.1: Machine Learning

Figure 2.2: Machine Learning Approaches

labels within the training data subset. However, both the two approaches' algorithms are used to accomplish different data mining tasks. Supervised Learning According to [17] , Supervised machine learning supplies algorithms with labelled training data and defines the variables they need the algorithm to assess for correlations. Unsupervised machine learning requires algorithms that train on unlabeled data. In scenarios where there is a substantial volume of input data and only a portion of that data is labelled, a challenge arises, this is called a semi-supervised learning problem. These problems sit in between supervised and unsupervised learning.

## 2.2 Deep learning

According to [18], deep learning is considered a specific subfield of machine learning(just means machine learning using deep neural networks). It has gained immense popularity

due to its exceptional performance in various complex tasks like image recognition. The deep in deep learning stands for this idea of successive layers of representations. These layers contribute to a model of the data is called the depth of the model. Other appropriate names for Deep learning are layered representations learning and hierarchical representations learning. According to [19] deep learning consists of tens or even hundreds of successive layers of representation and they're all learned automatically from each other to train data and extract features. Meanwhile, machine learning tends to focus on learning only from one or two layers of representations of the data; hence, they're sometimes called shallow learning.

### 2.2.1 Introduction to Deep learning

According to [20] and [21] deep learning is essentially a three-layer neural network(convolutional layers, pooling layers, and fully connected layers). Neural networks strive to emulate the functioning of the human brain by enabling it to learn from vast quantities of data., even though they fall far short of its capabilities. While a single-layer neural network may produce approximate predictions, additional hidden layers can help to optimize and improve accuracy.According to [22], one of the key advantages of deep learning is its ability to automatically learn hierarchical representations from raw data, eliminating the need for manual feature engineering, This feature makes deep learning particularly well-suited for many artificial intelligence (AI) apps and services which rely on deep learning. The objective of employing automation is to enhance efficiency by automating both analytical and physical tasks without the need for human participation. According to [23], there are a a lot of domains where deep learning has an important role in it (see Figure2.3) : • Semantic Parsing • Transfer Learning • Natural Language Processing • Computer

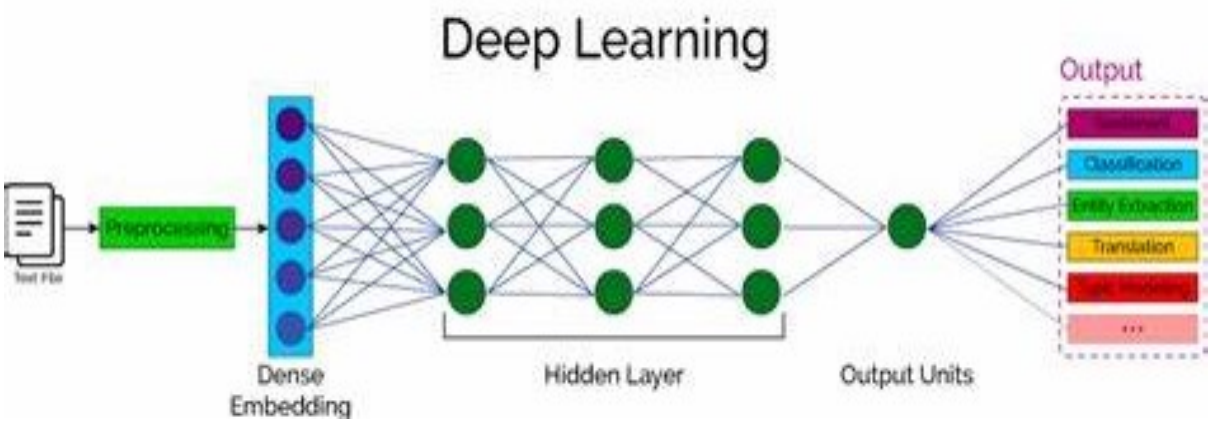



Figure 2.3: Deep learning for NLP

Vision However, according to [24], DL models require computational resources and large amounts of labelled data for training, which can be a challenge in certain domains. Additionally, interpreting the decisions generated by D learning models, often referred to as

the "black box" problem, is an ongoing research area. Nonetheless, deep learning continues to push the boundaries of AI and has opened up exciting opportunities for solving complex real-world problems. Its ability to learn intricate representations from data has made it a powerful tool in advancing the state of the art across various domains.

### 2.2.2 Deep learning approaches

According to [25], [26], [27], [28], and [29], this will get us into neural networks NN and how they get the data; extract the data, and push the data to the model. The first type of NN is Convolutional Neural Networks (CNNs), CNNs are widely used in computer vision tasks. They leverage specialized layers like convolutional and pooling layers to extract spatial features from images. CNNs demonstrate exceptional performance in various tasks, including image classification, object detection, and image segmentation. The second is Recurrent Neural Networks (RNNs), RNNs are designed to process sequential data by utilizing recurrent connections that allow information to persist across time steps. They are commonly used in natural language processing tasks, speech recognition, and language translation. Third is Generative Adversarial Networks (GANs): GANs consist of a generator and a discriminator network that compete against each other during training. GANs are employed for tasks like image generation, style transfer, and data augmentation. By employing these approaches, researchers and practitioners can use the power of DNN for an extensive spectrum of applications.

### 2.2.3 From Machine learning to Deep learning

According to [21], we now know and understand that ML is a subset of artificial intelligence (AI), and DL is a subset of machine learning. So every machine learning program is falling into the domain of AI programs but not vice versa. but the question then is are the approaches of ML and DL the same, the answer is yes, because every machine learning problem is an AI problem and deep learning is a subset of machine learning. We should not forget that deep learning is nothing more than methods that enhance machine learning algorithms to be more accurate and make some stages easy, like feature extractions, etc. The easiest takeaway for understanding the difference between machine learning and deep learning is to remember that deep learning is grouped under machine learning.

### 2.2.4 Machine learning vs Deep learning

According to [30] and [31], deep learning differs from traditional machine learning in the kind of data it uses and the learning algorithms it uses. The performance of deep learning differs from machine learning as the volume of the used data rises. (see Figure2.4) Deep learning algorithms do not work well when the amount of data provided is little because deep learning techniques require a significant amount of data to grasp it and produce higher performance than typical machine learning algorithms .

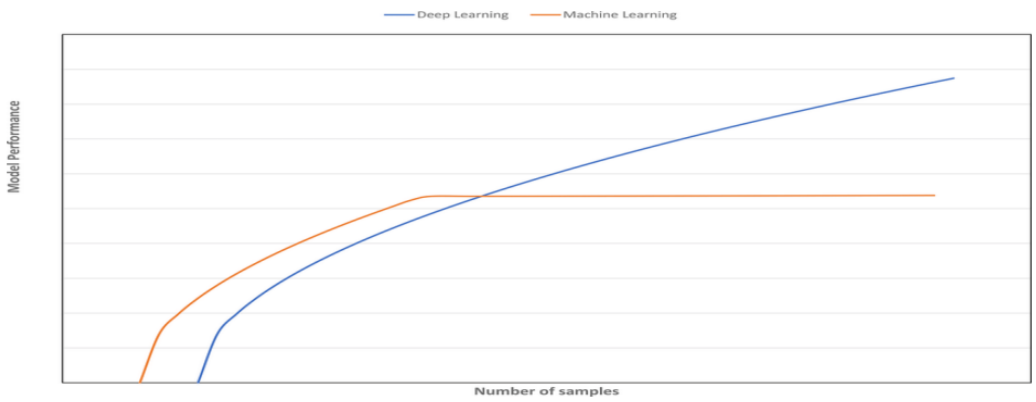


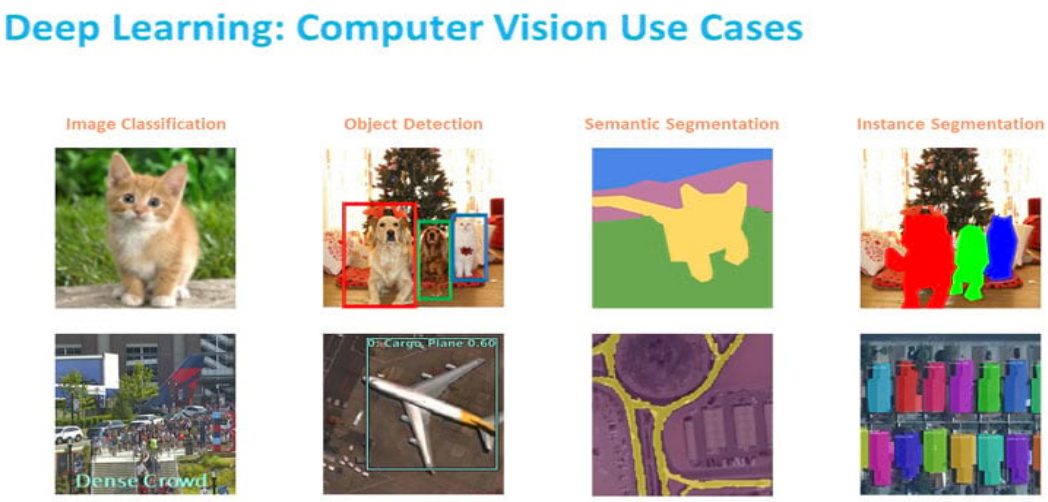

Figure 2.4: Model performance of deep learning vs. other machine learning algorithms as a function of number of samples.

centring

Figure 2.4: Performance of Deep learning vs Machine learning

### 2.2.5 Deep Learning for Computer Vision

Deep learning has a great role in Computer vision, according to [32]it has achieved remarkable breakthroughs in tasks such as object detection, image classification, image segmentation, and image generation. Deep learning models such as convolutional neural networks (CNNs) are designed to automatically learn hierarchically representations of visual data by progressively extracting and combining features at different levels of abstraction. According to [33] and [34] the convolutional layers in CNN can extract the features from images or video so machines or the block of code can benefit from these features in detecting objects, classifying them and so on (see Figure2.5) . One advantage of deep learning in computer vision is the capability to gain understanding from raw data, deep learning approaches automatically learn feature representations from the data itself. According to [35] and [36] training deep learning models for computer vision tasks require large datasets. The availability of datasets like ImageNet, COCO, and Pascal VOC can serve in training and also in testing which helps in computer vision tasks like detection and classification. By training on this huge amount of data, the models of the Deep learning can generalize well and achieve great performance on the range of visual recognition tasks. Also, deep learning for computer vision has benefited from advances in hardware and computational resources. Graphics processing units (GPUs) and specialized hardware like tensor processing units (TPUs) have accelerated the training and inference processes, making it feasible to train and deploy deep-learning models in real-world applications. With its ability to automatically learn complex features, deep learning has enabled accurate object recognition in images and videos, facilitated real-time object detection for autonomous vehicles and even in facial recognition and image segmentation. we can expect further innovations in the field of computer vision, unlocking new possibilities for applications in areas such as robotics, surveillance and augmented reality.




centring

Figure 2.5: Deep learning uses in Computer vision

## 2.3 Introduction to Computer Vision

According to [37] and [38] as humans, we perceive the three-dimensional structure of the world around us with apparent ease. Computer vision encompasses various approaches, including mimicking human vision, utilizing data and statistical methods, and leveraging geometric principles to address problems. It involves a combination of programming, modelling, and mathematics. Researchers in computer vision have been simultaneously advancing mathematical techniques to recover the three-dimensional shape and appearance of objects in videos or images. However, despite the mathematical complexities involved, it is often feasible to devise a mathematical representation of the problem for many vision-related challenges that either do not match realistic real-world conditions. What we need are algorithms that can help in defining the problem, take data from input and then output the result.

## 2.4 Object detection

Object detection is considered the goal of this project. According to [39] if we are given an image to analyze, we could try to apply a recognition algorithm to every possible sub-window in this image according to [40]. Algorithmic approaches of that nature are prone to being slow and error-prone. Consequently, a more effective strategy involves developing specialized detectors specifically designed to swiftly locate potential regions where specific objects are likely to be found.

### 2.4.1 What is Object detection

Object detection, as a sub-field of computer vision, focuses on localizing and classifying objects present in images or videos. This is accomplished by drawing bounding boxes around the detected objects, enabling us to precisely locate them within the specific input or track their movements throughout it according to [41]. Traditional object detection methods relied on features and machine learning algorithms. However, with the advent of deep learning, particularly convolutional neural networks (CNNs), object detection has witnessed significant advancements according to [42]. One of the primary obstacles in object detection is achieving a balance between accuracy and computational efficiency. Deep learning-based methods like CNN have made significant progress in addressing this challenge. They enable real-time object detection on resource-constrained devices by utilizing techniques like network pruning and quantization and leveraging hardware acceleration such as GPUs. Object detection techniques have also evolved to handle more complex scenarios. For instance, according to [43] some methods address object detection in crowded scenes and even in video sequences where temporal information is leveraged to improve detection accuracy. With advancements in deep learning and computer vision, object detection continues to progress, pushing the boundaries of what machines can understand in visual data. It plays a pivotal role in enabling machines to interact with the world and leads to many applications that benefit industries and society as well.

### 2.4.2 Object detection vs image classification

Object detection and image classification are two fundamental tasks in computer vision, but they differ in their objectives and methodologies according to [44]. Here, we will explore the key differences between object detection and image classification. Image classification involves passing an entire image through a classifier, such as a deep neural network, to obtain a corresponding label or tag. Classifiers consider the entire image as a whole but do not provide information about the specific location within the image where the tag is present, indicating the dominant object or scene category present in the image, but Object detection creates a bounding box around the detected object, it aims to identify and localize multiple objects within an image or video, providing not only the class labels but also the spatial coordinates (bounding boxes) of each detected object according to [45]. Classification offers certain advantages, particularly for tags that do not have well-defined physical boundaries, such as "blurry" or "sunny." However, when it comes to identifying objects with a distinct physical presence, such as a car, object detection systems tend to outperform classification networks consistently. Object detection systems excel in accurately identifying and localizing objects within an image, making them more suitable for tasks involving object recognition and localization. In terms of methodology, object detection typically involves a two-stage process: region proposal and classification. In the region proposal stage, potential object regions or bounding boxes are generated using techniques like selective search. (see Figure2.6) These proposals are then passed through a classification network that predicts the presence and class labels of objects

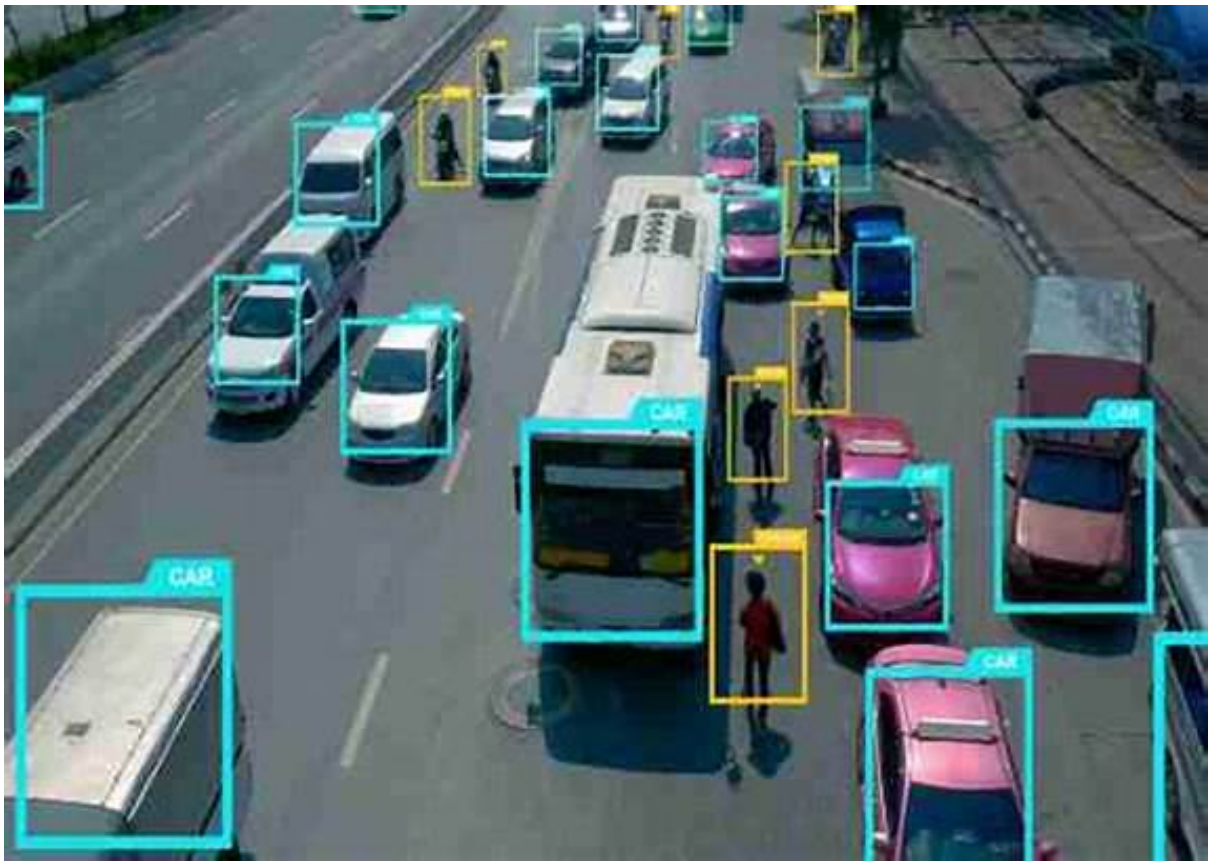



Figure 2.6: Example of classifying vehicles in image

within each proposed region. Object detection methods, such as Faster R-CNN and You look only once (YOLO) achieved impressive accuracy and real-time performance by leveraging deep learning and convolutional neural networks (CNNs). Image classification, on the other hand, focuses on training a CNN model to recognize and classify objects within images. The CNN processes the entire image and outputs a probability distribution over a predefined set of classes according to [46]. Notable CNN architectures for image classification include AlexNet, VGGNet, and ResNet. Image classification models are trained on large labelled datasets like ImageNet and can categorize images into various classes according to [47]. While both tasks share some similarities, such as leveraging CNNs and deep learning, object detection requires additional localization information, making it a more complex problem. Object detection provides more detailed information about the location and spatial extent of objects within an image, which is crucial for applications like autonomous driving, surveillance, and robotics.

### 2.4.3 Object detection vs image segmentation

Image classification Object detection and image segmentation are two crucial tasks in the field of CV that involve analyzing and understanding visual data (see Figure2.7) . While

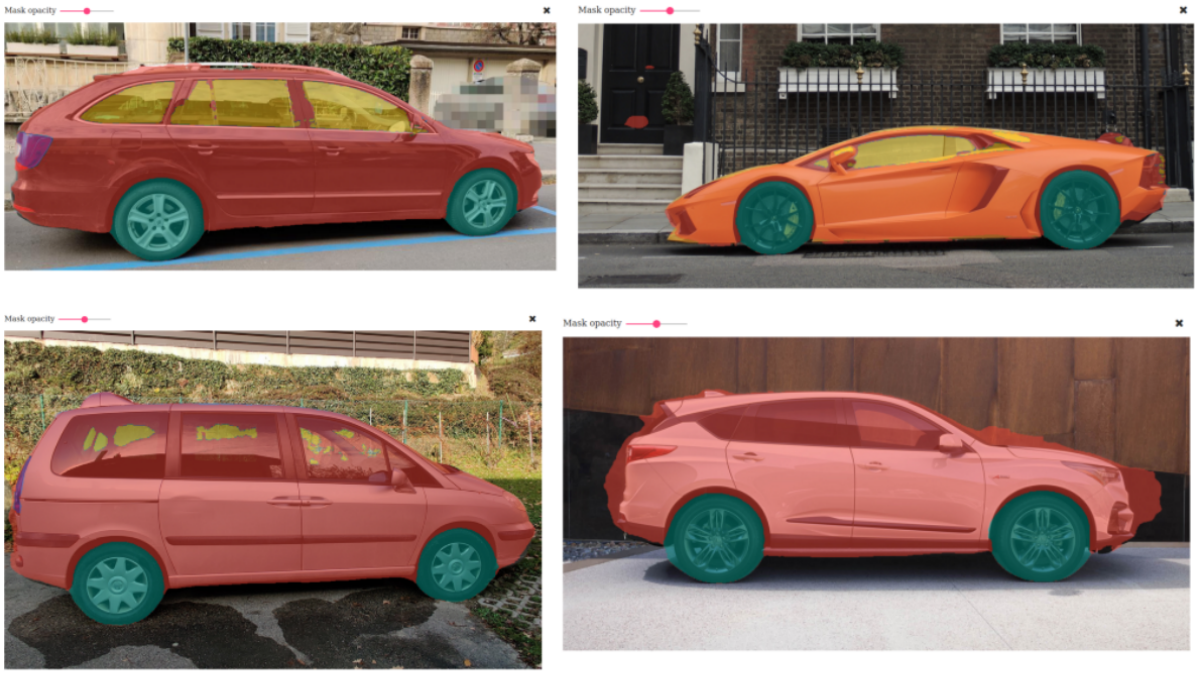



Figure 2.7: Object detection + image classification = instance segmentation

they share similarities, such as their reliance on deep learning techniques, there are distinct differences between object detection and image segmentation. image segmentation focuses on dividing the image into semantically meaningful regions, where each pixel is assigned a class label or a unique identifier. Image segmentation provides a more granular understanding of the image by capturing detailed object boundaries and segmenting objects from the background. Semantic image segmentation accurately labels all pixels associated with a specific class or tag, but it does not explicitly outline the boundaries of individual objects according to [48]. Image segmentation finds applications in tasks that require pixel-level understanding, such as medical image analysis, scene understanding, and semantic image editing. Unlike segmentation, object detection does not aim to precisely outline the boundaries of an object. Instead, it focuses on clearly defining the location of each object instance using bounding boxes. This approach allows for efficient and effective identification and localization of objects within the input we have. By integrating semantic segmentation and object detection, instance segmentation is achieved. This process involves initially identifying the object instances using object detection and subsequently segmenting each instance within the identified bounding boxes, commonly referred to as ROI.

### 2.4.4 Models of Object detection

Before the adoption of deep learning, conventional machine learning methods were predominantly used for object detection. Various commonly used techniques for object detection include the Viola-Jones method, scale-invariant feature transforms (SIFT), and histograms of oriented gradients. In contrast, deep learning-based approaches leverage neural network architectures such as RetinaNet, YOLO (You Only Look Once), CenterNet, SSD (Single Shot Multibox Detector), and region proposal methods like R-CNN, Fast-RCNN, Faster RCNN, and Cascade R-CNN. These deep learning models enable feature detection of objects and subsequent labelling. Object detection can be broadly categorized into two stages: (see Figure2.8) like Two-stage detectors: These detectors

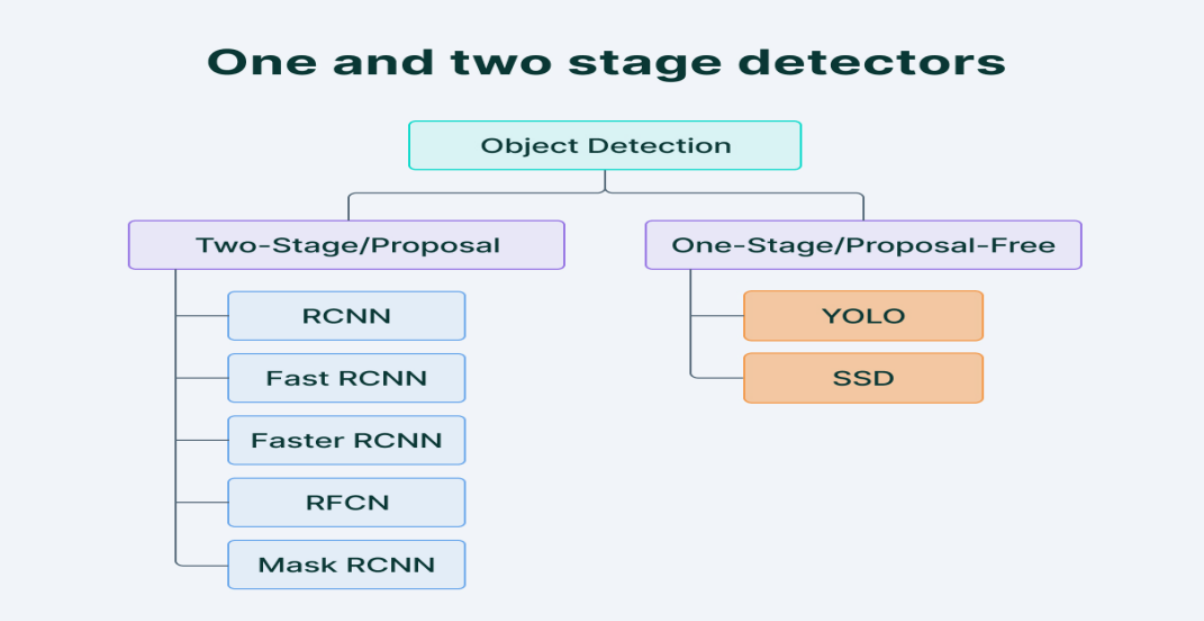



Figure 2.8: Models of one stage and two stage detectors

divide the object detection task into two stages. First, regions of interest (RoIs) are extracted, followed by the classification and regression of these RoIs to refine the object detection results.

Single-stage detectors: In contrast, single-stage detectors eliminate the RoI extraction process and directly perform classification and regression on candidate anchor boxes.

These stages represent different approaches to object detection, each with its advantages and applications within computer vision tasks[39] and [49].

### 2.4.5 Object detection projects

Object detection has many applications in life that help to detect objects in videos, images and video streams. In following examles object detection is used, so let‘s explain some

of them : Face Detection and Recognition: Face detection is an application of object detection (see Figure2.9). Projects like Facebook's, DeepFace and Google's FaceNet have employed deep-learning techniques to accurately detect and recognize faces in images and videos. These systems enable facial identification in social media platforms, photo tagging, biometric authentication and surveillance systems.

Figure 2.9: Example of object detection projects to detect faces in image

Facial Landmark Detection [50]: Facial landmark detection involves detecting and localizing key points on a face, such as the eyes, nose, and mouth. This technique is used for applications like facial expression analysis, virtual makeup and augmented reality effects. Projects like Adlib's facial landmark detector and OpenCV's facial landmark detection module have provided robust solutions for facial landmark localization. Pose Estimation: Pose estimation involves detecting and tracking the human body's joint positions and orientations in images or videos. It finds applications in gesture recognition, action recognition and human-computer interaction. Projects such as OpenPose, AlphaPose, and PoseNet utilize deep learning algorithms to estimate human poses accurately and in real time. Object Tracking [51]: Object tracking involves identifying and tracking a specific object or multiple objects over time in a video sequence. It is used in surveillance, activity recognition and robotics. Projects like SORT (Simple Online Real-Time Tracking) and DeepSORT utilizes object detection techniques to track objects across frames and maintain their identities. Car Detection [52]: Car detection involves detecting and localizing vehicles in images or videos, which is essential in traffic monitoring, autonomous driving and parking systems. Projects like the KITTI dataset and the Vehicle Detection and Tracking project in OpenCV provide datasets and algorithms for car detection and tracking. Document Text Detection [53]: Document text detection involves locating and extracting text regions from images or scanned documents. This technology is used in OCR (Optical Character Recognition) systems, document analysis and digitization projects. Projects like Tesseract OCR and the EAST (Efficient and Accurate Scene Text) algorithm focus on accurately detecting and extracting text from various document

types. Each one of them has its advantages and can be applied in the industry , Traffic Management, Sports Analytics and Augmented Reality.

## 2.5 Visual Recognition

Visual recognition, also known as Computer vision is an area of artificial intelligence that centres around equipping computers with the ability to comprehend and interpret visual information, such as images and videos. It encompasses the development of algorithms and models that can effectively analyze and extract valuable insights from visual data. By leveraging techniques from various disciplines, computer vision aims to enable machines to perceive and understand the visual world like humans, leading to advancements in tasks such as object recognition, image classification, video analysis, and scene understanding. Visual recognition encompasses tasks such as object detection, image classification, image segmentation, and scene understanding. The process of visual recognition involves several steps Data Acquisition: Visual data, such as images or videos, is captured using cameras or sensors. This data serves as the input for visual recognition algorithms. Preprocessing: The acquired visual data is preprocessed to enhance its quality and remove noise. Preprocessing techniques may include resizing, cropping, filtering, or normalizing the images to ensure consistency and improve analysis accuracy. Feature Extraction: In this step, relevant features are extracted from the preprocessed visual data. Various techniques, such as edge detection, texture analysis, or local feature descriptors, are applied to capture distinctive characteristics of objects, shapes, textures, or patterns within the images. Training Phase: Visual recognition heavily relies on machine learning, particularly neural networks. In the training phase, a neural network model is designed and trained on labelled datasets. The model learns to recognize patterns, objects, or classes by adjusting its internal parameters through a process called backpropagation and gradient descent. Neural Network Architecture: The neural network architecture used in visual recognition tasks can vary. Convolutional Neural Networks (CNNs) are widely employed due to their ability to extract spatial features and hierarchical representations from images. CNN layers, such as convolutional layers, pooling layers, and fully connected layers, are combined to create a deep neural network capable of learning complex visual representations. Inference Phase: Once the neural network model is trained, it is used in the inference phase to analyze new, unseen visual data. The model applies learned transformations and weights to the input images to make predictions, classify objects, detect features, or perform other specific tasks based on the problem at hand. Neural networks, especially deep learning architectures like CNNs, excel in capturing complex patterns and representations within visual data. The hierarchical structure of neural networks allows them to learn high-level features from low-level features, enabling better understanding and recognition of objects, textures, or scenes. Neural networks leverage their capability to automatically learn and adapt to visual data by adjusting the weights and biases of their interconnected neurons. This adaptability enables them to generalize well to unseen images or videos, improving the accuracy and robustness of visual recognition systems. Furthermore, the availability of large-scale labelled datasets and the

computational power to train and optimize deep neural networks have contributed to the success of visual recognition. Neural networks have revolutionized the field, surpassing traditional computer vision techniques by achieving state-of-the-art results across various visual recognition tasks, including object detection, image classification, semantic segmentation, and more. Other applications include image-based search and retrieval, content moderation in social media, quality control in manufacturing, facial recognition for access control and security, and even artistic style transfer in image editing. The potential of visual recognition extends to diverse fields, making it a crucial technology with profound implications for industry, research, and society as a whole [54, 55].

## 2.6 Literature Review

Vehicle counting and classification are essential tasks in various transportation applications, including traffic management, urban planning, and intelligent transportation systems. Over the years, researchers have explored different techniques and methodologies to accurately count and classify vehicles on roadways. This literature review presents a comprehensive overview of the key studies and advancements in the field.

One approach frequently used for vehicle counting is based on video processing. Cheng-Yang Fu and Alexander C. Berg proposed a method that utilizes background subtraction and blob analysis techniques to detect and track vehicles in real time using the SSD model. Their study demonstrated promising results in accurately counting vehicles under various environmental conditions [56]. Similarly, Kaur and Kaur (2018) employed video-based vehicle counting using image differencing and motion detection techniques. Their approach showed effectiveness in handling scenarios with varying lighting conditions and traffic densities [57].

In addition to video-based methods, there has been an increasing interest in using sensor-based technologies for vehicle counting. Li et al. (2017) explored the use of microwave sensors for vehicle counting in urban areas. Their study demonstrated that microwave sensors can achieve high accuracy and robustness in counting vehicles, particularly in situations where visual-based approaches face challenges such as occlusions or adverse weather conditions [58] Li et al. (2013) proposed a real-time vehicle detection and classification system that combined Haar-like features and AdaBoost classifier for vehicle detection. Their approach utilized Support Vector Machines (SVM) for vehicle classification, demonstrating the potential for integration into intelligent transportation systems. Yang et al. (2016) focused on developing a vehicle counting and classification system using magnetic sensors embedded in the road. By detecting changes in the magnetic field caused by passing vehicles, they achieved accurate counting and classification based on size and type.

The advent of deep learning techniques has revolutionized the field of vehicle analysis. Zhang et al. (2018) leveraged Convolutional Neural Networks (CNN) to develop a deep learning-based vehicle detection and classification system. By training the network on a large dataset of vehicle images, they achieved high accuracy in identifying and

categorizing vehicles into different classes. Similarly, Chen et al. (2015) explored vision-based sensors, employing image-processing techniques and machine-learning algorithms for vehicle counting and classification. Their approach demonstrated reliable results and provided valuable insights into traffic patterns.

In the pursuit of enhanced accuracy and robustness, Li et al. (2017) proposed an integrated vehicle counting and classification system that fused data from multiple sensors, such as video cameras and magnetic sensors. This fusion of sensor data improved the overall performance of the system and overcame the limitations of individual sensor technologies. Another innovative approach was introduced by Huang et al. (2019), who utilized Unmanned Aerial Vehicles (UAVs) equipped with cameras for aerial vehicle counting and classification. By capturing aerial images of traffic scenes and employing computer vision algorithms, they showcased the feasibility of UAV-based approaches for traffic analysis.

Furthermore, Wang et al. (2020) aimed to develop a smart city traffic management system that heavily relied on vehicle counting and classification. Their system integrated various sensors, including cameras, radar, and LiDAR, to collect comprehensive data on vehicle presence, count, and classification. Advanced algorithms processed the collected data, enabling real-time traffic monitoring and effective traffic management.

These projects collectively highlight the diverse range of approaches employed in vehicle counting and classification. From traditional methods utilizing image processing and machine learning techniques to the emergence of deep learning and sensor fusion, each project has contributed to the advancement of the field. These developments have paved the way for more accurate, efficient, and intelligent vehicle analysis systems, supporting traffic management efforts and informing transportation planning decisions.

Zhang et al. (2017) proposed a deep learning-based approach for vehicle detection and classification. The researchers utilized a Convolutional Neural Network (CNN) architecture, specifically a modified version of the Faster R-CNN (Region-based Convolutional Neural Network) model. This model combined object detection and classification into a single unified framework. The Faster R-CNN model consisted of two main components: a region proposal network (RPN) and a classification network. The RPN generated potential bounding box proposals for vehicles in an image, while the classification network classified the proposed regions into different vehicle classes. The researchers trained the model on a large dataset of labelled vehicle images to learn the representations and features required for accurate detection and classification.

The deep learning-based approach demonstrated excellent performance in terms of both accuracy and speed. It achieved high accuracy in detecting and classifying vehicles, even in complex and cluttered environments. The research highlighted the effectiveness of deep learning techniques in vehicle detection and classification tasks and provided insights into the potential applications of deep learning models in intelligent transportation systems.

Sun et al. (2019) proposed a vision-based system for vehicle counting and classification using a combination of computer vision techniques and machine learning algorithms. The

system aimed to provide accurate and real-time information on vehicle flow and types for traffic management purposes. The researchers employed image processing techniques such as edge detection and morphological operations to extract vehicle features and detect their presence in images or video frames. They utilized background subtraction algorithms to differentiate moving vehicles from the static background, enabling vehicle detection in dynamic traffic scenarios.

For vehicle classification, Sun et al. utilized machine learning algorithms such as Support Vector Machines (SVM) and Random Forests. These algorithms were trained on a dataset of labelled vehicle images, with features extracted from the detected vehicles. The trained models could then classify vehicles into different categories such as cars, buses, or motorcycles based on their visual characteristics.

The system demonstrated promising results in terms of accuracy and efficiency in vehicle counting and classification. It provided valuable data for traffic flow analysis, congestion management, and infrastructure planning. The research highlighted the significance of combining computer vision techniques and machine learning algorithms in developing effective and practical solutions for vehicle counting and classification in real-world traffic scenarios.

Sharma et al. (2019) proposed a vision-based approach for vehicle counting and classification using a combination of background subtraction and support vector machines (SVM). The objective of the project was to develop an efficient and accurate system for monitoring vehicle traffic in urban environments. In this approach, the first step involved background subtraction to detect moving objects in the video frames. The background model was created by considering a certain number of previous frames to estimate the static background. By subtracting the current frame from the background model, the moving objects, which represent vehicles, were detected.

Next, various features were extracted from the detected vehicles to classify them into different categories. Features such as size, shape, and color were computed to represent the visual characteristics of the vehicles. These features were then used as inputs to an SVM classifier, which was trained on a labelled dataset of vehicle images.

The SVM classifier was responsible for categorizing the vehicles into different classes, such as cars, motorcycles, buses, or trucks. The training phase involved learning the decision boundary that best separates the different vehicle categories based on the extracted features. Once trained, the SVM classifier could classify unseen vehicles in real time.

Finally, the system performed vehicle counting by tracking the detected vehicles over consecutive frames. Each vehicle was assigned a unique ID and its trajectory was recorded. By analyzing the trajectories, the system was able to accurately count the number of vehicles passing through a specified region of interest.

Kim et al. (2020) proposed a deep learning-based approach for vehicle counting and classification using a combination of a convolutional neural network (CNN) and a long short-term memory (LSTM) network. The objective of the project was to develop a system that could accurately count and classify vehicles in real time. The approach

consisted of two main stages: detection and classification. In the detection stage, a pre-trained object detection model, such as YOLO (You Only Look Once), was utilized to detect vehicles in the input video frames. The object detection model could identify the presence and location of vehicles in the scene.

Once the vehicles were detected, the classification stage began. A CNN-LSTM hybrid architecture was employed for vehicle classification. The CNN part of the architecture extracted visual features from the detected vehicle regions, capturing their distinctive characteristics. The LSTM network was then used to learn the temporal dependencies and context information from the sequence of frames, enhancing the classification accuracy.

The training phase involved collecting a large dataset of labelled vehicle images and videos. The CNN and LSTM networks were trained using this dataset to learn the features and temporal patterns associated with different vehicle categories. The model was optimized using techniques like backpropagation and gradient descent to minimize the classification error.

During the testing phase, the trained model was applied to real-time video streams. The detected vehicles were classified into different classes, such as sedans, SUVs, trucks, or motorcycles, based on the learned features and temporal context. The system also performed accurate vehicle counting by tracking the identified vehicles and maintaining a count of unique vehicles passing through a specified area of interest.

Li et al. (2018) proposed a deep learning-based approach for vehicle counting and classification using a single overhead camera. The objective of the project was to develop an efficient and robust system for monitoring vehicle traffic in real time. The approach employed a combination of image processing techniques and deep learning algorithms. It consisted of several key steps. First, the input video frames from the overhead camera were captured and preprocessed. The preprocessing step included tasks such as image enhancement, noise reduction, and frame stabilization to improve the quality and clarity of the frames.

Next, a vehicle detection module based on the Faster R-CNN (Region-based Convolutional Neural Networks) algorithm was applied to identify the presence and location of vehicles in each frame. The Faster R-CNN model was trained on a large dataset of annotated vehicle images to learn the features and characteristics of different vehicle types.

After vehicle detection, a vehicle tracking algorithm was employed to assign unique IDs to each detected vehicle and maintain their trajectories over time. The tracking algorithm utilized techniques like Kalman filtering and data association to accurately track the vehicles, even in complex scenarios with occlusions and overlapping.

Once the vehicles were detected and tracked, a vehicle classification module was applied to classify them into different categories, such as cars, trucks, buses, or motorcycles. The classification module was based on a deep convolutional neural network (CNN) architecture, such as VGG (Visual Geometry Group) or ResNet (Residual Network). The CNN model was trained on a large dataset of labelled vehicle images to learn the discriminative features for accurate classification.

The final step of the system was vehicle counting. The system maintained a count of the unique vehicles passing through specific regions of interest within the camera's field of view. This was achieved by analyzing the trajectories of the tracked vehicles and determining when they entered or exited the predefined counting zones. The counting results were continuously updated and displayed in real time.

The proposed system was evaluated on various real-world scenarios, including highways, intersections, and parking lots. The results showed high accuracy in vehicle detection, tracking, and classification, even in challenging conditions such as varying lighting, different weather conditions, and crowded traffic scenes. The system proved to be effective for monitoring and analyzing traffic patterns, enabling efficient traffic management and planning.

Overall, the experimental results demonstrate that SSD achieves competitive performance on the PASCAL VOC2007 dataset, outperforming Fast R-CNN and Faster R-CNN in terms of mAP while providing efficient and real-time object detection. Furthermore, Li and Zhang (2019) proposed a fusion-based approach that combines data from multiple sensors, including video cameras and infrared sensors, to enhance the accuracy of vehicle counting and classification [59].

Moreover, machine learning and computer vision techniques have been applied to vehicle counting and classification tasks. Zhang et al. (2020) developed a deep learning-based method using convolutional neural networks (CNNs) for vehicle counting. Their model achieved high accuracy and efficiency in counting vehicles from video streams. Similarly, Chen et al. (2019) used a combination of CNN and recurrent neural networks (RNNs) to classify vehicles based on their visual features. The integration of machine learning techniques has significantly improved the accuracy and automation of vehicle counting and classification systems [60].

Furthermore, Li et al. (2021) introduced a novel approach that leverages 3D LiDAR data for vehicle counting and classification. By capturing the spatial information of vehicles, their method achieved excellent performance in accurately counting and categorizing vehicles based on their size and type [61] In the field of horticulture, the recognition and counting of fruit, as well as the identification of other plant vehicles, can provide valuable information for tasks such as estimating yield, guiding robotic picking, precision spraying, and pruning. Traditional methods for fruit recognition based on 2D images have achieved good results by utilizing colour, shape, and texture features. These methods involve the use of different colour spaces, shape descriptors like the histogram of oriented gradient (HOG), Hough transformation, shape context algorithm, template matching, and machine learning classifiers such as relevance vector machines and decision trees.

More recently, deep learning approaches based on convolutional neural networks (CNNs) have been used for fruit recognition, but they require large amounts of data and time-consuming training. Fruit counting, an important aspect of yield estimation, has been primarily performed using clustering algorithms based on Euclidean distance or fuzzy C-means. Thermal images have also been utilized for fruit identification.

Compared to 2D images, 3D point clouds provide more comprehensive plant information. Three methods are commonly used to acquire 3D point clouds: (1) Expensive and high-precision equipment like 3D laser scanners, laser radar, and ultrasonic devices, which are suitable for large-scale scene reconstruction but limited in general applications due to cost; (2) Low-cost RGB-D cameras, which are used for small-scale scene reconstruction and have higher accuracy in such scenarios; (3) Reconstruction based on multiple 2D images, which utilizes ordinary cameras but is influenced by algorithmic and camera resolution limitations.

Researchers have commonly employed SVM classifiers based on colour and 3D shape features for vehicle classification and fruit recognition using 3D point clouds. Some studies have focused solely on using 3D features for fruit identification. Clustering algorithms have also been employed for individual fruit detection.

In that research paper, a method for vehicle classification and counting is proposed, utilizing the fusion of shape and colour features extracted from 3D point clouds. Shape descriptors, which have been predominantly used for industrial robot identification, are applied in the agricultural context. The proposed method can be employed for various tasks in orchards, including pruning, spraying, and yield estimation, under natural environmental conditions. In that article, the authors acquired 3D point cloud data of pomegranate trees using an RGB-D camera (RealSense D435, Intel, California, USA) at China Agricultural University. The data collection period was from July 20, 2019, to August 26, 2019. Six mature pomegranate trees were scanned, and the point cloud reconstruction was performed using Dot3D software. The data processing and analysis were conducted using Matlab R2019a.

The preprocessing steps for the 3D point cloud data are illustrated in Figure 3. The original point cloud, consisting of 13,244,046 points, was obtained from the RGB-D camera. The background was removed, reducing the number of points to 7,116,143 (Fig. 3b). Down-sampling was performed, resulting in a point cloud with 961,003 points (Fig. 3c). Outlier filtering was applied to refine the point cloud (Fig. 3d).

For vehicle classification, a Support Vector Machine (SVM) classifier was used. The results of the SVM classifier for different vehicles were obtained, but the specific results were not mentioned in the provided text.

In conclusion, this paper focused on acquiring and preprocessing 3D point cloud data of pomegranate trees using an RGB-D camera. The data were processed by combining RGB colour information with shape features. An SVM classifier with colour and 2-scale shape features (k1 = 200, k2 = 400) was utilized for vehicle classification, achieving different AUC values for fruit, non-fruit, branch, and leaf. Additionally, K-nearest neighbours (KNN) smoothing with k = 50 was applied for further analysis, but specific details were not provided in the given text.

# Chapter 3

# Neural Network and Convolutional Neural Network

This chapter includes an explanation of the Neural network (NN) and how it works also the Convolutional neural network (CNN) and how it works also which will help in understanding how algorithms detect the object and easily classify them.

## 3.1 What is Neural Network

A neural network is a computational framework that takes inspiration from the intricate organization of the human brain. It is a subset of a machine learning algorithm that learns from data to recognize patterns, make predictions, or perform tasks. Neural networks consist of interconnected neurons, also known as nodes, organized in layers. Each neuron is connected to other neurons through weighted connections (see Figure3.1) . These weights determine the strength of the connection and influence the contribution

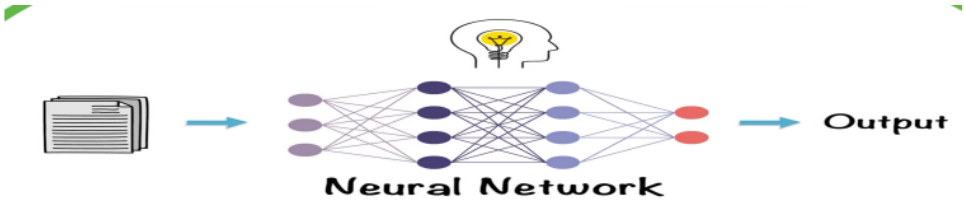



Figure 3.1: Simple shape of how neural network works

of the input to the neuron's output. NNs have proven to be powerful tools for solving many problems like detection and continue to drive advancements in artificial intelligence applications.

### 3.1.1 Simple way of how neural network works

NN take the data as input trains themselves to recognise the pattern in this data and then predicts the output of these data (see Figure3.1) .

### 3.1.2 Layers of neural network

The fundamental data structure in neural networks is the layer, layers play a crucial role in organizing and processing information. Each layer consists of a group of neurons or units that perform computations on the input data. You can think of layers as the LEGO bricks of deep learning. They are mainly 3 layers [62]. Let's explore the types of layers in neural networks. Input layer: The input layer is responsible for receiving the initial data. It acts as the entry point of the neural network. The number of neurons in the input layer corresponds to the number of input features or dimensions of the data. Each neuron in the input layer represents a specific feature and passes its value to the next layer. Hidden layers: Hidden layers are layers between the input and output layers. They are responsible for performing computations and extracting higher-level representations from the input data. The number of hidden layers and neurons in each layer are design choices that depend on the complexity of the problem at hand. The more layers and neurons, the greater the network's capacity to learn complex patterns. Output layer: The output layer is the final layer of the neural network, responsible for producing the network's predictions or outputs. The number of neurons in the output layer depends on the nature of the problem. For binary classification, a single neuron with a sigmoid activation function is commonly used. For multi-class classification, the output layer may have multiple neurons, where each neuron represents a class and applies a function called softmax activation function to produce class probabilities. In regression tasks, the output layer can have a single neuron or multiple neurons, depending on the number of output dimensions [63].

### 3.1.3 Loss function

It quantifies the difference between the predicted outputs of the network and the true or desired outputs. The goal of training a neural network is to minimize this loss function, effectively optimizing the network's performance. The choice of an appropriate loss function depends on the nature of the problem being addressed. Here are a few of them that are used in NN. Mean Squared Error (MSE): MSE is a popular loss function for regression tasks. It calculates the average squared difference between the predicted and true values. MSE penalizes large deviations between predictions and targets, providing a method of how the network's predictions match the ground truth. Binary Cross-Entropy: Binary cross-entropy is commonly used for binary classification tasks. It quantifies the disparity between the predicted probabilities and the actual binary labels. It encourages the network to assign higher probabilities to the correct class and lower probabilities to

the incorrect class. Sparse Categorical Cross-Entropy: Sparse categorical cross-entropy is similar to categorical cross-entropy but is used when the true labels are integers instead of one-hot encoded vectors. It is commonly employed in situations where each input has a single correct label. Selecting the suitable loss function is crucial as it determines the network's learning behaviour and the type of outputs it produces. it ensures that the network is trained effectively suited for the specific task [64] and [65].

### 3.1.4 Training and Optimization

Training and optimization are essential components to build effective neural networks. Training involves iteratively adjusting the network's parameters to minimize the loss function and improve its performance on a given task. Optimization algorithms play important role in this process by determining how these parameter updates are performed. In the training process, neural networks are typically trained using large datasets [66]. The training data is divided into subsets and each part contains a subset of the training examples. This allows for efficient computation. The choice of part size depends on the available resources and the specific characteristics of the dataset. Then we go to Forward Propagation where the network takes the input data and processes it through its layers to generate predictions. Every neuron calculates a weighted sum of its input signals, applies an activation function and passes the output to the next layer. The predictions are derived from the output layer of the network. After obtaining the predictions, the loss function is computed to quantify the discrepancy between the predicted outputs and the true targets (see Figure3.2) . The loss serves as a measure of the network's performance and guides the

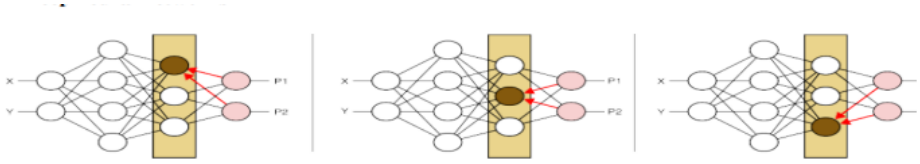


Figure 3.2: Train Neural network on some data and use this to predict the output

training process. In the optimization process, some optimization algorithms determine how the network's parameters are updated based on the computed gradients like the gradient descent algorithm which updates the parameters by incrementally moving in the direction of the negative gradient through small steps. It iteratively adjusts the parameters to find the optimal values that minimize the loss function [67]. Training and optimization in neural networks require careful consideration of some factors, such as the choice of the optimization algorithm. Iterative adjustments to these factors allow us to develop effective neural networks that can be performed on desired tasks [68].

### 3.1.5 Backpropagation

Once we have established our neural network by determining the number of layers, specifying the loss function, and initializing the weights, we can proceed to train the network

using our sample data. After the Neural network gets the output it‘s starting with these outputs and working way back through the network towards the inputs, as shown in Figure 3.2. This procedure is known as backpropagation [69]. Backpropagation is a fundamental algorithm used to compute the gradients of the loss function concerning the network's parameters. It involves propagating the error backwards from the output layer to the input layer. The gradients are then used to update the parameters and minimize the loss function during the optimization phase [70].

### 3.1.6 Example on Neural Network

we still don‘t know what is exactly going on inside the layers and how the Neural network detects and classify the object, so let‘s see this example. So far we see the Convnet as a black box where input images of raw pixels are coming in one side, going to many layers of convulsion and pooling and on the outside we ultimately obtain some of the scores. So far we see the Convnet as a black box where some input images of raw pixels are coming in one side, going to many layers of convulsion and pooling and on the outside we end up with some of the scores. Suppose we have a dataset of handwritten digits (0-9) and our goal is to build a neural network that can correctly classify these digits. We will use a feedforward neural network with a single hidden layer. Data Preparation: We start by preparing our dataset. Each handwritten digit is represented as an image, which is converted into a numerical format. The images are typically represented as a grid of pixel values, where each pixel represents the intensity of the grayscale or the RGB channels. We flatten each image into a vector and normalize the pixel values to a range between 0 and 1.

Network Architecture: Our neural network will have an input layer, a hidden layer, and an output layer. For this example, let's assume we have 784 input neurons (corresponding to the flattened image size), 100 neurons in the hidden layer, and 10 output neurons (one for each digit) (see Figure3.3) .

Initialization: We initialize the weights and biases of our neural network randomly. These weights and biases will be adjusted during the training process to improve the network's performance.

Forward Propagation: In the forward propagation step, we pass the input through the network to generate predictions. neuron in the hidden and output layers performs a computation based on the inputs it receives, the corresponding weights, and an activation function.

For each neuron in the hidden layer, we calculate the weighted sum of its inputs (pixel values) by multiplying them with the corresponding weights and adding the bias term. Then, we pass the sum through an activation function such as the sigmoid or ReLU to introduce non-linearity and generate the neuron's output.

The outputs from the hidden layer neurons serve as inputs to the output layer. We repeat the weighted sum and activation function computation for each output neuron to generate the final predictions for each digit.

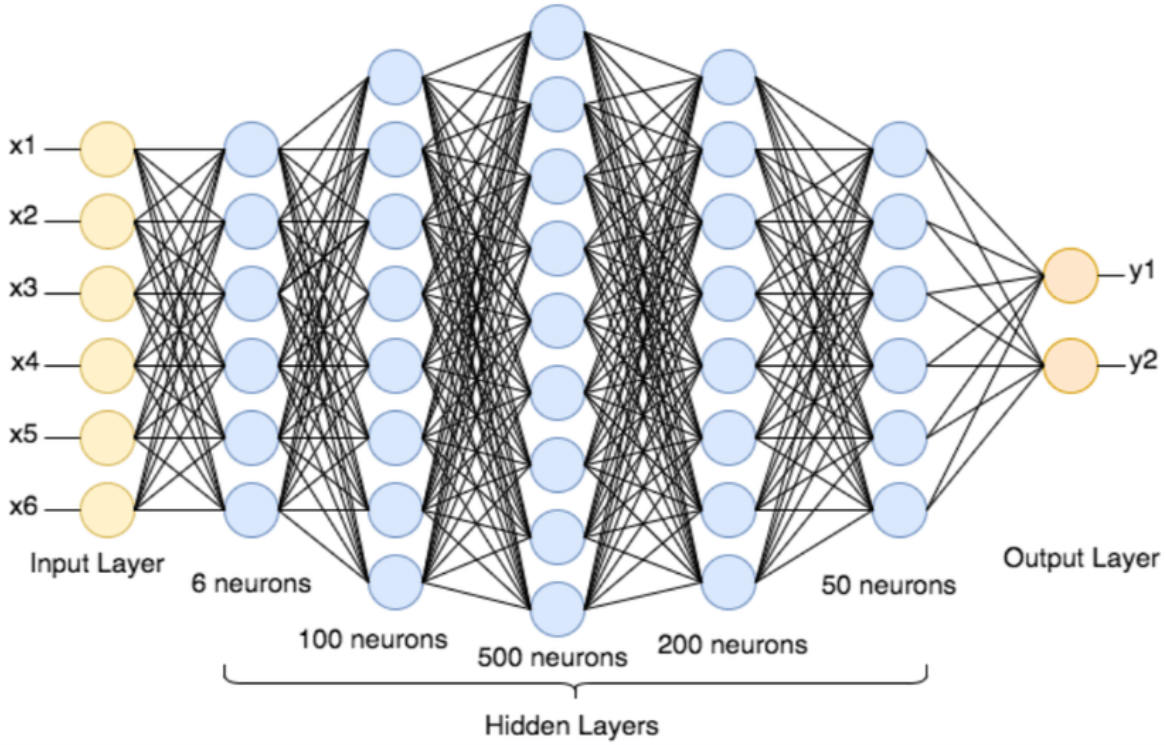



Figure 3.3: Layer of Neural network as black box

Loss Calculation: We compare the predicted output with the true label of the digit and calculate a loss value. The loss represents the difference between the predicted and actual values and quantifies how well the network is performing.

Backpropagation: Backpropagation is the key step in training a neural network. It involves calculating the gradients of the weights and biases concerning the loss. These gradients indicate the direction and magnitude of adjustments required to minimize the loss.

Starting from the output layer, we propagate the error backwards through the network, updating the weights and biases along the way. This is done by the chain rule of calculus to compute the partial derivatives of the loss concerning the weights and biases at each layer.

Weight and Bias Update: With the gradients calculated, we update the weights and biases using an optimization algorithm such as stochastic gradient descent (SGD). SGD adjusts the weights in the inverse direction of the gradients, taking into account a learning rate that determines the step size of the update.

Iterative Training: We repeat steps 4-7 (forward propagation, loss calculation, backpropagation, weight, and bias update) for multiple iterations or epochs, allowing the trained network to adjust its parameters and improve its performance.

Prediction: Once the training is complete, our neural network is ready to make predictions on new, unseen images. We pass the image through this network using forward

propagation, and the output neuron with the highest activation represents the predicted digit.

By iteratively adjusting the weights and biases based on the training data, the neural network learns to recognize patterns and features that are relevant for classifying the handwritten digits. Through this process, the network improves its accuracy in predicting the correct digit label for unseen images.

This example provides a simplified understanding of how a neural network works. In practice, neural networks can have multiple hidden layers, different activation functions, and various optimization techniques. However, the underlying principles of forward propagation, backpropagation, and iterative training remain the same.

## 3.2 Convolutional Neural Network

Convolutional neural networks, also known as convents are a type of deep learning model used in computer vision applications. A convolutional neural network (CNN) receives an input image and assigns significance (adjustable weights and biases) to different elements/objects within the image, and is capable of distinguishing between them. The preprocessing required in a CNN is much lower as compared with other classification algorithms. The fundamental component of convolutional neural networks is the utilization of the convolution operation for processing data.

### 3.2.1 Convolutional operation

Convolution operation involves the application of a filter or kernel to an input image or feature map to extract relevant features and capture spatial relationships. The convolution operation operates by sliding the filter over the input image, computing the element multiplication between the filter and the corresponding input pixels [71]. The resulting values are summed, and the sum represents the output value at that specific location. This process is repeated for each location, producing an output feature map. The convolution operation is effective in capturing local patterns and spatial hierarchies in images. By using different filters, CNNs can learn to detect various features, like edges, textures, and shapes. Moreover, the shared weights of the filters enable parameter sharing, which reduces the number of the used parameters in the trained network, making it computationally efficient and enabling generalization to similar patterns across the entire image. The size and stride of the filter, in addition to the padding which is applied on the given input, influence the output feature map's size. By adjusting these parameters, CNNs can control the level of information compression and field size of the learned features [72].

### 3.2.2 Layers of CNN

ConvNets are composed of multiple layers. multiple layers that perform different operations on the input data. The arrangement and combination of these layers vary based

on the specific architecture and the task. Let's see what are the basic layers in CNN and what is the definition for each one of them.

1- Convolutional layer : The fundamental element of a ConvNet is the convolutional layer, which employs a collection of adaptable filters or kernels to process the input image or feature map. In a convolutional layer, each filter is convolved with the input data by performing element-wise multiplications and summing the results. This process involves sliding the filter across the input spatially, extracting local information at each location. The output of the convolutional layer is a feature map, which represents the presence and location of specific features in the input data. The filters in the convolutional layer learn to detect various visual patterns such as edges, textures, or shapes. Through the training process, the network learns to adjust the filter weights to capture meaningful and discriminative features. The shared weights of the filters enable parameter sharing, making the convolutional layer computationally efficient and enabling the Conv network to generalize across different spatial locations. By stacking multiple convolutional layers, ConvNets can abstract representations of the input data. The depth of the convolutional layers helps the network capture both low-level and high-level features, understanding the visual information and enabling effective feature extraction for computer vision tasks.

2- Pooling layer : The pooling layer helps reduce the spatial dimensions of the feature maps, resulting in downsampling. It helps in capturing the most salient information while reducing computational complexity and improving the network's ability to generalize. The pooling layer operates by dividing the input feature map into non-overlapping or overlapping regions called pooling windows or kernels. The most common pooling techniques are max pooling and average pooling. Max pooling selects the maximum value within each pooling window, while average pooling computes the average value. These operations summarize the information within each region, reducing the spatial dimensions of the feature map while retaining the most important features. it provides translation invariance, implying that the network can recognize a specific feature regardless of its precise location in the input. This property enables ConvNets to capture spatial hierarchies and detect features irrespective of their exact position. also, the pooling layer decreases the size of the feature maps, which can be beneficial for memory efficiency and computational speed. By downsampling the feature maps, the pooling layer discards redundant or less relevant information, focusing on the most distinctive features. This simplification of the data helps prevent overfitting and makes the network more computationally efficient. However, it's important to notice that pooling also results in information loss since it discards some fine-grained details. Therefore, the choice of pooling technique, pooling window size, and stride should be carefully considered. pooling with large windows may lead to excessive loss of information and reduced spatial resolution, which can be detrimental to tasks such as object localization.

3- Fully Connected layer : The last layer of them is the Fully connected layer, fully connected layer, also known as the dense layer, is a key component of many neural network architectures. It connects every neuron in the current layer to every neuron in the subsequent layer, forming a fully connected network structure. Its purpose is to capture high-level abstract representations and make predictions based on the learned

features. Each neuron in the fully connected layer receives inputs from all the neurons in the previous layer. These inputs are multiplied by learnable weights associated with the link between these neurons. The weighted inputs are then summed, and an activation function is applied to produce the output of each one of neurons. The fully connected layer is responsible for learning complex relationships and combining features from different parts of the input. It can capture global patterns and dependencies that are not captured by the convolutional and pooling layers alone. In classification tasks, the output of the fully connected layer is often fed into a softmax activation function to obtain class probabilities. The softmax function ensures that the predicted probabilities sum to one, permitting the network to make confident predictions about the input data. However, they have some limitations. One limitation is their high computational cost and a huge number of parameters, especially when they are dealing with high-dimensional inputs. Additionally, they do not explicitly exploit the spatial structure of data, which is crucial in computer vision tasks. To address these limitations, modern ConvNet architectures often use a combination of convolutional layers and FCL. The convolutional layers extract local features and capture spatial relationships, while the fully connected layers capture high-level abstract representations and make predictions based on the learned features [73, 74, 75, 76, 77, 78].

### 3.2.3 Backpropagation through Convolutional layer and Pooling layer

Backpropagation allows for the efficient computation of gradients and updating of the network's parameters. Every output feature map is generated by convolving an image or a feature map from the previous layer with a filter kernel, whose weights are learned through backpropagation. The weights in the filter kernel are shared for a specific input–output feature-map combination. During backpropagation through a convolutional layer, the gradients are computed concerning the weights and biases of the layer. The gradients are then propagated backwards through the layer using the chain rule of calculus. This involves calculating the gradients of the layer's inputs by convolving the gradients of its outputs with the flipped version of the layer's filters. This operation is known as the convolution operation, but with the filters flipped. In the case of a pooling layer, the backpropagation process is slightly different. Since pooling layers do not have learnable parameters, the gradients are only computed concerning the input feature maps. The gradients are then propagated back to the earlier layer by using the pooling operation in reverse. This involves distributing the gradients to their original locations within the pooling windows based on the maximum value or average value selected during the forward pass [79, 80].

### 3.2.4 Example on CNN

In the Conv Layer, we have our image we divide it into pixels like 4*4 or 5*5 and we have a filter. this filter is just a matrix with dimensions like 3*3 and we apply this filter in

the image by multiplying every individual pixel by the value in the filter and sum them at the end. Suppose that we have filter 3*3 [[1,1,1] , [0,1,1] , [0,0,1]] So result = 1*1 + 1*0 + 1*1 + 0+1 +0 +0 +0+1 = 4 and we keep doing this till we cover the whole pixels of the image. As we saw in the previous example we could get info from the pixel of the image but some pixels are used once and this is not sufficient for us to extract more info because we apply the filter on those pixels once and don't completely make use of it. from this situation, the padding idea comes. Padding is simply adding a layer to the image (adding additional rows and columns) so that we can extract more info from the pixel. We can define the value of padding by any value we want. There are 2 types of padding the first one is valid padding and when we write it we tell the code that we don't want to add layers to the image, the next one is the same padding and it's most used in code. So as we see we move one step from one pixel to another pixel and this is called stride (see Figure3.4) . A stride is a tool that we use to control how many steps we want to go so

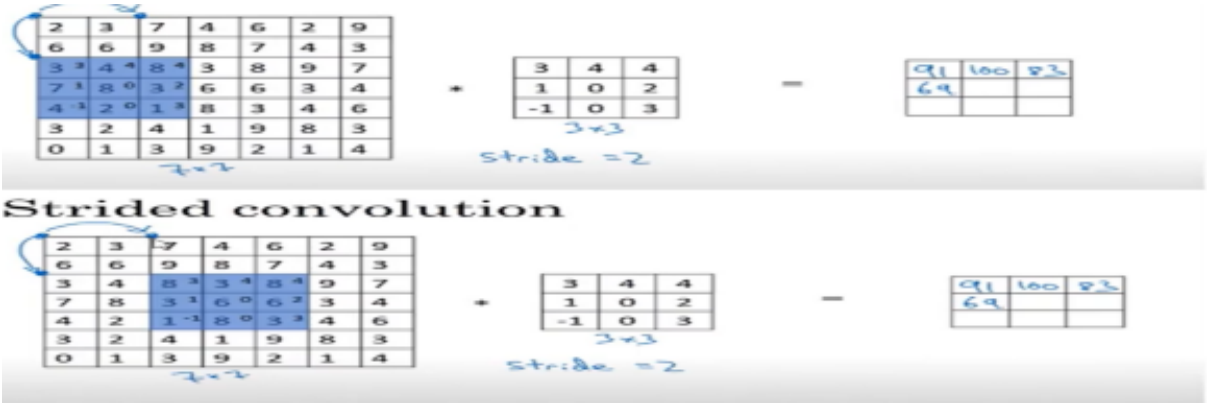



Figure 3.4: Stride determines the steps of how you want to apply the filter on pixel

we use it to determine how long will we apply the filter on the pixels. Then we have the next layer called Relu Layer, Relu layer in simple words takes the make value between the 0 and the value after we apply the Convolution layer so if the first pixel is 0.77 so the new value will also be 0.77 as its greater than 0 and if the value is negative so we will put 0. We go then to the Pooling layer and in this layer we downsample the previous layer so that we decrease the volume of the representation so that we reduce the number of parameters and computations (see Figure3.5) . First, we define the size of pooling there are 2 types of Pooling the max one which takes the highest value within the defined size and the other type is average pooling which takes the average in the defined size. For example, if the size was 2x2 then the new values will be. In the final layer which is the Fully Connected layer, it takes the values from the pooling layer make them flatten into a one-dimensional array and put them in one column and then take these values and put them in NN.

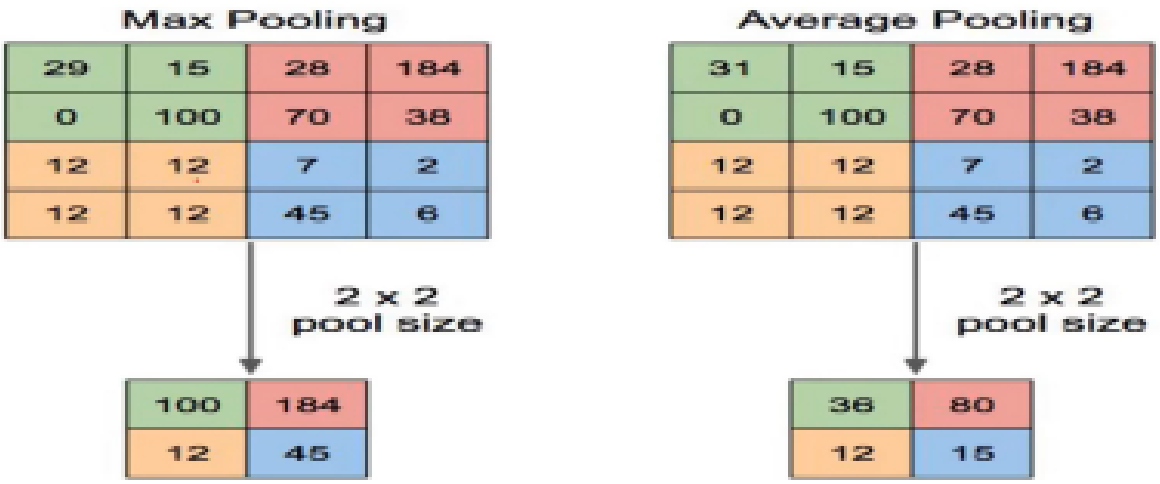



Figure 3.5: Max pooling takes the max value in our determined size and avg pooling takes the average on the determined size

# Chapter 4

# Hardware and GPU

This chapter will provide us with an in-depth analysis of the role of hardware and Graphics Processing Units (GPUs) in the field of CV. We explore the significance of hardware accelerators and GPUs in enhancing the performance of computer vision systems. We delve into various hardware architectures and GPUs and discuss their contributions to accelerating deep neural networks.

## 4.1 Hardware and GPU role in Computer vision

The role of GPUs in computer vision is vital, as they significantly accelerate various computational tasks involved in image and video processing. Here's a brief overview of some roles : GPUs for Parallel Computing: Graphics Processing Units (GPUs) excel at performing parallel computations, making them well-suited for the parallelizable nature of computer vision algorithms. NVIDIA, a prominent GPU manufacturer, provides technical documentation and resources on GPU architecture and parallel computing capabilities. Deep Learning and Convolutional Neural Networks (CNNs): Deep learning, particularly CNNs, has revolutionized computer vision. Training large-scale CNN models are computationally intensive, but GPUs can dramatically speed up the process. The book "Deep Learning" by Goodfellow et al. provides in-depth coverage of deep learning concepts and GPU utilization [2]. GPU Programming: To leverage GPUs effectively, developers can use programming frameworks like CUDA (Compute Unified Device Architecture) provided by NVIDIA. "CUDA by Example" by Sanders and Kandrot offers a practical introduction to GPU programming using CUDA [81, 82]. GPU-Accelerated Libraries: Various GPU-accelerated libraries, such as OpenCV and cuDNN (CUDA Deep Neural Network library), enable more efficient computer vision algorithms. The book "Practical OpenCV" by Brahmbhatt explores GPU acceleration techniques in OpenCV applications. Real-Time Applications: GPUs play a crucial role in real-time computer vision tasks, where low latency is essential. [83] shows cases of the utilization of GPUs for real-time vision algorithms using OpenCV and Python. Parallel Processing with GPUs: GPUs excel at parallel processing, allowing for significant speedups in computer vision

algorithms. NVIDIA's CUDA programming model facilitates harnessing the power of GPUs for parallel computations. GPU Cluster Computing: Large-scale computer vision systems may require GPU cluster computing. [84] discusses distributed GPU computing for scaling up computer vision models. Real-Time Computer Vision: Real-time computer vision applications demand fast processing. GPUs enable parallel execution, leading to real-time performance in tasks like video analysis, tracking, and augmented reality. [85] explores GPU optimization techniques for real-time computer vision applications. In conclusion, hardware, particularly GPUs, plays a dynamic role in computer vision. GPUs accelerate deep learning tasks, enable real-time performance, and facilitate efficient data processing. They offer scalability options and energy efficiency, preparing them for various computer vision applications. As computer vision continues to advance, the role of hardware, including GPUs, will remain instrumental in driving further breakthroughs in the field [86, 87].

# Chapter 5

# Models used in Vehicle count and Classification

This chapter will introduce all the methods which are used in this project is for counting and classifying vehicles in video or image. it will explain how object detection is working and will provide the code of each method.

## 5.1 Region-based Convolutional neural network (R-CNN)

We Use object detection when we have multiple objects in the image and we do not know the number of them. the idea is to take different crops from the input image then feed it to our CNN and CNN classifies that input crop. But the problem lies in how we choose the crop because there could be any number of Objects in the image and these objects can appear at any location in the image and could appear at any size in the image and could appear at any aspect ratio so if we going to do a brute force in the sliding window approach we will end up having 1000 and tens of 1000 of different crops to tackle this problem with brute force and every one of those crops is going to be fed through CNN and this will be completely Computationally intractable (see Figure5.1) . it uses traditional signal processing , image processing to make a list of proposals. So given an input image, a region proposal network will give you something like 1000 boxes where an object might be present so we can imagine that maybe we do look for some edges in the image and try to draw boxes that contain closed edges (see Figure5.2) . These types of image processing but this region proposal will look for the object in the image and give us some sets of candidate proposal regions where objects might be potentially found. Most of regions will not be true objects but if there is an object in the image then it does tend to get covered by these region proposals, So now rather than applying our classification network to every possible location and scale in the image instead we can use first apply one of these RP to get some set of PR where objects are likely located and then applying CNN

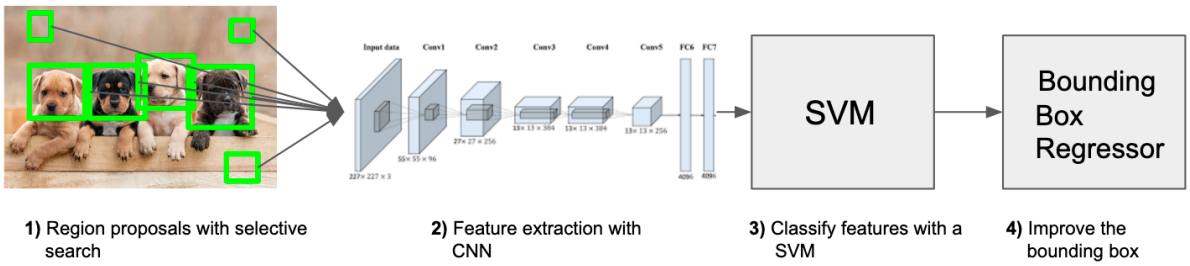
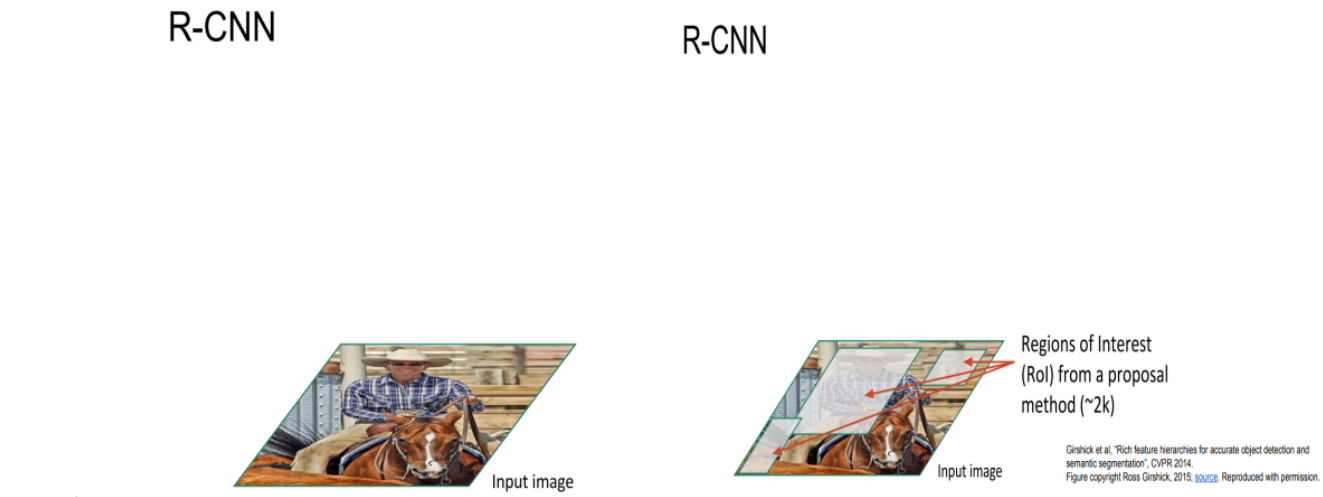



Figure 5.1: Shape of R-CNN and how it works




centring

Figure 5.2: Taking the input image then draw rectangles around the ROI

for classification to each of these PR and this approach prove to be more computationally feasible compared to attempting to handle all possible locations and scales and R-CNN does exactly that, so given an input in this case we will run RPNetwork to get our proposals, these are called Region of Interest ROI. One of the problems is that these regions in the input image could have different sizes but if we are going to run them all through CNN our CNN for classification wants images of the same input size typically due to Fully Connected layers (see Figure5.3) , so we need to take each of these RPs and wrap them to that fixed square size is expected as input, So we will crap out those RP to that fixed size and run each of them through a CNN and then use SVM (support vector machines that analyze data for classification and regression analysis.) to make a classification decision for each of those crops to predict categories. But R-CNN have some problems like Size and the runtime as its quite slow (see Figure5.4) . After we saw the idea of how R-CNN works, we can move forward to the code which helps us to detect Vehicles in the image. we first import the required libraries, then we can prepare the dataset and preprocess the images inside this dataset then train the R-CNN model but this consumes a lot of time and need you to have a large dataset of images like 20k images

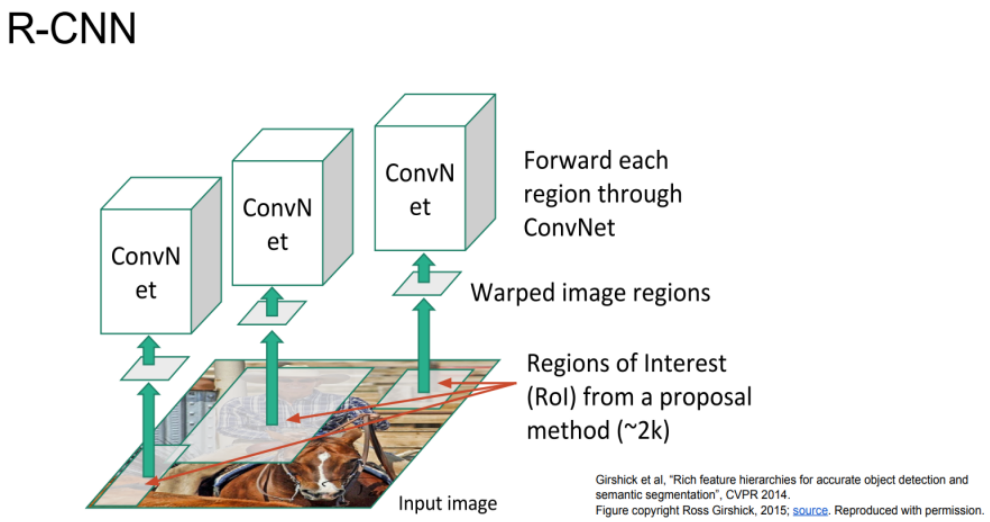



Figure 5.3: Running these rectangles on CNN

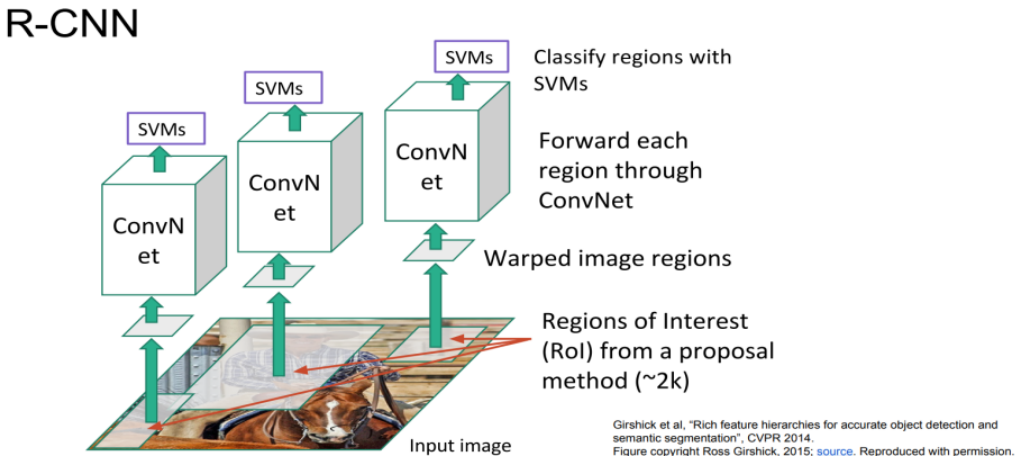
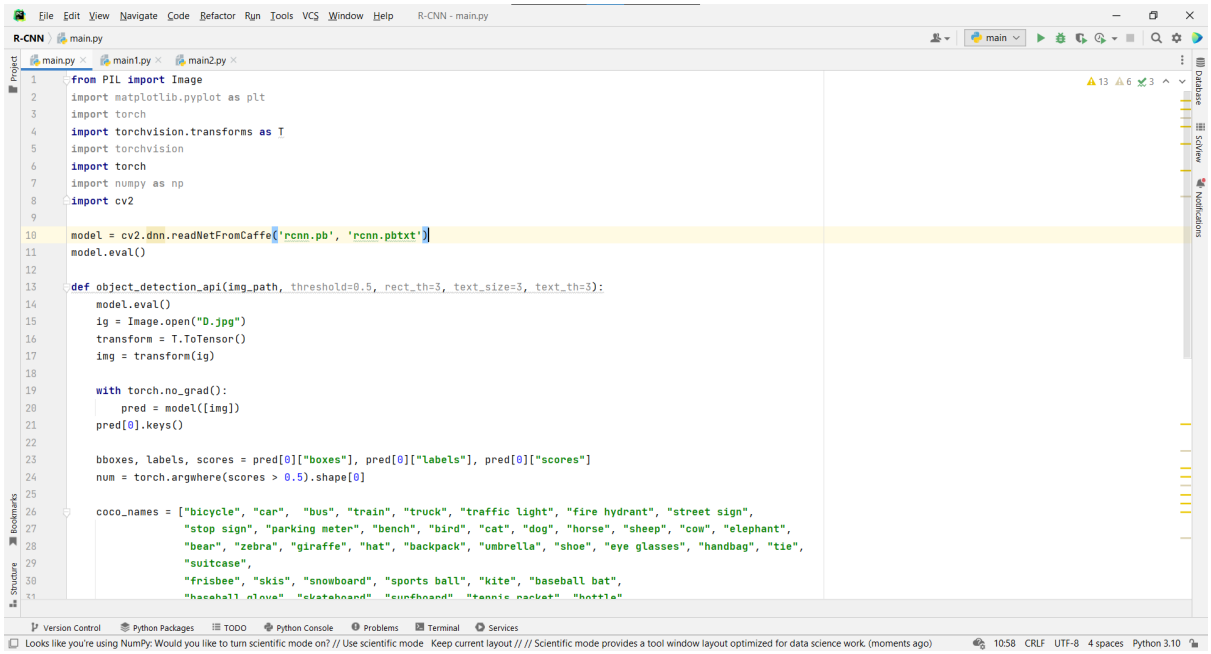



Figure 5.4: use SVM to make a classification decision about input

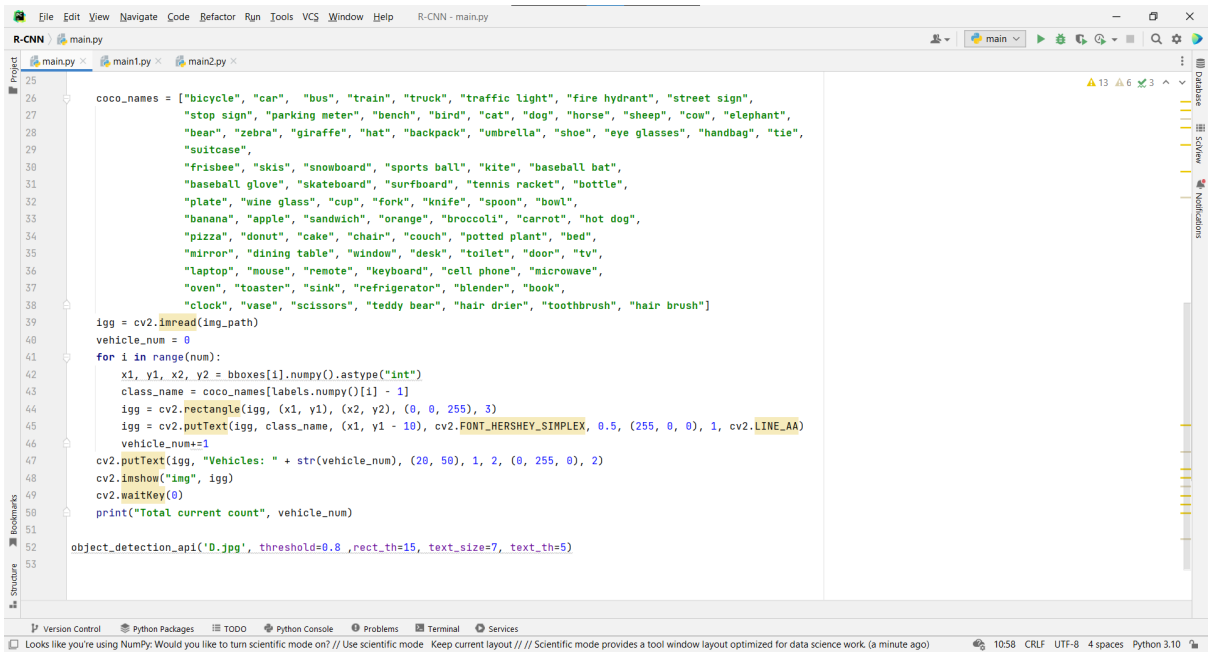
for train and testing, you can use pre-trained model and then go to the next step which is to Perform the object detection on the image and at the end we Count the vehicles that exist in this image so we ensure that vehicles in the image are detected and classified. and for the video, we do the same steps but in the detection process instead of we run on one image we loop on the frames of the video and at each frame, we count the vehicles, so the code will be like this. Let's see the code of Object detection using a pre-trained model (see Figure5.5 and Figure5.6)

[88, 89, 90] .



```python
from PIL import Image
import matplotlib.pyplot as plt
import torch
import torchvision.transforms as T
import torchvision
import torch
import numpy as np
import cv2

model = cv2.dnn.readNetFromCaffe('rcnn.pb', 'rcnn.pbtxt')
model.eval()

def object_detection_api(img_path, threshold=0.5, rect_th=3, text_size=3, text_th=3):
    model.eval()
    ig = Image.open("D.jpg")
    transform = T.ToTensor()
    img = transform(ig)

    with torch.no_grad():
        pred = model([img])
    pred[0].keys()

    bboxes, labels, scores = pred[0]["boxes"], pred[0]["labels"], pred[0]["scores"]
    num = torch.argwhere(scores > 0.5).shape[0]

    coco_names = ["bicycle", "car",  "bus", "train", "truck", "traffic light", "fire hydrant", "street sign",
                  "stop sign", "parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant",
                  "bear", "zebra", "giraffe", "hat", "backpack", "umbrella", "shoe", "eye glasses", "handbag", "tie",
                  "suitcase",
                  "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
```

Figure 5.5: import libraries, use pre-trained model and take the input image



```python
    coco_names = ["bicycle", "car",  "bus", "train", "truck", "traffic light", "fire hydrant", "street sign",
                  "stop sign", "parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant",
                  "bear", "zebra", "giraffe", "hat", "backpack", "umbrella", "shoe", "eye glasses", "handbag", "tie",
                  "suitcase",
                  "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
                  "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle",
                  "plate", "wine glass", "cup", "fork", "knife", "spoon", "bowl",
                  "banana", "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog",
                  "pizza", "donut", "cake", "chair", "couch", "potted plant", "bed",
                  "mirror", "dining table", "window", "desk", "toilet", "door", "tv",
                  "laptop", "mouse", "remote", "keyboard", "cell phone", "microwave",
                  "oven", "toaster", "sink", "refrigerator", "blender", "book",
                  "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush", "hair brush"]
    igg = cv2.imread(img_path)
    vehicle_num = 0
    for i in range(num):
        x1, y1, x2, y2 = bboxes[i].numpy().astype("int")
        class_name = coco_names[labels.numpy()[i] - 1]
        igg = cv2.rectangle(igg, (x1, y1), (x2, y2), (0, 0, 255), 3)
        igg = cv2.putText(igg, class_name, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1, cv2.LINE_AA)
        vehicle_num+=1
    cv2.putText(igg, "Vehicles: " + str(vehicle_num), (20, 50), 1, 2, (0, 255, 0), 2)
    cv2.imshow("img", igg)
    cv2.waitKey(0)
    print("Total current count", vehicle_num)

object_detection_api('D.jpg', threshold=0.8 ,rect_th=15, text_size=7, text_th=5)
```

Figure 5.6: detect the vehicles in the given image and print show the result on image

## 5.2 Fast R-CNN

Rather than processing each region of interest separately instead we‘re going to run the entire image through some CNN at once, and now we still are using some RP but rather than cropping out the pixels of the image corresponding to RPs instead, we imagine

projecting those RPs onto this C feature map and then taking crops from the C feature map rather than taking crops from the image and (this allows us to reuse a lot of this expensive C computation across the entire image when we have many many crops per image) and this method is called Fast R CNN. The Fast R-CNN (Region-based Convolutional Neural Network) model is a popular object detection algorithm that builds upon the earlier R-CNN and Fast R-CNN models. It improves both the speed and accuracy of object detection by introducing several key innovations. Here are the steps involved in the Fast R-CNN model:

Input Image and Region Proposals: The model takes an input image as well as a set of region proposals generated by a region proposal network (RPN) or another region proposal method. Region proposals are bounding boxes that potentially contain objects of interest (see Figure5.7) .

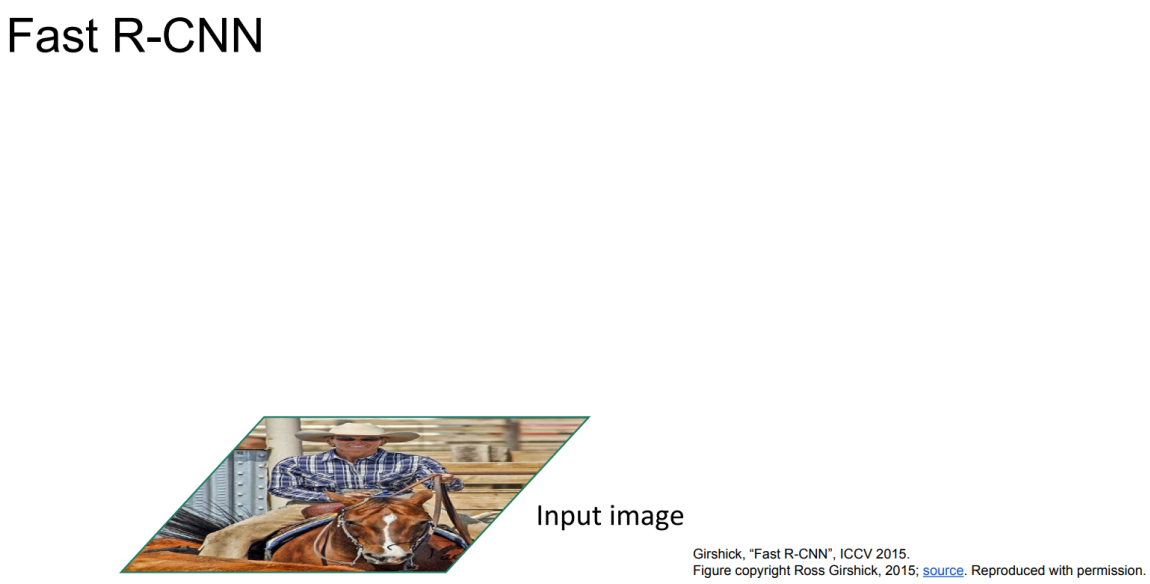



Figure 5.7: Taking input image and pass it to Fast R-CNN model

Region of Interest (RoI) Pooling: The input image and region proposals are fed into an RoI pooling layer. RoI pooling divides each region proposal into fixed-sized sections and scales them to a pre-defined size, typically a small spatial grid. This allows the RoI pooling layer to extract fixed-sized feature maps from each region proposal, regardless of the original size or aspect ratio.

Convolutional Feature Extraction: The RoI pooling layer outputs the fixed-sized feature maps for each region proposal. These feature maps are then passed through a shared convolutional network, such as a pre-trained CNN like VGG or ResNet. The convolutional layers extract rich feature representations from the region proposals . (see Figure5.9) Region Classification and Bounding Box Regression: The output of the convolutional layers is used for region classification and bounding box regression. A set of fully connected layers takes the extracted features as input and performs two parallel tasks: Region

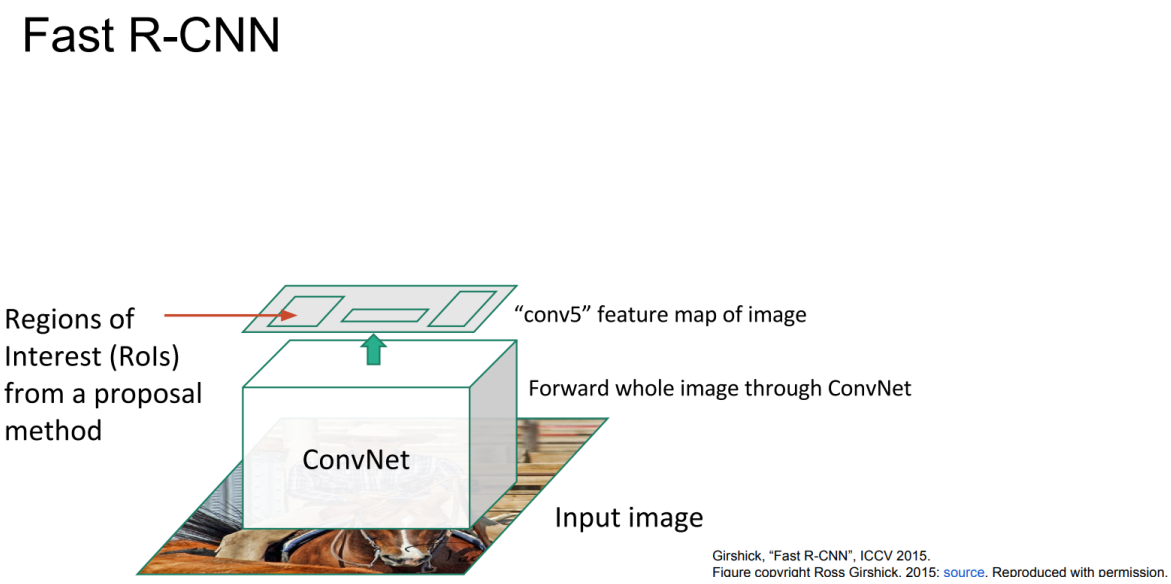



Figure 5.8: Divide the image into ROIs and pass them to ConvNet layers

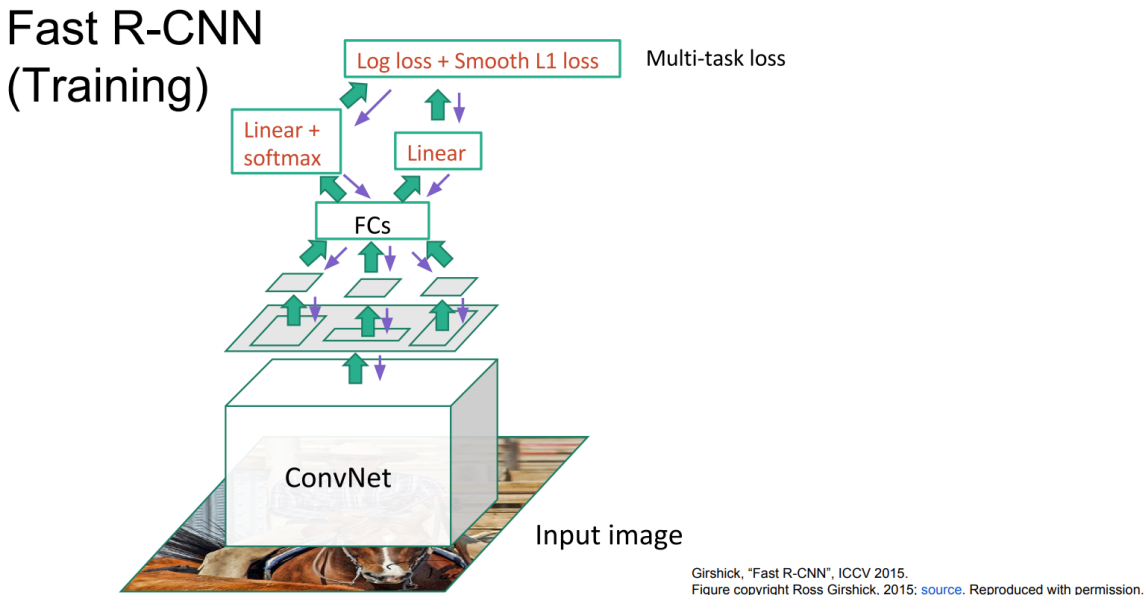
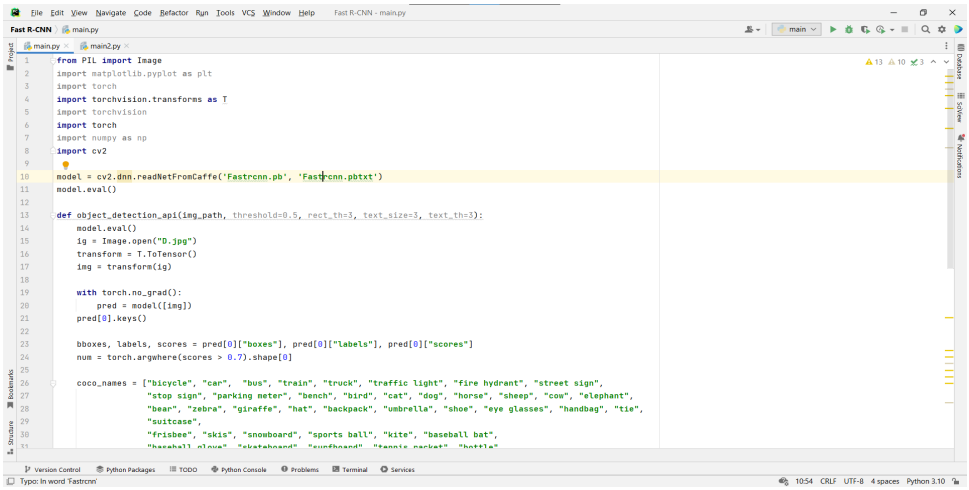



Figure 5.9: Get final output after training Fast R-CNN model

Classification: It predicts the class probabilities for each region proposal, indicating the presence or absence of different object categories. Bounding Box Regression: It refines the coordinates of the region proposal's bounding box to improve localization accuracy.

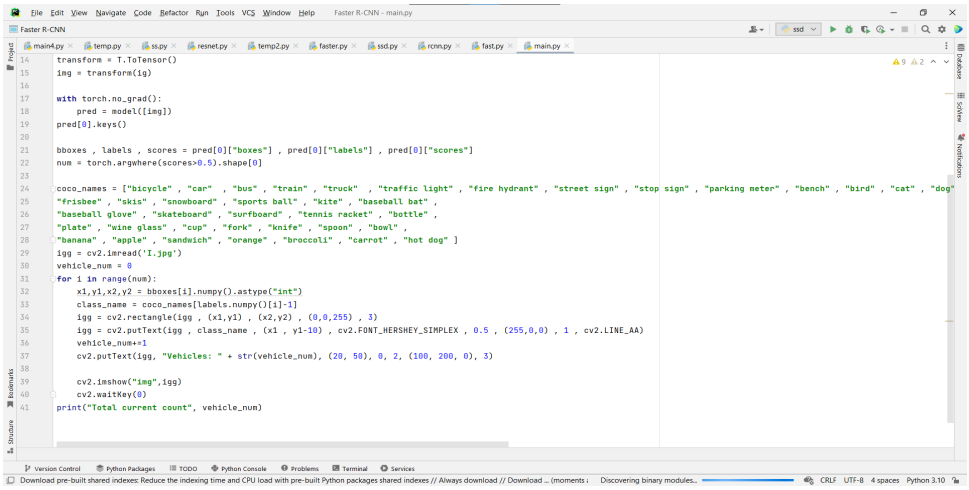
Non-Maximum Suppression (NMS): After obtaining the classification scores and refined bounding box coordinates, non-maximum suppression is applied to remove duplicate or highly overlapping detections. NMS selects the most confident detection for each object instance and suppresses other detections that have a significant overlap with it.

Output: The final output of the Fast R-CNN model includes the object class labels, bounding box coordinates, and confidence scores for each detected object instance (see Figure5.9). Let's see the code of Object detection using a pre-trained model (see Figure5.10 and Figure5.11)



```
from PIL import Image
import matplotlib.pyplot as plt
import torch
import torchvision.transforms as T
import torchvision
import torch
import numpy as np
import cv2

model = cv2.dnn.readNetFromCaffe('Fastrcnn.pb', 'Fastrcnn.pbtxt')
model.eval()

def object_detection_api(img_path, threshold=0.5, rect_th=3, text_size=3, text_th=3):
    model.eval()
    ig = Image.open("D.jpg")
    transform = T.ToTensor()
    img = transform(ig)

    with torch.no_grad():
        pred = model([img])
    pred[0].keys()

    bboxes, labels, scores = pred[0]["boxes"], pred[0]["labels"], pred[0]["scores"]
    num = torch.argwhere(scores > 0.7).shape[0]

    coco_names = ["bicycle", "car",  "bus", "train", "truck", "traffic light", "fire hydrant", "street sign",
                  "stop sign", "parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant",
                  "bear", "zebra", "giraffe", "hat", "backpack", "umbrella", "shoe", "eye glasses", "handbag", "tie",
                  "suitcase",
                  "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
```

Figure 5.10: import libraries, use pre-trained model and take the input image



```
transform = T.ToTensor()
img = transform(ig)

with torch.no_grad():
    pred = model([img])
pred[0].keys()

bboxes , labels , scores = pred[0]["boxes"] , pred[0]["labels"] , pred[0]["scores"]
num = torch.argwhere(scores>0.5).shape[0]

coco_names = ["bicycle" , "car"  , "bus" , "train" , "truck"  , "traffic light" , "fire hydrant" , "street sign" , "stop sign" , "parking meter" , "bench" , "bird" , "cat" , "dog"
"frisbee" , "skis" , "snowboard" , "sports ball" , "kite" , "baseball bat" ,
"baseball glove" , "skateboard" , "surfboard" , "tennis racket" , "bottle" ,
"plate" , "wine glass" , "cup" , "fork" , "knife" , "spoon" , "bowl" ,
"banana" , "apple" , "sandwich" , "orange" , "broccoli" , "carrot" , "hot dog" ]
igg = cv2.imread('I.jpg')
vehicle_num = 0
for i in range(num):
    x1,y1,x2,y2 = bboxes[i].numpy().astype("int")
    class_name = coco_names[labels.numpy()[i]-1]
    igg = cv2.rectangle(igg , (x1,y1) , (x2,y2) , (0,0,255) , 3)
    igg = cv2.putText(igg , class_name , (x1 , y1-10) , cv2.FONT_HERSHEY_SIMPLEX , 0.5 , (255,0,0) , 1 , cv2.LINE_AA)
    vehicle_num+=1
    cv2.putText(igg, "Vehicles: " + str(vehicle_num), (20, 50), 0, 2, (100, 200, 0), 3)

    cv2.imshow("img",igg)
    cv2.waitKey(0)
print("Total current count", vehicle_num)
```

Figure 5.11: detect the vehicles in the given image and print show the result on image

[91, 92, 93].

## 5.3 Faster R-CNN

So instead we'll just make the network itself predict its own region proposals. And so the way that this sort of works is that again, we take our input image, run the entire input

image altogether through some convolutional layers to get some convolutional feature map representing the entire high resolution image but the projection on the feature map was from selective search in the case of R-CNN but here it's not from selective Search it's from RPN(Region proposal network)so the feature map that is generated from the input image by Conv layers this feature map is passed to Convnet to predict object classification whether it's presented or not and bounding box, and this is called the Faster R-CNN model. The Faster R-CNN (Region-based Convolutional Neural Network) model is an extension of the Fast R-CNN model that improves the efficiency and accuracy of object detection. It introduces a Region Proposal Network (RPN) to generate region proposals, eliminating the need for external region proposal methods. Let's explore the details of how Faster R-CNN works:

Input Image: The Faster R-CNN model takes an input image as its initial input.

Convolutional Feature Extraction: The input image is passed through a shared convolutional network, such as VGG, ResNet, or another pre-trained CNN. The convolutional layers of the network extract feature maps from the image.

Region Proposal Network (RPN): The feature maps obtained from the convolutional layers are fed into the RPN. The RPN slides a small-sized window (called an anchor) across the feature maps and predicts whether each anchor contains an object or not. The RPN also predicts adjustments to the anchor's coordinates to improve the accuracy of the bounding box proposals. The RPN generates region proposals based on the predicted scores and coordinates of the anchors.

RoI (Region of Interest) Pooling: The feature maps from the convolutional layers and the region proposals generated by the Region Proposal Network (RPN) are then fed into the RoI pooling layer. The RoI pooling layer extracts fixed-sized feature maps for each region proposal, regardless of their original size or aspect ratio. This allows for efficient feature extraction and consistent input sizes for subsequent layers.

Region Classification and Bounding Box Regression: The fixed-sized feature maps from the RoI pooling layer are passed through a set of fully connected layers. These layers perform two parallel tasks: Region Classification: They predict the probabilities of different object categories for each region proposal, indicating the presence or absence of objects. Bounding Box Regression: They refine the coordinates of the region proposal's bounding box to improve localization accuracy.

Non-Maximum Suppression (NMS): After obtaining the classification scores and refined bounding box coordinates, non-maximum suppression is applied to remove duplicate or highly overlapping detections. NMS selects the most confident detection for each object instance and suppresses other detections that have a significant overlap with it.

Output: The final output of the Faster R-CNN model includes the object class labels, bounding box coordinates, and confidence scores for each detected object instance (see Figure5.12) . Let‘s see the code of Object detection using a pre-trained model (see Figure5.14 and Figure5.13) [94, 95, 96, 97].

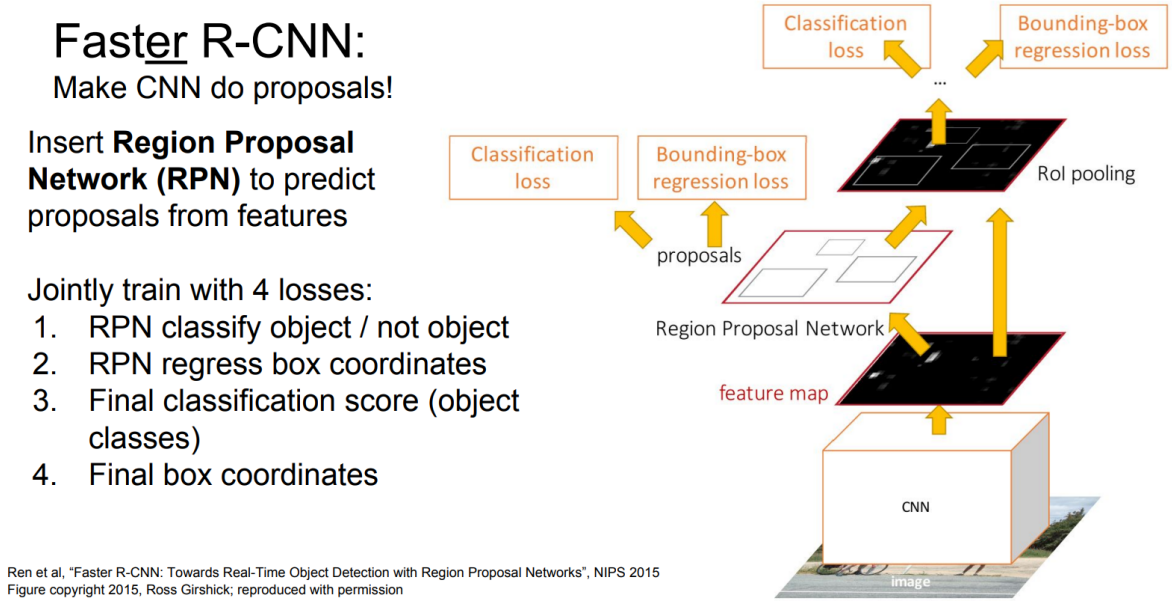



Figure 5.12: Get final output after training Faster R-CNN model

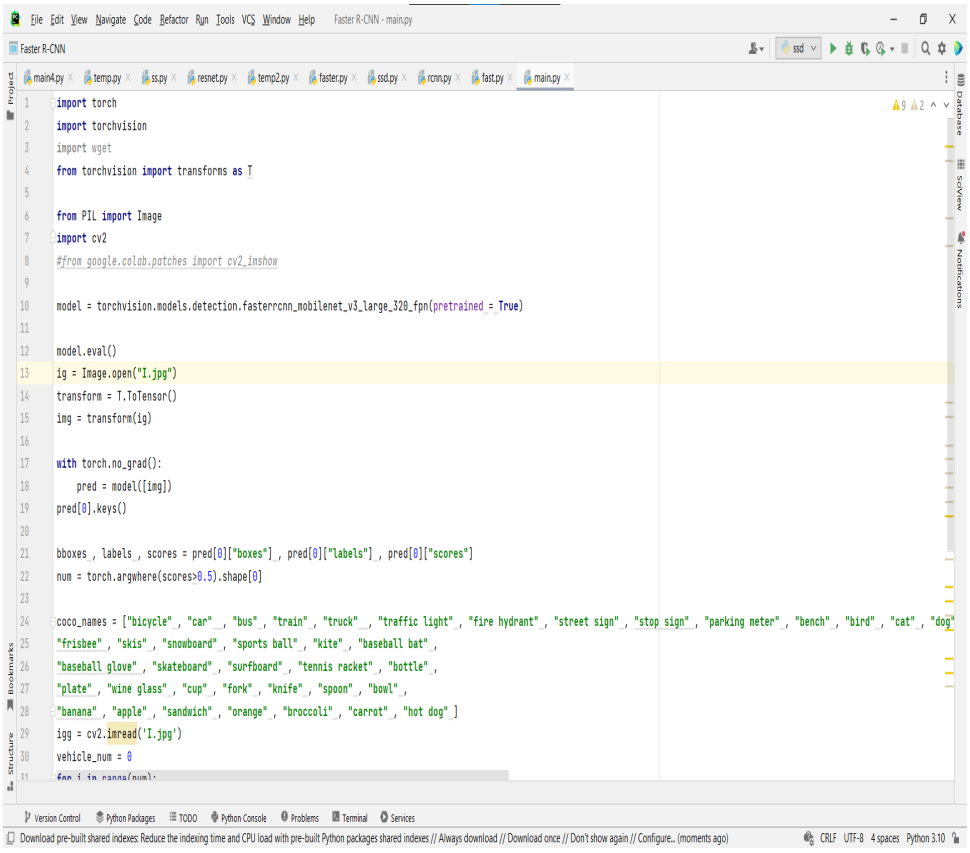



Figure 5.13: detect the vehicles in the given image and print show the result on image

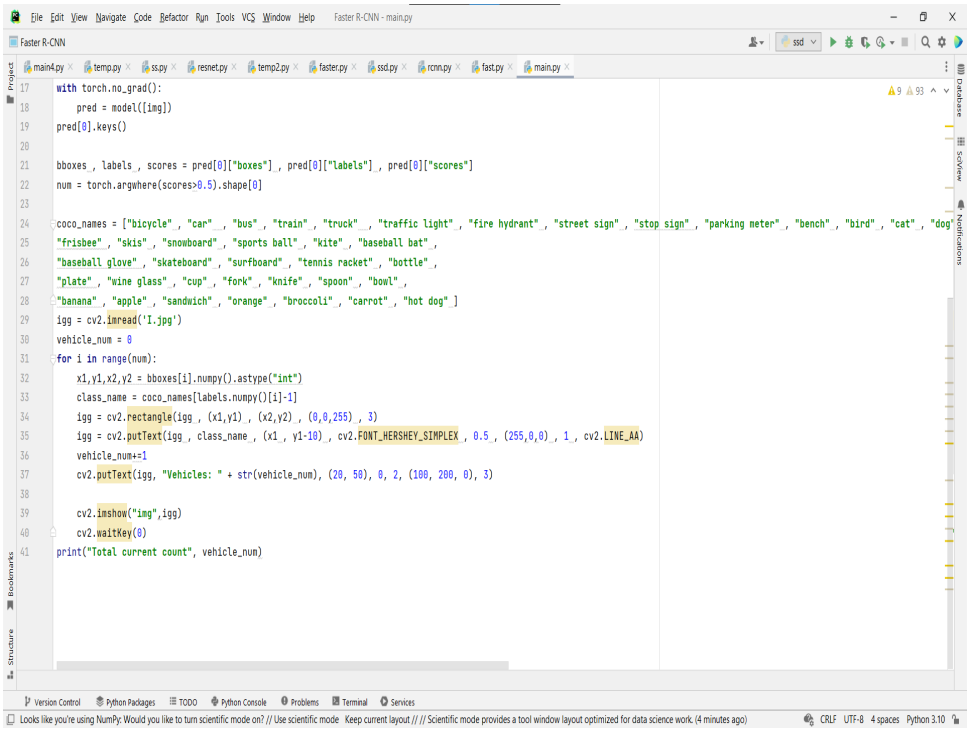



Figure 5.14: import libraries, use pre-trained model and take the input image

## 5.4 Res-Net

The ResNet (Residual Neural Network) is a deep convolutional neural network architecture that addresses the degradation problem encountered in training very deep networks. It introduces the concept of residual connections, allowing the network to learn residual functions instead of directly learning the desired underlying mappings. This enables the training of extremely deep networks with improved accuracy (see Figure5.15) . Let's dive into the details of how Res-Net works:

Input Image: The ResNet model takes an input image as its initial input.

Convolutional Layers: The input image is passed through a series of convolutional layers with different filter sizes to extract features. These layers perform convolutions and apply activation functions to capture hierarchical representations of the input image.

Residual Blocks: ResNet introduces residual blocks, which are the key building blocks of the architecture. A residual block consists of two main components: the identity shortcut and the residual mapping. The identity shortcut skips one or more convolutional layers and directly connects the input to the output of the residual block. The residual mapping represents the desired underlying mapping to be learned by the network. By learning the residual mapping instead of the direct mapping, the network can focus on learning the residual details, which makes training easier.

Skip Connections: The skip connections, formed by the identity shortcuts, allow the gradient flow during backpropagation to bypass the residual block. This addresses the

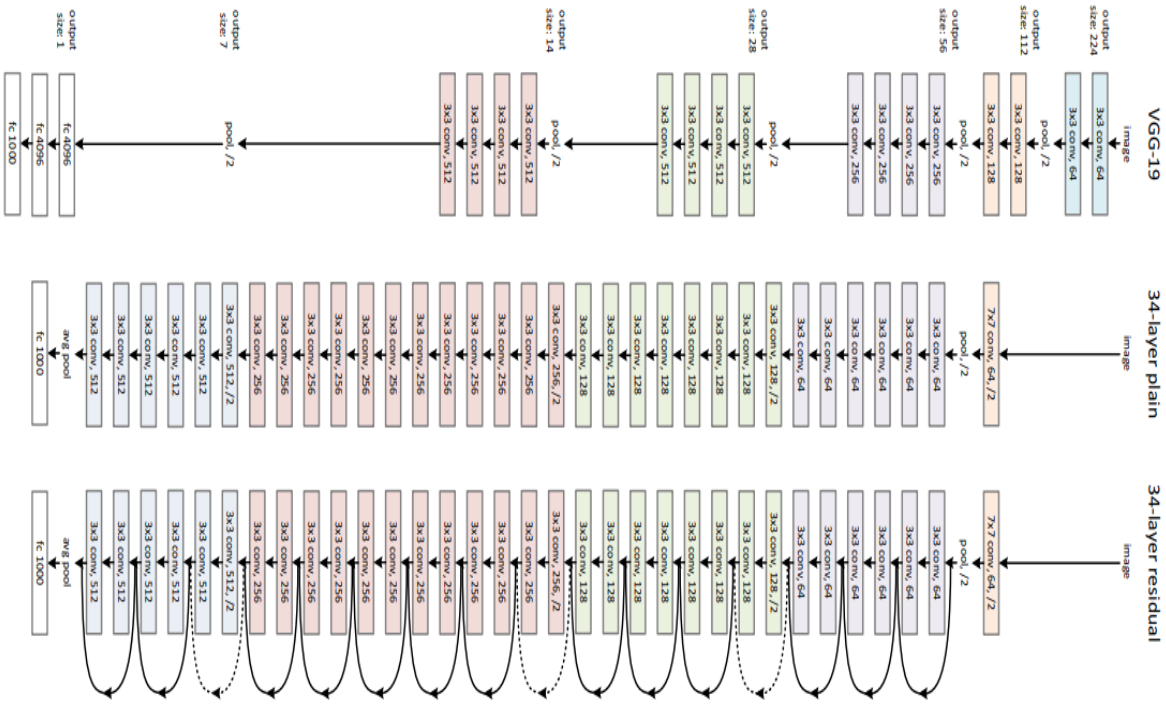



Figure 5.15: The shape of the Res-Net model

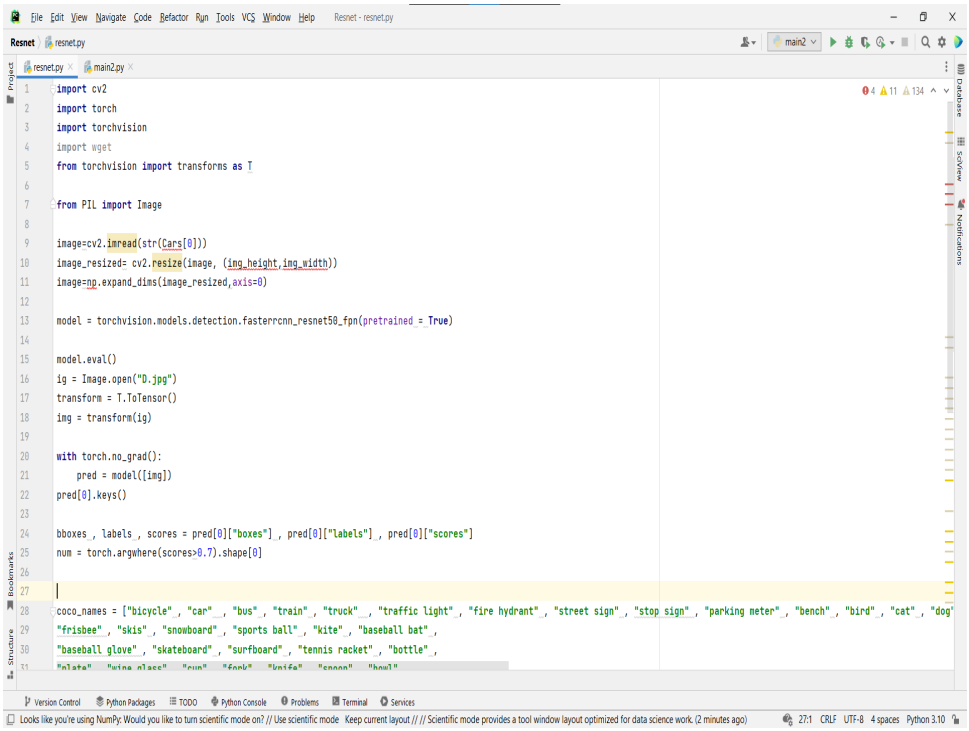
degradation problem, where the accuracy of deep networks starts to degrade as the depth increases. The skip connections enable the network to learn and propagate gradients effectively, facilitating the training of very deep networks.

Bottleneck Architecture: ResNet employs a bottleneck architecture in certain layers to reduce computational complexity. A bottleneck layer reduces the number of channels by applying 1x1 convolutions, which reduces the computational cost while preserving important features.

Global Average Pooling: After the convolutional layers and residual blocks, the output feature maps are passed through a global average pooling layer. Global average pooling reduces the spatial dimensions of the feature maps to a fixed size, resulting in a compact feature representation.

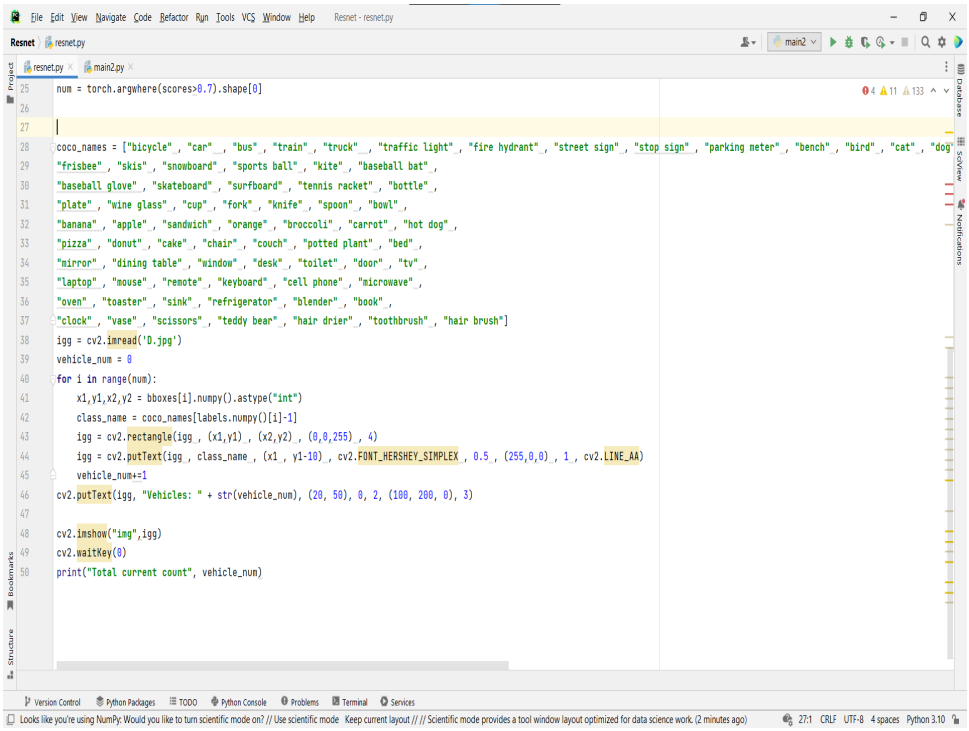
Fully Connected Layers: The compact feature representation obtained from global average pooling is flattened and fed into fully connected layers. These layers perform classification or regression tasks, depending on the specific application of the ResNet model.

Output: The final output of the ResNet model includes the predicted class probabilities or regression values for the given input image. Let‘s see the code of Object detection using a pre-trained model (see Figure5.17 and Figure5.16) [94, 98, 99, 100].



```
import cv2
import torch
import torchvision
import wget
from torchvision import transforms as T

from PIL import Image

image=cv2.imread(str(Cars[0]))
image_resized= cv2.resize(image, (img_height,img_width))
image=np.expand_dims(image_resized,axis=0)

model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained = True)

model.eval()
ig = Image.open("D.jpg")
transform = T.ToTensor()
img = transform(ig)

with torch.no_grad():
    pred = model([img])
pred[0].keys()

bboxes , labels , scores = pred[0]["boxes"] , pred[0]["labels"] , pred[0]["scores"]
num = torch.argwhere(scores>0.7).shape[0]

coco_names = ["bicycle" , "car" , "bus" , "train" , "truck" , "traffic light" , "fire hydrant" , "street sign" , "stop sign" , "parking meter" , "bench" , "bird" , "cat" , "dog"
"frisbee" , "skis" , "snowboard" , "sports ball" , "kite" , "baseball bat" ,
"baseball glove" , "skateboard" , "surfboard" , "tennis racket" , "bottle" ,
"plate" , "wine glass" , "cup" , "fork" , "knife" , "spoon" , "bowl"
```

Figure 5.16: detect the vehicles in the given image and print show the result on image



```
num = torch.argwhere(scores>0.7).shape[0]

coco_names = ["bicycle" , "car" , "bus" , "train" , "truck" , "traffic light" , "fire hydrant" , "street sign" , "stop sign" , "parking meter" , "bench" , "bird" , "cat" , "dog"
"frisbee" , "skis" , "snowboard" , "sports ball" , "kite" , "baseball bat" ,
"baseball glove" , "skateboard" , "surfboard" , "tennis racket" , "bottle" ,
"plate" , "wine glass" , "cup" , "fork" , "knife" , "spoon" , "bowl" ,
"banana" , "apple" , "sandwich" , "orange" , "broccoli" , "carrot" , "hot dog" ,
"pizza" , "donut" , "cake" , "chair" , "couch" , "potted plant" , "bed" ,
"mirror" , "dining table" , "window" , "desk" , "toilet" , "door" , "tv" ,
"laptop" , "mouse" , "remote" , "keyboard" , "cell phone" , "microwave" ,
"oven" , "toaster" , "sink" , "refrigerator" , "blender" , "book" ,
"clock" , "vase" , "scissors" , "teddy bear" , "hair drier" , "toothbrush" , "hair brush"]
igg = cv2.imread('D.jpg')
vehicle_num = 0
for i in range(num):
    x1,y1,x2,y2 = bboxes[i].numpy().astype("int")
    class_name = coco_names[labels.numpy()[i]-1]
    igg = cv2.rectangle(igg , (x1,y1) , (x2,y2) , (0,0,255) , 4)
    igg = cv2.putText(igg , class_name , (x1 , y1-10) , cv2.FONT_HERSHEY_SIMPLEX , 0.5 , (255,0,0) , 1 , cv2.LINE_AA)
    vehicle_num+=1
cv2.putText(igg, "Vehicles: " + str(vehicle_num), (20, 50), 0, 2, (100, 200, 0), 3)

cv2.imshow("img",igg)
cv2.waitKey(0)
print("Total current count", vehicle_num)
```

Figure 5.17: import libraries, use pre-trained model and take the input image

## 5.5 Single Shot Detector (SSD)

The Single Shot Detector (SSD) is a popular object detection model known for its efficiency and accuracy. It is a one-stage detector that directly predicts object bounding boxes and class probabilities for the region proposals in a single pass through the network. SSD combines feature maps at multiple scales to detect objects of various sizes, enabling it to achieve real-time performance (see Figure5.18) . Let's explore the steps of how SSD

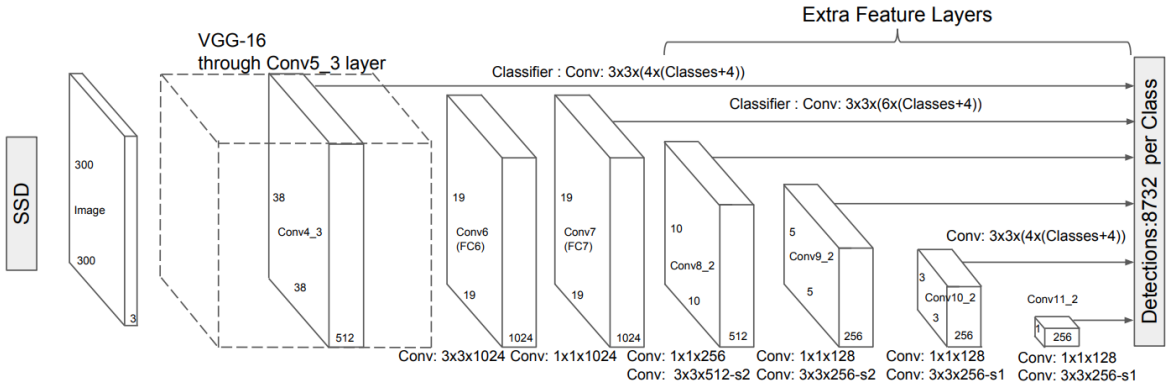



Figure 5.18: The architecture of the SSD model

works in detail:

Input Image: The SSD model takes an input image as its initial input.

Base Convolutional Network: The input image is passed through a base convolutional network, such as VGG, ResNet, or another pre-trained CNN. The base network extracts feature maps at multiple scales, capturing hierarchical representations of the input image.

Feature Map Pyramid: SSD creates a feature map pyramid by applying a set of convolutional layers (usually 3x3 convolutional filters) to the output feature maps of the base network. These layers progressively reduce the spatial dimensions of the feature maps while increasing the number of channels. The resulting feature maps have different spatial resolutions, allowing the model to detect objects at various scales.

Multi-scale Feature Maps: Each level of the feature map pyramid corresponds to a specific scale of objects. The feature maps at different levels capture both fine-grained details and high-level semantic information.

Anchor Boxes: SSD defines a set of anchor boxes (or default boxes) of different scales and aspect ratios on each feature map level. These anchor boxes act as reference bounding boxes that are densely distributed across the feature map. For each spatial location on a feature map, multiple anchor boxes are associated with it.

Predictions: SSD applies a set of convolutional layers to each level of the feature map pyramid. These layers predict the class probabilities and offsets (or regressions) for each anchor box. The class probabilities indicate the presence of different object classes, and

the offsets refine the positions and sizes of the anchor boxes to match the ground truth bounding boxes.

Non-Maximum Suppression (NMS): After obtaining the predictions from all levels of the feature map pyramid, non-maximum suppression is applied to remove duplicate or highly overlapping detections. NMS selects the most confident detection for each object instance and suppresses other detections that have a significant overlap with it.

Output: The final output of the SSD model includes the predicted class labels, bounding box coordinates, and confidence scores for each detected object instance.

SSD's key strength lies in its ability to detect objects at multiple scales using feature maps of different resolutions. By predicting object boundaries and class probabilities directly, SSD achieves real-time performance without the need for complex post-processing steps. Let's see the code of Object detection using a pre-trained model (see Figure5.20 and Figure5.19) [94, 101, 102].

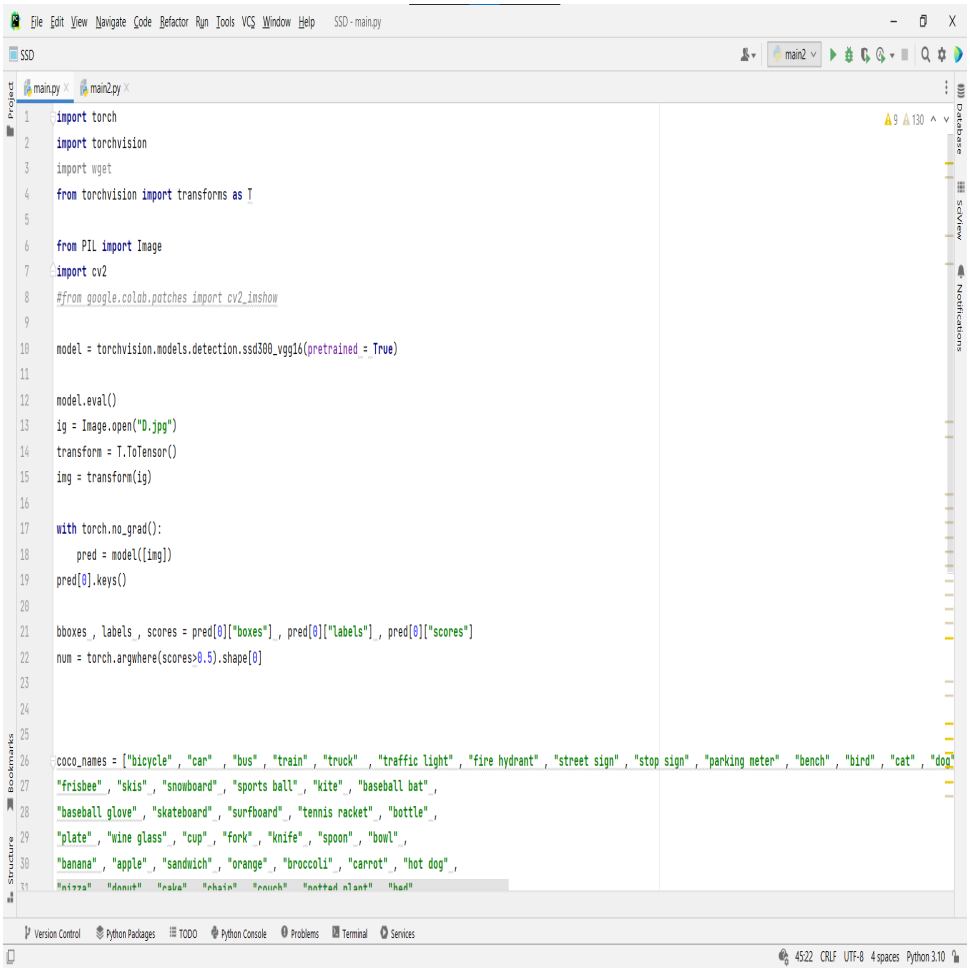
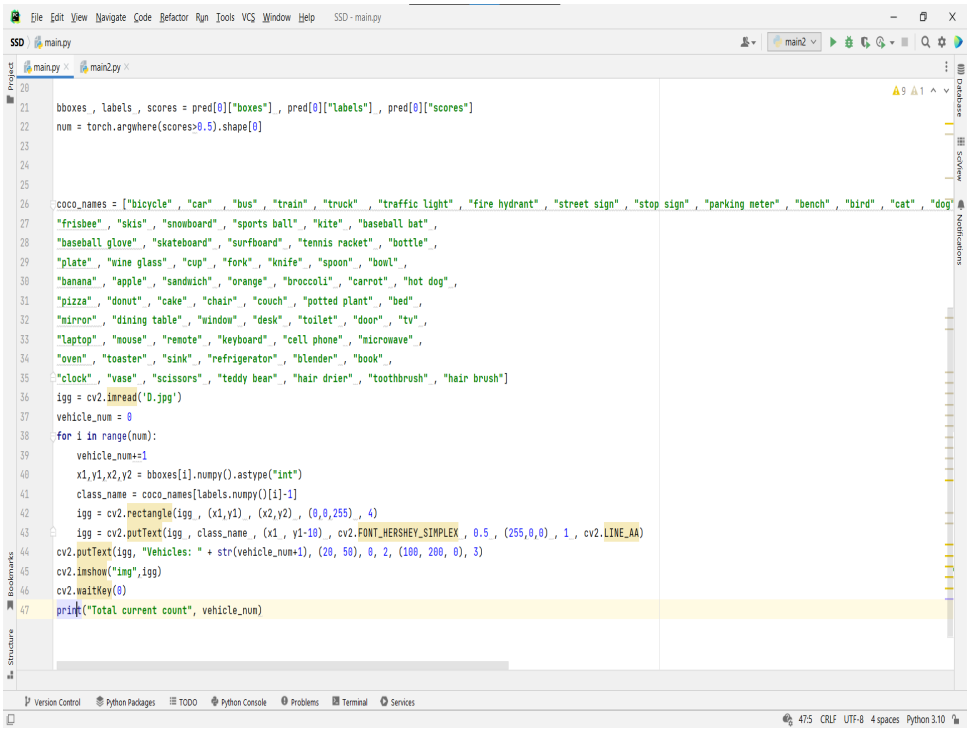



Figure 5.19: detect the vehicles in the given image and print show the result on image



```
bboxes , labels , scores = pred[0]["boxes"] , pred[0]["labels"] , pred[0]["scores"]
num = torch.argwhere(scores>0.5).shape[0]



coco_names = ["bicycle" , "car" , "bus" , "train" , "truck" , "traffic light" , "fire hydrant" , "street sign" , "stop sign" , "parking meter" , "bench" , "bird" , "cat" , "dog"
"frisbee" , "skis" , "snowboard" , "sports ball" , "kite" , "baseball bat" ,
"baseball glove" , "skateboard" , "surfboard" , "tennis racket" , "bottle" ,
"plate" , "wine glass" , "cup" , "fork" , "knife" , "spoon" , "bowl" ,
"banana" , "apple" , "sandwich" , "orange" , "broccoli" , "carrot" , "hot dog" ,
"pizza" , "donut" , "cake" , "chair" , "couch" , "potted plant" , "bed" ,
"mirror" , "dining table" , "window" , "desk" , "toilet" , "door" , "tv" ,
"laptop" , "mouse" , "remote" , "keyboard" , "cell phone" , "microwave" ,
"oven" , "toaster" , "sink" , "refrigerator" , "blender" , "book" ,
"clock" , "vase" , "scissors" , "teddy bear" , "hair drier" , "toothbrush" , "hair brush"]
igg = cv2.imread('D.jpg')
vehicle_num = 0
for i in range(num):
    vehicle_num+=1
    x1,y1,x2,y2 = bboxes[i].numpy().astype("int")
    class_name = coco_names[labels.numpy()[i]-1]
    igg = cv2.rectangle(igg , (x1,y1) , (x2,y2) , (0,0,255) , 4)
    igg = cv2.putText(igg , class_name , (x1 , y1-10) , cv2.FONT_HERSHEY_SIMPLEX , 0.5 , (255,0,0) , 1 , cv2.LINE_AA)
cv2.putText(igg, "Vehicles: " + str(vehicle_num+1), (20, 50), 0, 2, (100, 200, 0), 3)
cv2.imshow("img",igg)
cv2.waitKey(0)
print("Total current count", vehicle_num)
```

Figure 5.20: import libraries, use pre-trained model and take the input image

## 5.6 You Only Look Once version 8 YOLOv8

YOLOv8 is the most recent addition to the influential family of models based on the YOLO (You Only Look Once) architecture. Developed by Ultralytics, the team behind YOLOv3 and YOLOv5, YOLOv8 offers not only object detection but also instance segmentation and image classification. Built on PyTorch, the model can run on both CPU and GPU.

In this guide, we will walk you through the step-by-step process of training the YOLOv8 object detection model using a custom dataset. You will learn how to utilize the new API, prepare the dataset, and importantly, train and validate the model.

The training process for a YOLOv8 object detection model on custom data involves the following steps:

Prepare a custom dataset for YOLOv8: Creating a custom dataset can be a laborious task, requiring hours of collecting, labelling, and formatting images. Thankfully, Roboflow simplifies this process through their dashboard. You can create a new project, upload your images (the dashboard automatically reads images and annotations), label the images using Roboflow Annotate, and generate a new version of the dataset. Optionally, you can add preprocessing and augmentations to enhance the model's robustness. Once done, export the dataset for easy loading into the training notebook.

Train YOLOv8 on the custom dataset: After pasting the dataset download snippet into your YOLOv8 Colab notebook, you can start the training process. The model will train for a duration depending on the dataset size and selected training options.

Validate with a new model: It is good practice to evaluate the trained model on unseen images. By dividing the dataset into three parts, you can reserve one part as a test dataset for validation and prediction with the custom model.

Export and upload weights: Once the training is complete, you will have a set of trained weights located in the "/runs/detect/train/weights/best.pt" folder of your project. These weights can be uploaded to Roboflow Deploy using the "deploy()" function in the Roboflow pip package, allowing you to utilize the trained weights in a hosted API endpoint.

Once the model weights are uploaded, you can incorporate your custom trained YOLOv8 model into production applications or share it externally. The provided link contains the code for object detection, which can be executed on Google Colab.

By following these steps, you can effectively train and deploy a YOLOv8 model for object detection using your custom dataset. The code of detection is provided in this link "https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/how-to-track-and-count-vehicles-with-yolov8.ipynb". This code runs on Google collab

# Chapter 6

# Results

The results in the below figures show the results of the 6 models achieved around the same images on the tested dataset. Furthermore, when testing with the same test data, it is evident that the YOLO v8 model's accuracy is statistically similar to that of the SSD model. These results are shown below in the figures. Consequently, we may infer that YOLO v8 outperforms the other 5 models.

## 6.1 Results from each model

For the first model which is the R-CNN model the average detection of vehicles in image and video and classifying them was (52.43which is quite bad. This result is derived from the different images where in some of them the classification was wrong, the model sometimes classifies the car as trunk and sometimes the model does not detect vehicles so the number of vehicles is not accurate (see Figure6.1 and Figure6.2) .

While the second model which is the Fast R-CNN model the average detection of vehicles in image and video and classifying them was (66.3which is good compared to the R-CNN model. This result is based on the same images from the dataset which we used in the first model (see Figure6.3) .

And the Third model which is the Faster R-CNN model the average detection of vehicles in image and video and classifying them was (69.81which is quite good compared to the Fast R-CNN model and better than R-CNN. This result is derived from the same images from the dataset which we used in the first model (see Figure6.4) .

For the fourth model which is Res-Net the average detection of vehicles in image and video and classifying them was (81.15which is better compared to R-CNN and good compared to the Fast R-CNN model and Faster R-CNN. This result is built upon the foundation of the same images from the dataset which we used in the first model (see Figure6.5) .

The Fifth one which is SSD gives better results compared to the other models. The average detection was 85.61(see Figure6.6)

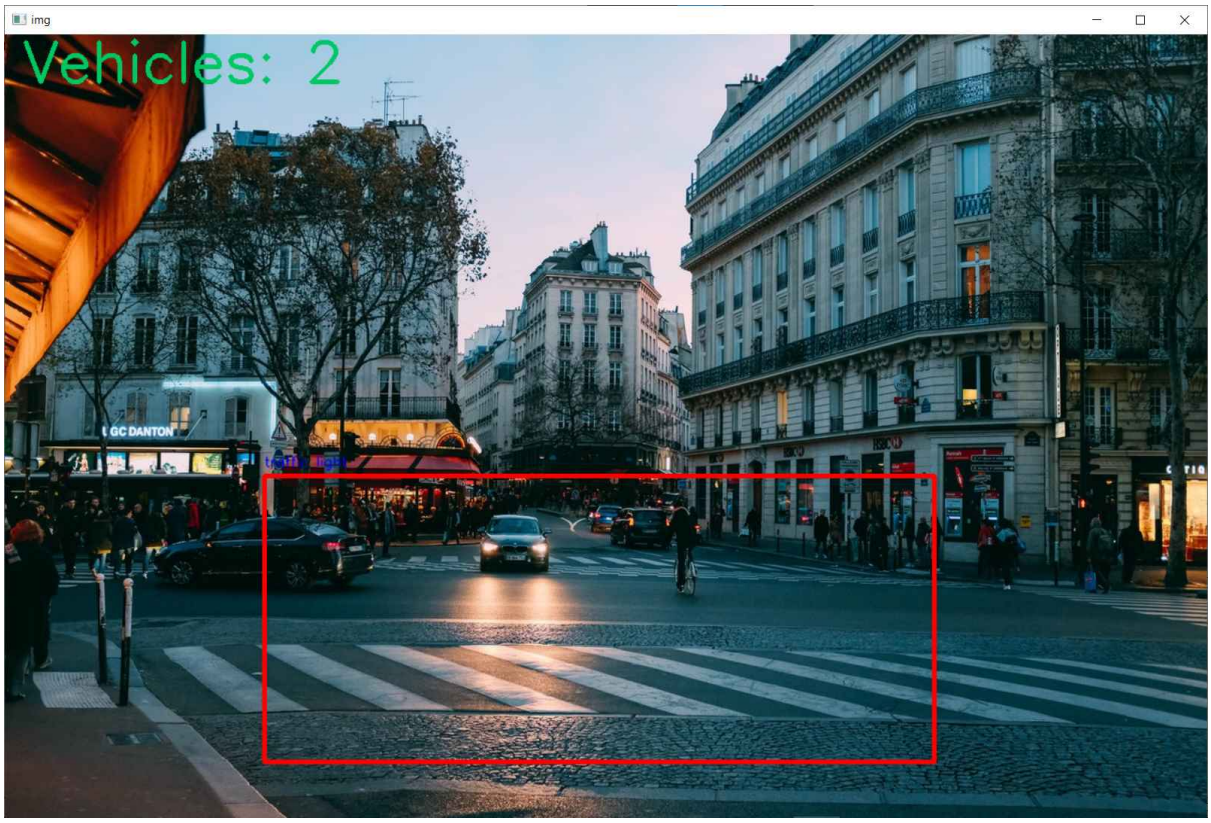



Figure 6.1: Results from an image using R-CNN model

For the last one which is YOLO v8 the results were the best by 91for detecting and classifying the vehicles (see Figure6.7) .

Now we can see the results from some random video inputs and different results from each model. The first one is YOLO v8 (see Figure6.8 , Figure6.9 , and Figure6.10).

Second model which is the SSD model (see Figure6.11 and Figure6.12) .

The third one is Faster R-CNN the results were (see Figure6.13 and Figure6.14)

For the fourth one which is Res-Net the results as we see are (see Figure6.15 and Figure6.16) .

The fifth one is Fast R-CNN we can see the results (see Figure6.17 and Figure6.18)

And finally, in the last model which is R-CNN the results are (see Figure6.19 and Figure6.20)

## 6.2 Comparing results

We used a dataset from Kaggle and the GPU I ran these lines of code was GTX 1050Ti, so from the above results we could conclude that the YOLO v8 has the highest results compared to the other five models by 91This percentage can differ based on the dataset we used and the model we used in training which these factors can affect the final results

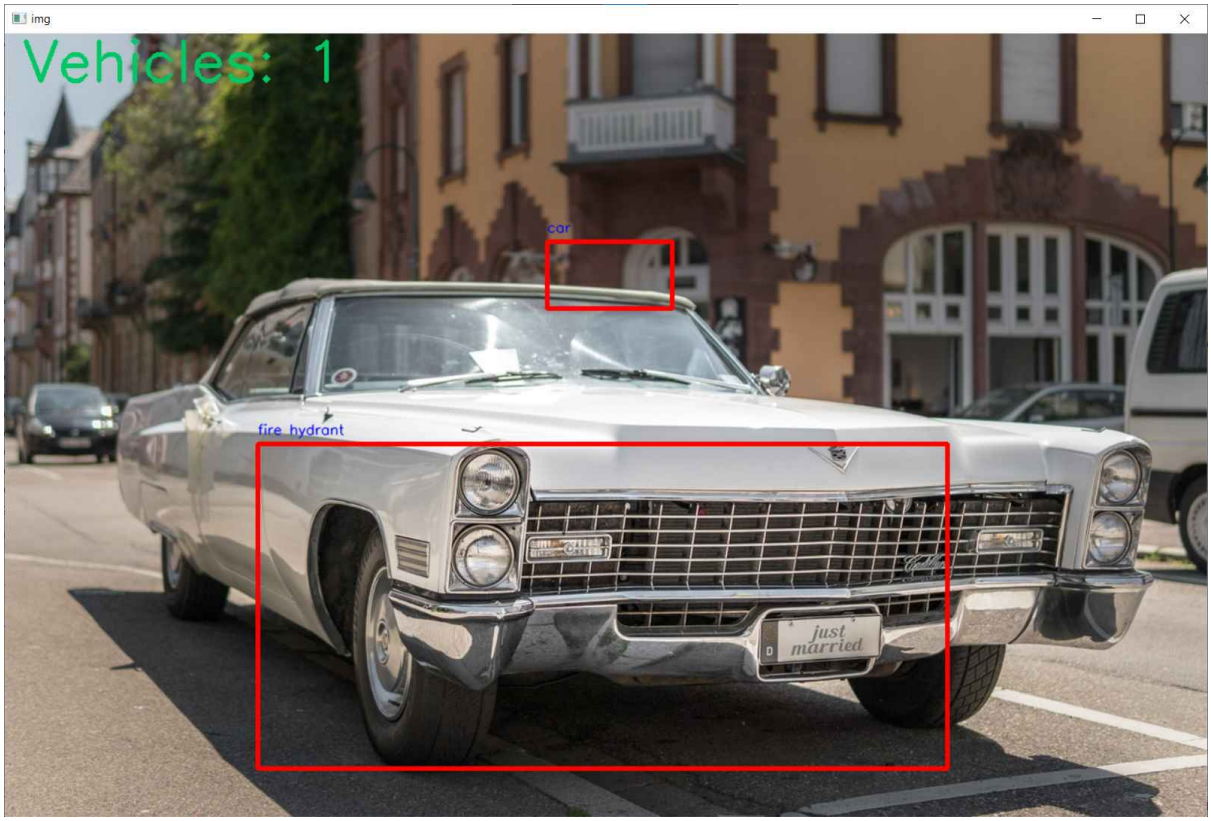



Figure 6.2: Results from another image using R-CNN model


Figure 6.3: Results from an image using Fast R-CNN model

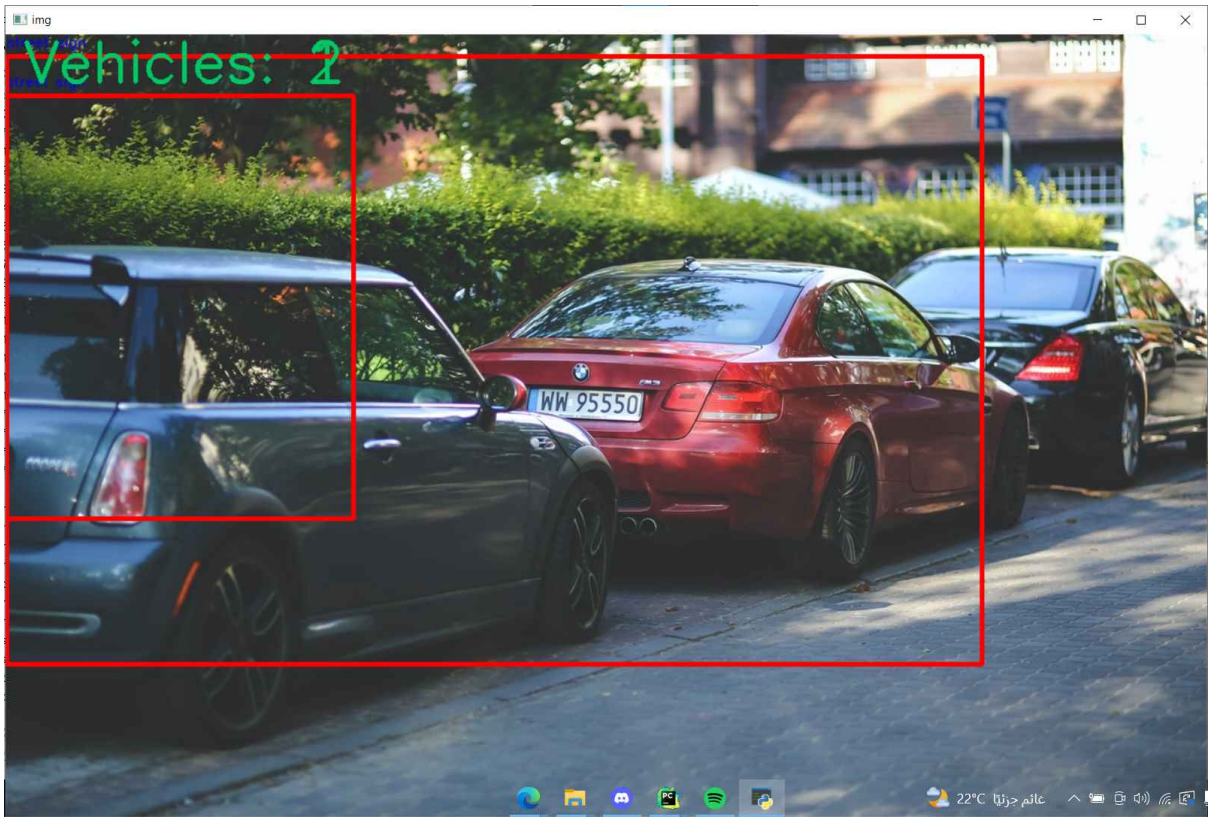



Figure 6.4: Results from an image using Faster R-CNN model

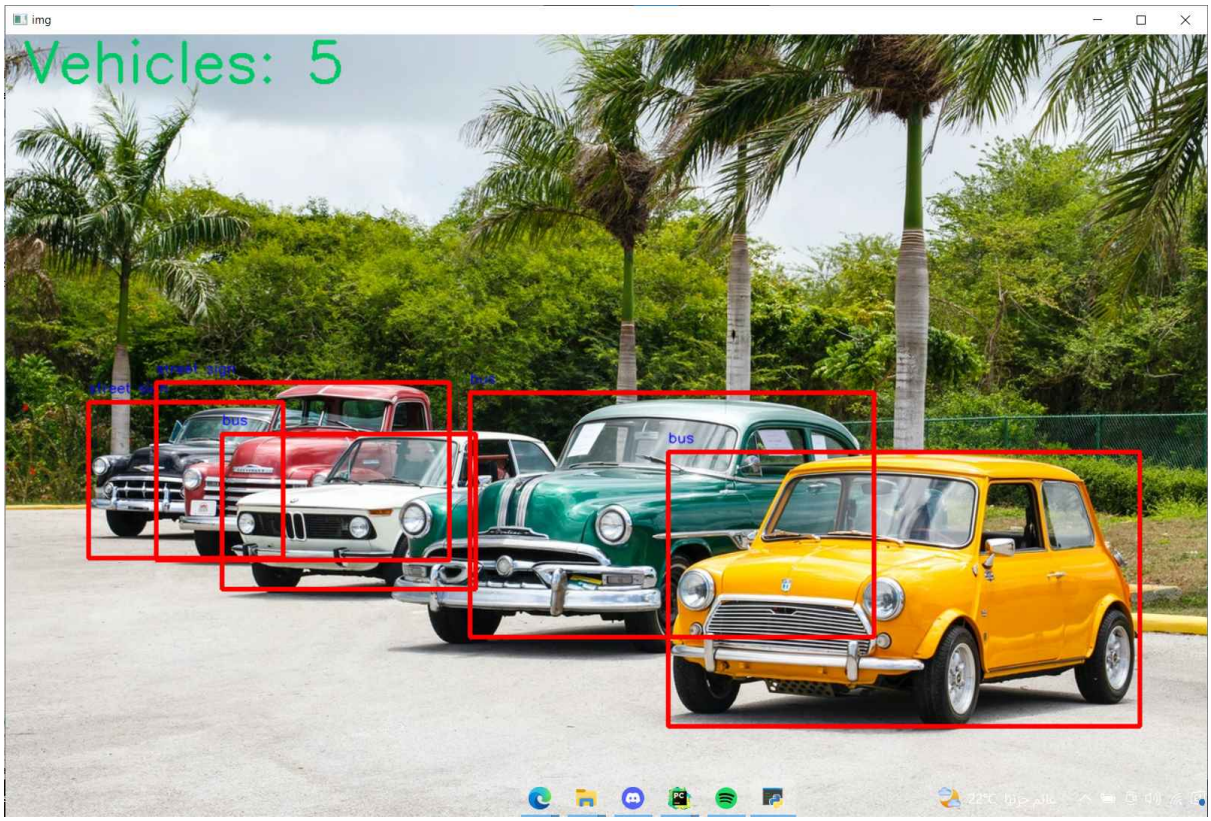



Figure 6.5: Results from an image using Res-Net model


Figure 6.6: Results from an image using SSD model

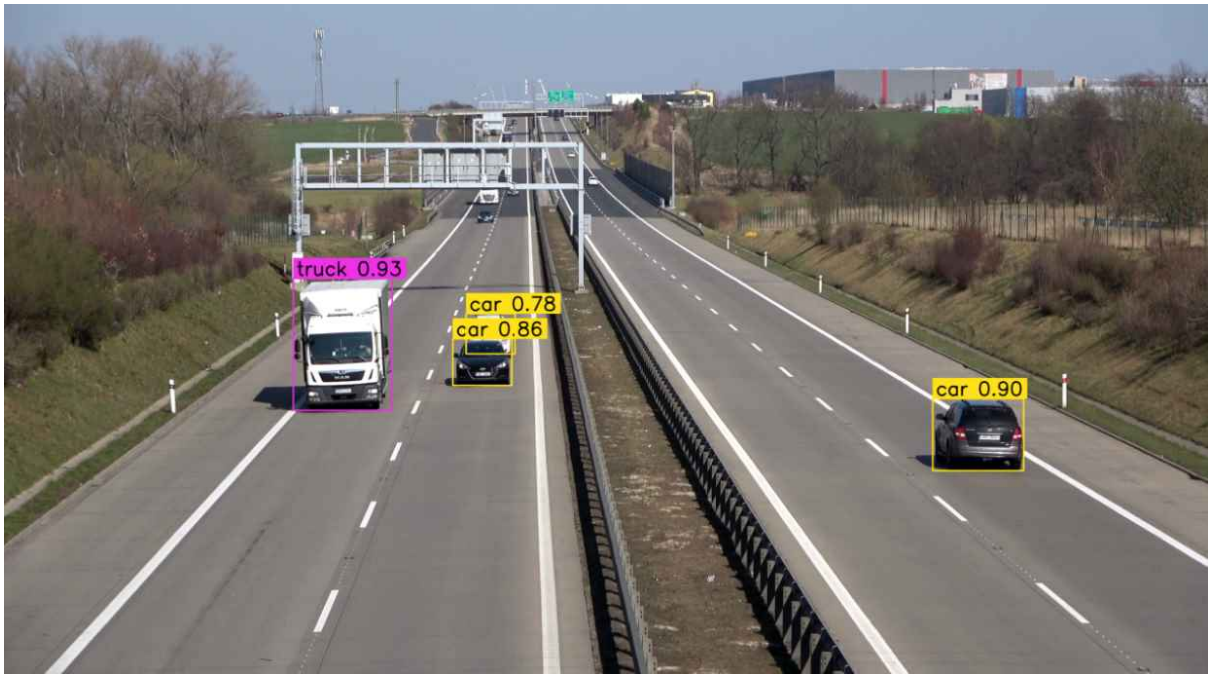



Figure 6.7: Results from image using YOLO v8 model

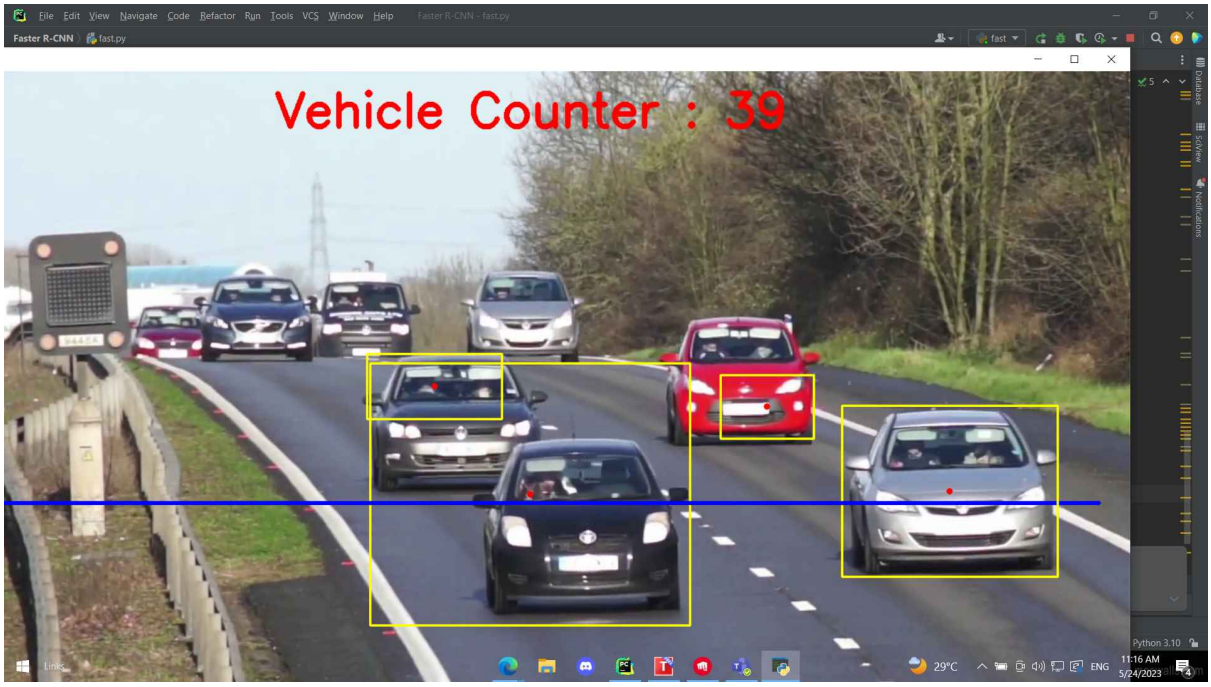



Figure 6.8: Results from video input using YOLO v8 model

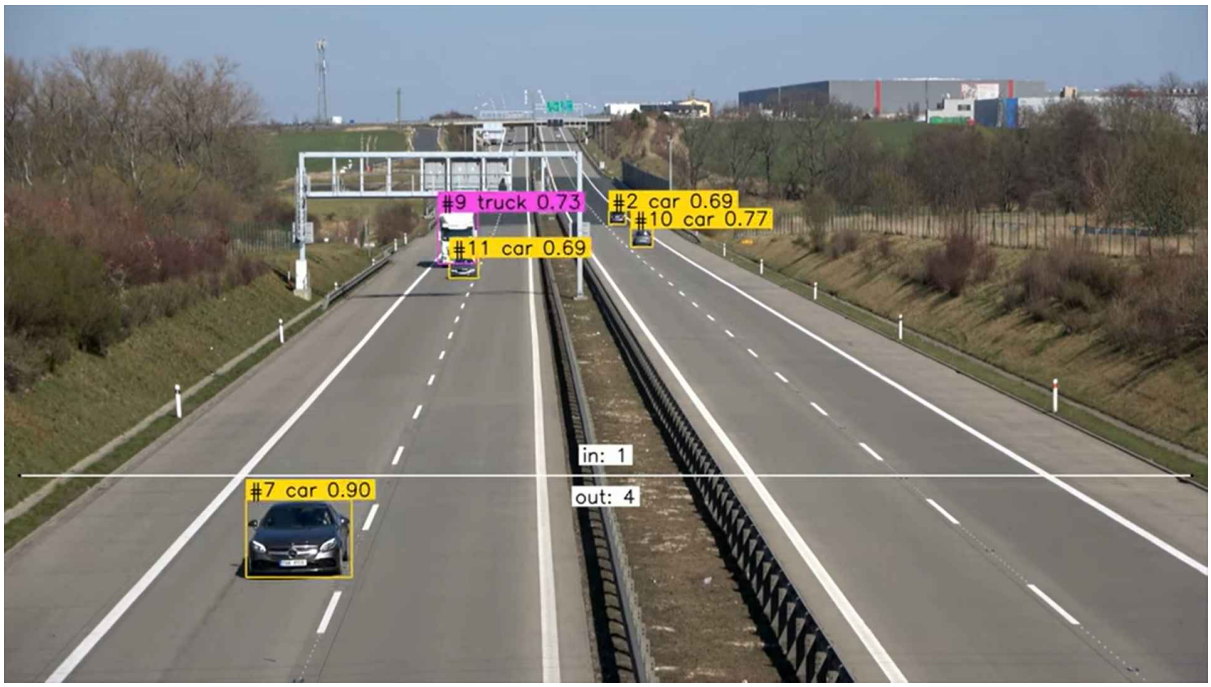



Figure 6.9: Results from video input using YOLO v8 model

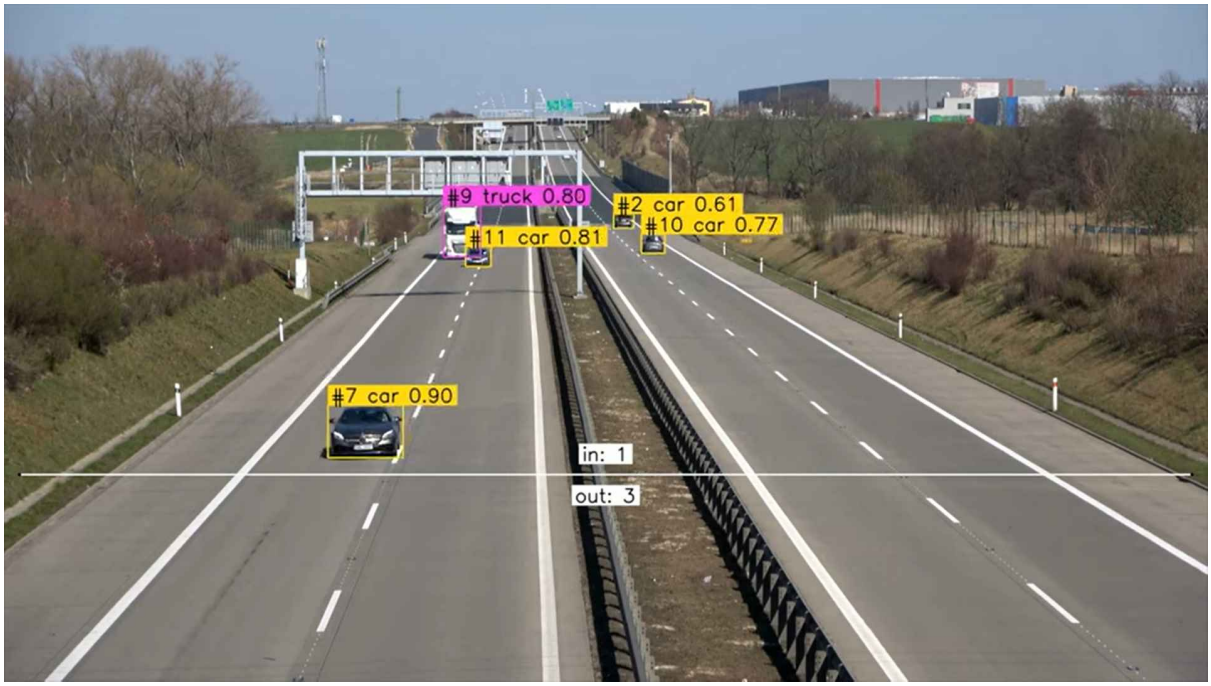



Figure 6.10: Results from video input using YOLO v8 model

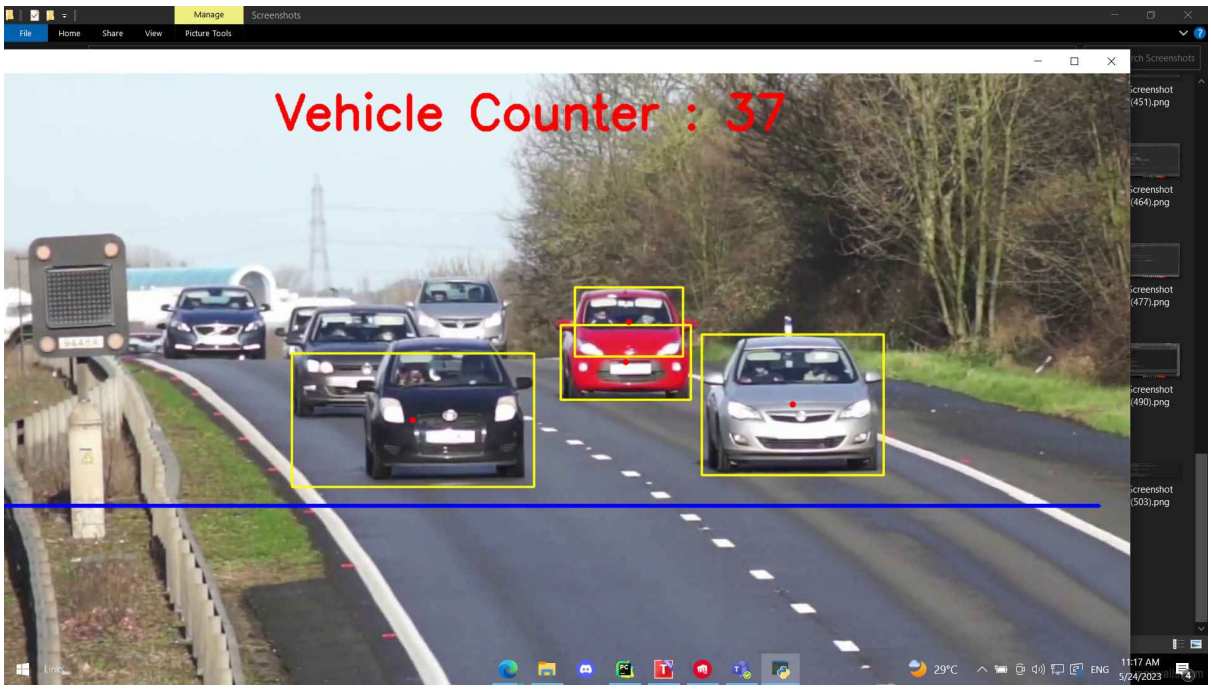



Figure 6.11: Results from video using SSD model

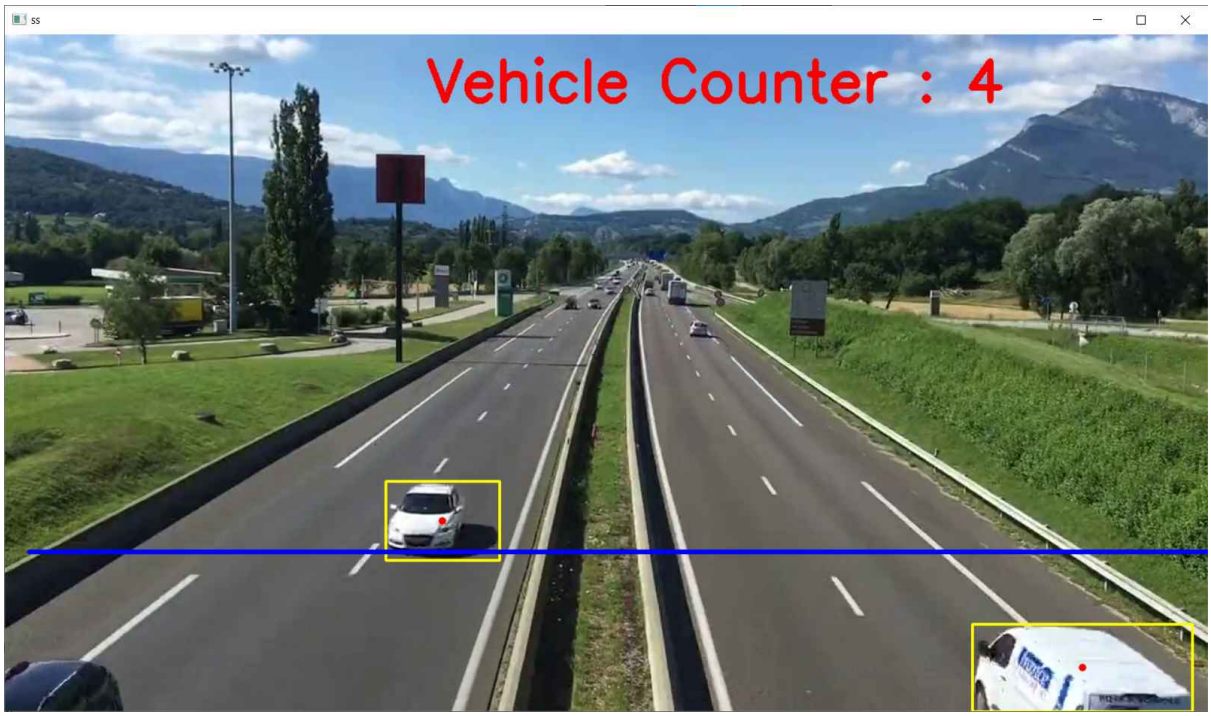



Figure 6.12: Results from video using SSD model

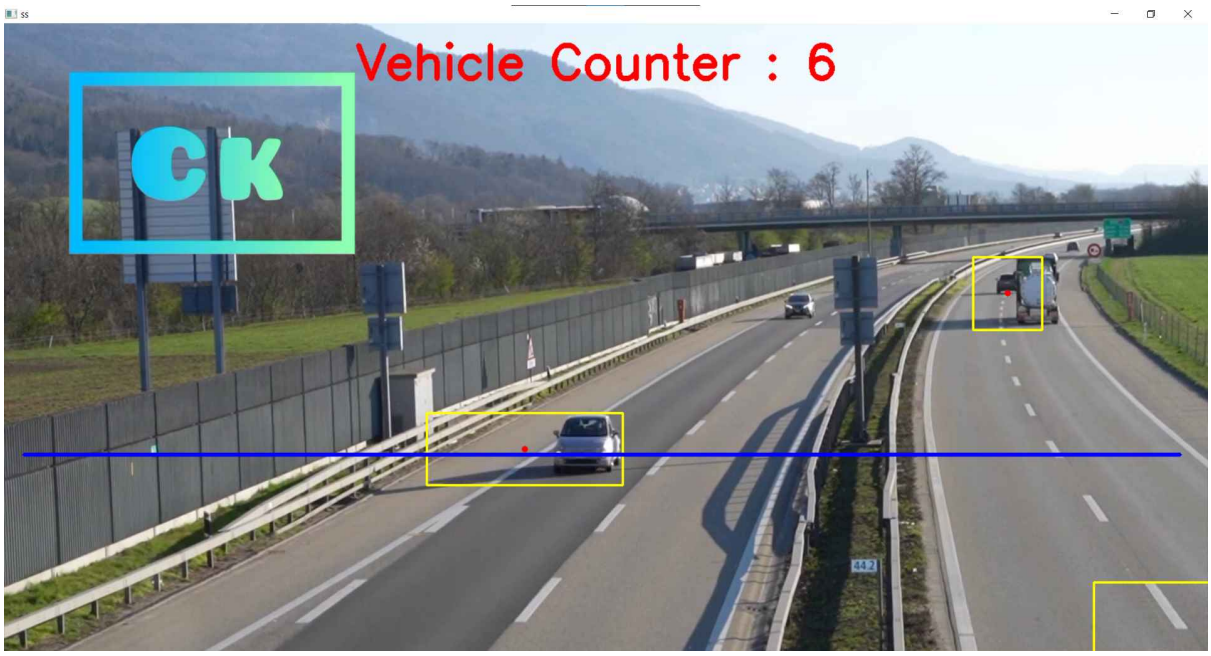



Figure 6.13: Results from video using Faster R-CNN model

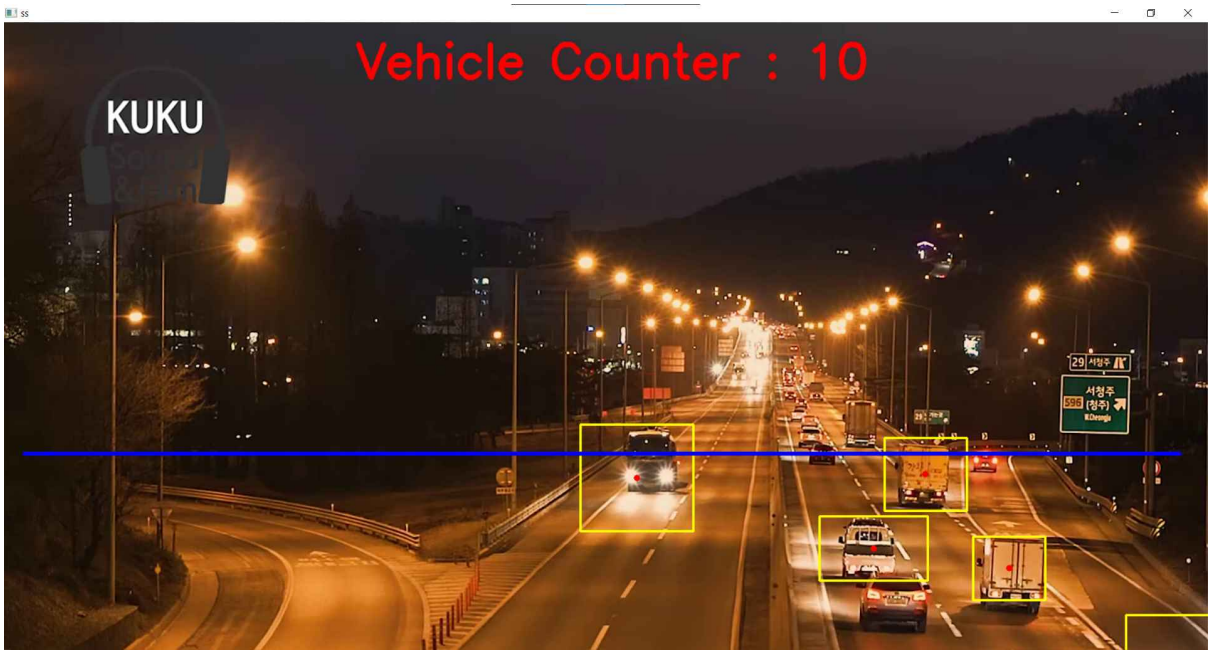



Figure 6.14: Results from video using Faster R-CNN model

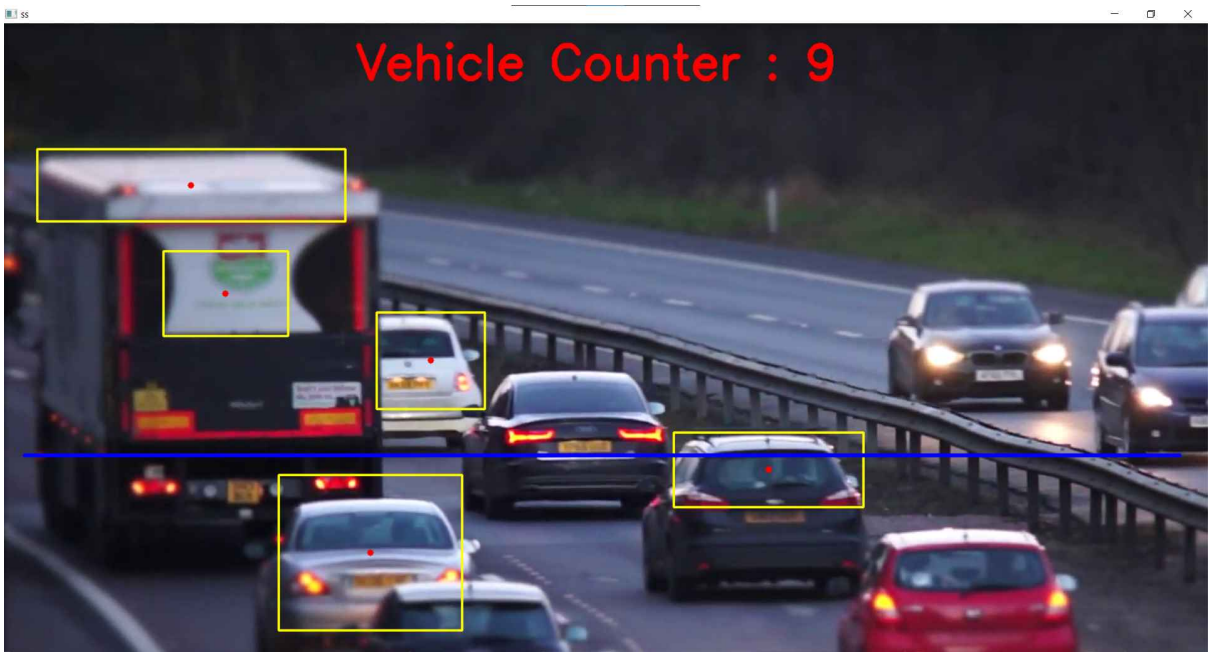



Figure 6.15: Results from video using Res-Net model

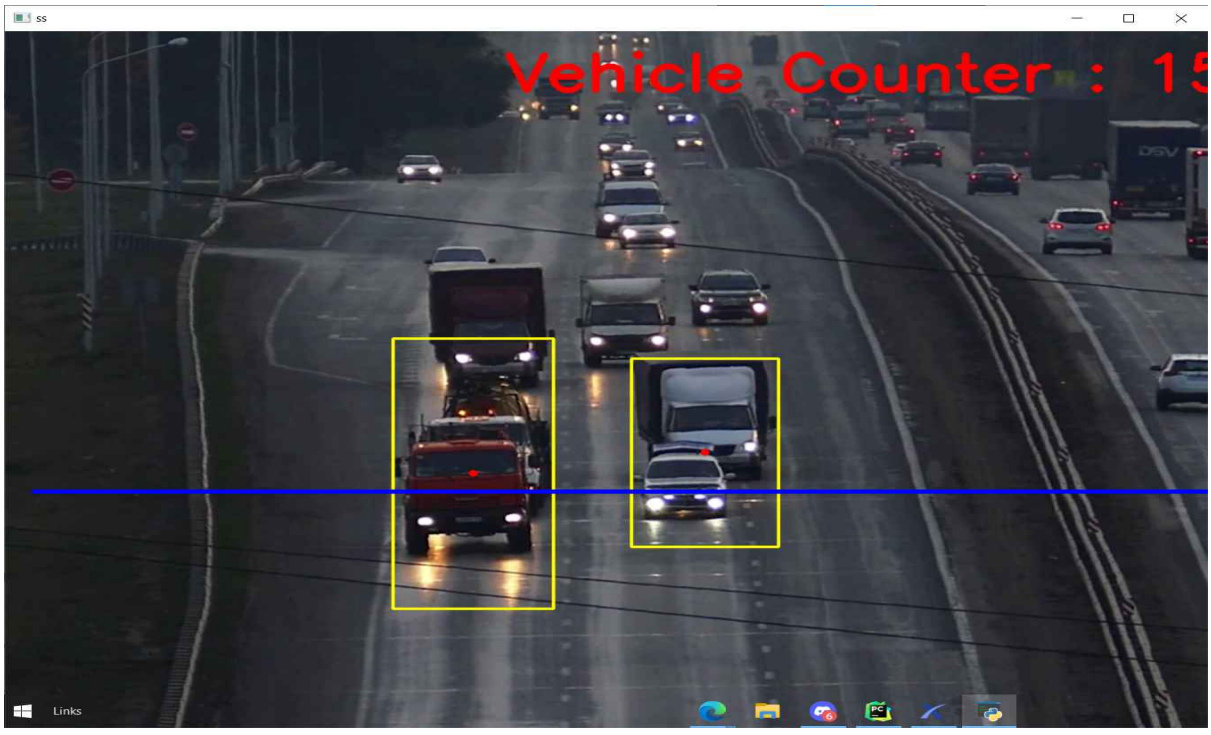



Figure 6.16: Results from video using Res-Net model

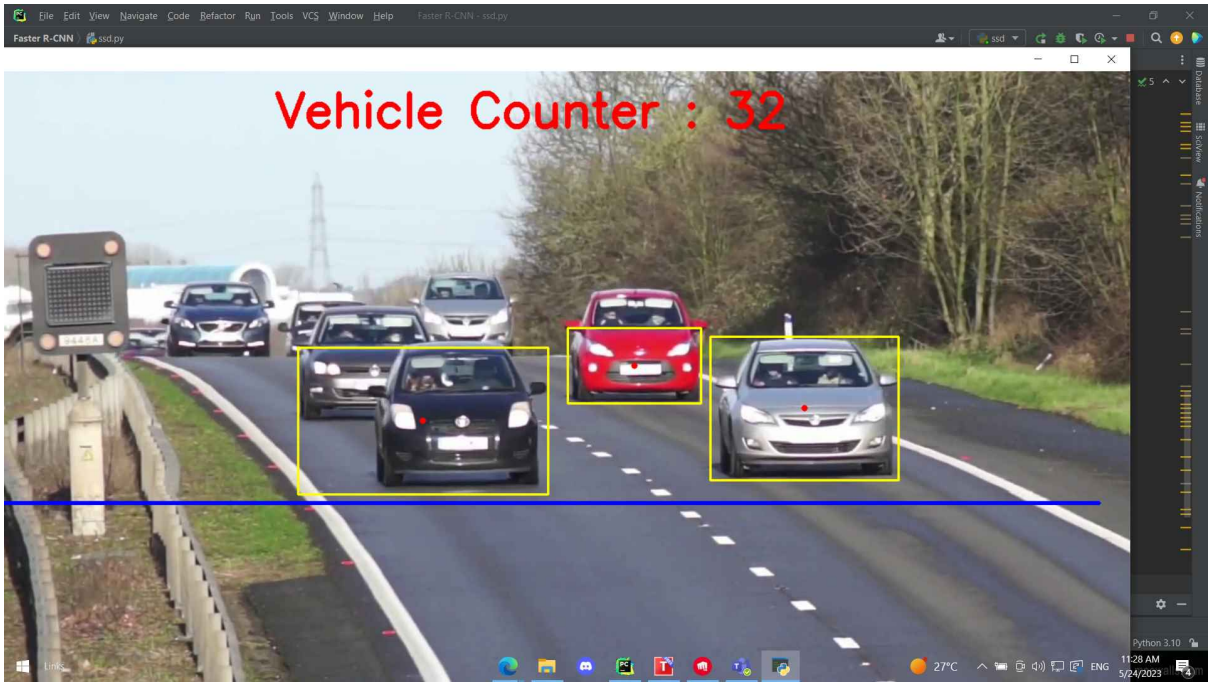



Figure 6.17: Results from video using Fast R-CNN model

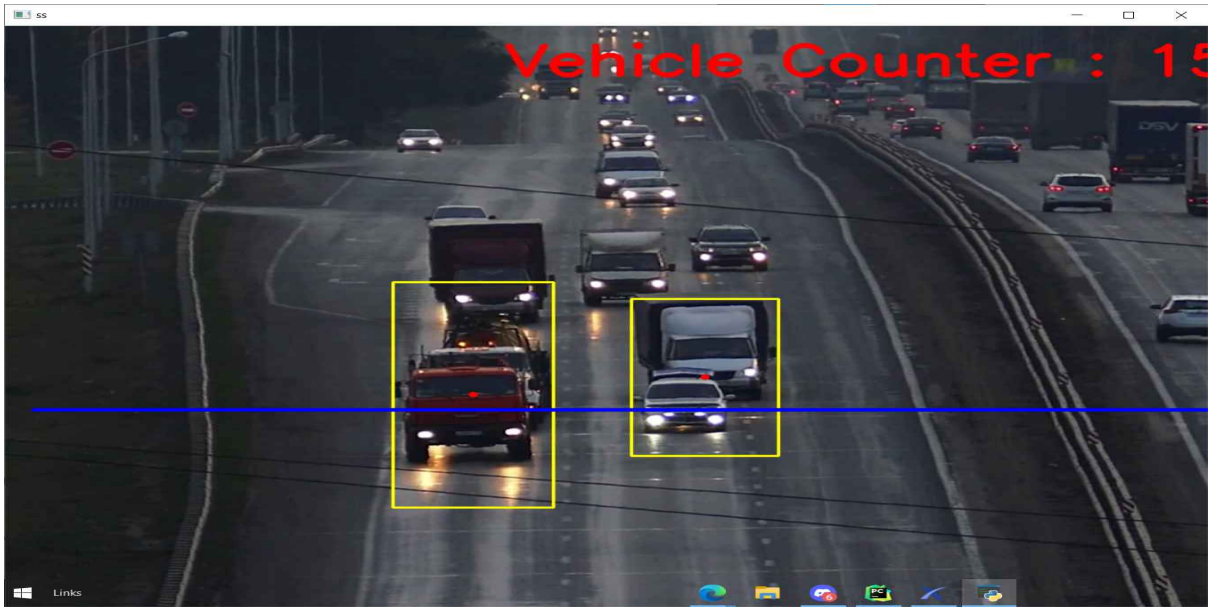



Figure 6.18: Results from video using Fast R-CNN model


Figure 6.19: Results from video using R-CNN model

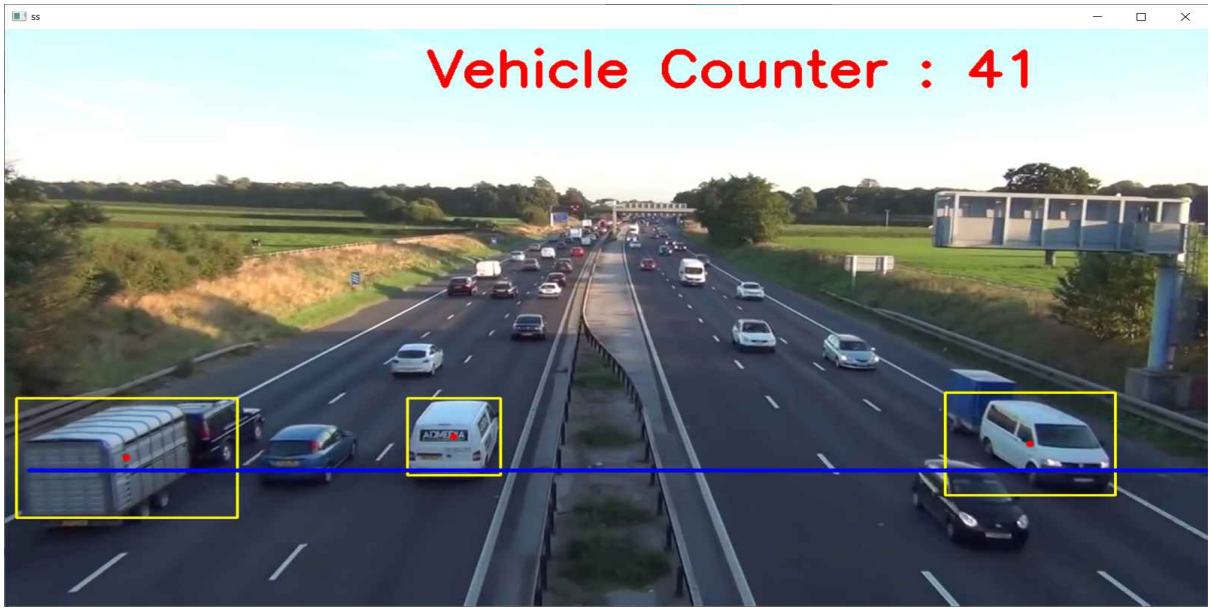



Figure 6.20: Results from video using R-CNN model

# Chapter 7

# Conclusion and future work

In this chapter, we will conclude our paper. And finally, we suggest future works.

## 7.1 Conclusion

We investigated Vehicle count and classification using Deep Learning models like R-CNN, Fast R-CNN, Faster R-CNN, Res-Net, SSD and YOLO v8. This is accomplished by first installing libraries and using the pre-trained model to detect vehicles. The goal of this study was to compare R-CNN, Fast R-CNN, Faster R-CNN, Res-Net, SSD, YOLO v8 and to find the model with the best performance, we tried different pre-trained models of the CNN family to improve its accuracy. Our findings revealed that YOLO v8 outperformed the other models.

## 7.2 Future work

The field of vehicle counting and classification has made significant advancements in recent years, driven by the increasing demand for intelligent transportation systems and traffic management. However, there are still exciting avenues for future work and research in this domain. first Deep learning has shown remarkable success in various computer vision tasks, including object detection and classification. Future work can focus on developing and refining deep learning architectures specifically designed for vehicle counting and classification. Architectures like convolutional neural networks (CNNs) can be explored to improve the accuracy and efficiency of vehicle detection and classification algorithms. second Most existing vehicle counting and classification systems are designed for controlled environments with clear visibility. Future research can tackle the challenges of unconstrained environments, such as adverse weather conditions, occlusions and varying lighting conditions. Robust algorithms that can handle these challenging scenarios would greatly benefit real-world applications. These future directions hold great potential

for advancing the field of vehicle counting and classification, leading to more accurate, efficient, and robust systems. By addressing the challenges mentioned earlier and harnessing the potential of emerging technologies, researchers can contribute to enhancing transportation systems, optimizing traffic flow, and improving road safety.

# Appendix

# Appendix A

# Lists

| | |
|---|---|
| **CNN** | Convolutional Neural network |
| **GPU** | Graphics processing unit |
| **ML** | Machine Learning |
| **DL** | Deep Learning |
| **R-CNN** | Region-based Convolutional Neural Networks |
| **SVM** | Support Vector Machine |
| **AI** | Artificial Intelligence |
| **MSE** | Mean Squared Error |
| **SSD** | Photon Unity Networking |
| **CV** | Computer Vision |
| **RNNs** | Recurrent Neural Networks |
| **COCO** | Common Objects in Context |
| **GANs** | Generative Adversarial Networks |
| **TPUs** | Tensor Processing Units |
| **YOLO** | You Only Look Once |

| | |
|---|---|
| **SIFT** | Scale-Invarient Feature Transform |
| **OCR** | Optical Character Recognition |
| **EAST** | Easy and Accurate Scene Text Detector |
| **CUDA** | Compute Unified Device Architecture |
| **RP** | Region Proposal |
| **PR** | Proposal Region |
| **FCL** | Fully Connected Layer |
| **VGG Net** | Visual Geometry Group Network |

# List of Figures

# Bibliography


[1] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2020.

[2] Akshansh Sharma, Firoj Khan, Deepak Sharma, Sunil Gupta, and FY Student. Python: the programming language of future. *International Journal of Innovative Research in Technology*, 6(12):115–118, 2020.

[3] KR Srinath. Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, 4(12):354–357, 2017.

[4] AL Sayeth Saabith, MMM Fareez, and T Vinothraj. Python current trend applications-an overview. popular web development in python. *Int. J. of Advanced Engineering and Research Development*, pages 6–10, 2019.

[5] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

[6] Samira Gholizadeh. Top popular python libraries in research. j robot auto res 3 (2), 142-145, 2022.

[7] Gregory Piatetsky. Python leads the 11 top data science. *Machine Learning Platforms: Trends and Analysis*, 2019, 2019.

[8] Stijn Heldens, Alessio Sclocco, Henk Dreuning, Ben van Werkhoven, Pieter Hijma, Jason Maassen, and Rob V van Nieuwpoort. litstudy: A python package for literature reviews. *SoftwareX*, 20:101207, 2022.

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[10] Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J Aljaaf. A systematic review on supervised and unsupervised machine learning algorithms for data science. *Supervised and unsupervised learning for data science*, pages 3–21, 2020.

[11] Akila Sarirete, Zain Balfagih, Tayeb Brahimi, Miltiadis D Lytras, and Anna Visvizi. Artificial intelligence and machine learning research: towards digital transformation at a global scale, 2021.

[12] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[13] Rahul Rai, Manoj Kumar Tiwari, Dmitry Ivanov, and Alexandre Dolgui. Machine learning in manufacturing and industry 4.0 applications. *International Journal of Production Research*, 59(16):4773–4778, 2021.

[14] Ziqiu Kang, Cagatay Catal, and Bedir Tekinerdogan. Machine learning applications in production lines: A systematic literature review. *Computers & Industrial Engineering*, 149:106773, 2020.

[15] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[16] Raffaele Pugliese, Stefano Regondi, and Riccardo Marini. Machine learning-based approach: Global trends, research directions, and regulatory standpoints. *Data Science and Management*, 4:19–29, 2021.

[17] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[18] Iqbal H Sarker. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):420, 2021.

[19] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021.

[20] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

[21] Hisham El-Amir and Mahmoud Hamdy. *Deep learning pipeline: building a deep learning model with TensorFlow*. Apress, 2019.

[22] Garima Gupta and Rahul Katarya. Research on understanding the effect of deep learning on user preferences. *Arabian Journal for Science and Engineering*, 46:3247–3286, 2021.

[23] Ajeet Ram Pathak, Manjusha Pandey, and Siddharth Rautaray. Application of deep learning for object detection. *Procedia computer science*, 132:1706–1717, 2018.

[24] Khadijeh Alibabaei, Pedro D Gaspar, Tânia M Lima, Rebeca M Campos, Inês Girão, Jorge Monteiro, and Carlos M Lopes. A review of the challenges of using deep learning algorithms to support decision-making in agricultural activities. *Remote Sensing*, 14(3):638, 2022.

[25] Giovanna Castellano and Gennaro Vessio. Deep learning approaches to pattern extraction and recognition in paintings and drawings: An overview. *Neural Computing and Applications*, 33(19):12263–12282, 2021.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[27] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and trends® in signal processing*, 7(3–4):197–387, 2014.

[28] Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. Deep learning for ai. *Communications of the ACM*, 64(7):58–65, 2021.

[29] Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic Programming and Evolvable Machines*, 19(1-2):305–307, 2018.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[31] Stephen Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[34] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis, et al. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

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

[36] Junyi Chai, Hao Zeng, Anming Li, and Eric WT Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021.

[37] Ahmed Fathi Bekhit and Ahmed Fathi Bekhit. Introduction to computer vision. *Computer Vision and Augmented Reality in iOS: OpenCV and ARKit Applications*, pages 1–20, 2022.

[38] Muhammad Haroon. An introduction to computer vision. 09 2021.

[39] Youzi Xiao, Zhiqiang Tian, Jiachen Yu, Yinshu Zhang, Shuai Liu, Shaoyi Du, and Xuguang Lan. A review of object detection based on deep learning. *Multimedia Tools and Applications*, 79:23729–23791, 2020.

[40] Mukkamala Rohith Sri Sai, Sindhusha Rella, and Sainagesh Veeravalli. *OBJECT DETECTION AND IDENTIFICATION A Project Report*. PhD thesis, 11 2019.

[41] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

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

[44] Aditi and Aman Dureja. A review: Image classification and object detection with deep learning. In Xiao-Zhi Gao, Rajesh Kumar, Sumit Srivastava, and Bhanu Pratap Soni, editors, *Applications of Artificial Intelligence in Engineering*, pages 69–91, Singapore, 2021. Springer Singapore.

[45] Juan K Leonard. Image classification and object detection algorithm based on convolutional neural network. *Science Insights*, 31(1):85–100, 2019.

[46] Aditi and Aman Dureja. A review: Image classification and object detection with deep learning. In Xiao-Zhi Gao, Rajesh Kumar, Sumit Srivastava, and Bhanu Pratap Soni, editors, *Applications of Artificial Intelligence in Engineering*, pages 69–91, Singapore, 2021. Springer Singapore.

[47] SV Viraktamath, Madhuri Yavagal, and Rachita Byahatti. Object detection and classification using yolov3. *Int. J. Eng. Res. Technol. IJERT*, 10, 2021.

[48] Thorsten Hoeser, Felix Bachofer, and Claudia Kuenzer. Object detection and image segmentation with deep learning on earth observation data: A review—part ii: Applications. *Remote Sensing*, 12(18):3053, Sep 2020.

[49] Mukkamala Rohith Sri Sai, Sindhusha Rella, and Sainagesh Veeravalli. *OBJECT DETECTION AND IDENTIFICATION A Project Report*. PhD thesis, 11 2019.

[50] Chen Zhongshan, Feng Xinning, Adhiyaman Manickam, and VE Sathishkumar. Facial landmark detection using artificial intelligence techniques. *Annals of Operations Research*, pages 1–19, 2021.

[51] Fatih Porikli and Alper Yilmaz. *Object Detection and Tracking*, volume 409. 01 2012.

[52] Huansheng Song, Haoxiang Liang, Huaiyu Li, Zhe Dai, and Xu Yun. Vision-based vehicle detection and counting system using deep learning in highway scenes. *European Transport Research Review*, 11, 12 2019.

[53] Sana Ali, Khalid Iqbal, Saira Khan, Rehan Tariq, and Qazi Aqil. A review on text detection techniques. *VFAST Transactions on Software Engineering*, 8, 11 2015.

[54] Kalanit Grill-Spector and Nancy Kanwisher. Visual recognition: As soon as you know it is there, you know what it is. *Psychological Science*, 16(2):152–160, 2005.

[55] Jerome S Bruner and Mary C Potter. Interference in visual recognition. *Science*, 144(3617):424–425, 1964.

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

[57] Bhavneet Kaur, Meenakshi Sharma, Mamta Mittal, Amit Verma, Lalit Mohan Goyal, and D Jude Hemanth. An improved salient object detection algorithm combining background and foreground connectivity for brain image analysis. *Computers & Electrical Engineering*, 71:692–703, 2018.

[58] Maojin Sun, Yan Wang, Teng Li, Jing Lv, and Jun Wu. Vehicle counting in crowded scenes with multi-channel and multi-task convolutional neural networks. *Journal of Visual Communication and Image Representation*, 49:412–419, 2017.

[59] Zhe Dai, Huansheng Song, Xuan Wang, Yong Fang, Xu Yun, Zhaoyang Zhang, and Huaiyu Li. Video-based vehicle counting framework. *IEEE Access*, 7:64460–64470, 2019.

[60] Zhongji Liu, Wei Zhang, Xu Gao, Hao Meng, Xiao Tan, Xiaoxing Zhu, Zhan Xue, Xiaoqing Ye, Hongwu Zhang, Shilei Wen, et al. Robust movement-specific vehicle counting at crowded intersections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 614–615, 2020.

[61] Chunlong Zhang, Kaifei Zhang, Luzhen Ge, Kunlin Zou, Song Wang, Junxiong Zhang, and Wei Li. A method for organs classification and fruit counting on pomegranate trees based on multi-features fusion and support vector machine by 3d point cloud. *Scientia Horticulturae*, 278:109791, 2021.

[62] Suzanna Parkinson, Greg Ongie, and Rebecca Willett. Linear neural network layers promote learning single- and multiple-index models. 05 2023.

[63] Enis Çoban. Neural networks and their applications. 02 2016.

[64] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020.

[65] Feiping Nie, Hu Zhanxuan, and Xuelong Li. An investigation for loss functions widely used in machine learning. *Communications in Information and Systems*, 18:37–52, 01 2018.

[66] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[67] Faisal Mehmood, Shabir Ahmad, and Taeg Keun Whangbo. An efficient optimization technique for training deep neural networks. *Mathematics*, 11(6):1360, 2023.

[68] Haoran Sun, Xiangyi Chen, Qingjiang Shi, Mingyi Hong, Xiao Fu, and Nikos D Sidiropoulos. Learning to optimize: Training deep neural networks for wireless resource management. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–6. IEEE, 2017.

[69] Randall J Erb. Introduction to backpropagation neural network computation. *Pharmaceutical research*, 10:165–170, 1993.

[70] Massimo Buscema. Back propagation neural networks. *Substance use misuse*, 33:233–70, 02 1998.

[71] Zhichen Wang and Hongliang Li. Research on a convolution operation method based on domain transformation in deep learning. *Journal of Physics: Conference Series*, 1550:032132, 05 2020.

[72] Anamika Dhillon and Gyanendra K Verma. Convolutional neural network: a review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, 9(2):85–112, 2020.

[73] Timea Bezdan and Nebojsa Bacanin. Convolutional neural network layers and architectures. pages 445–451, 01 2019.

[74] Phil Kim and Phil Kim. Convolutional neural network. *MATLAB deep learning: with machine learning, neural networks and artificial intelligence*, pages 121–147, 2017.

[75] Saad Albawi, Tareq Abed Mohammed, and Saad ALZAWI. Understanding of a convolutional neural network. 08 2017.

[76] Ivars Namatevs. Deep convolutional neural networks: Structure, feature extraction and training. *Information Technology and Management Science*, 20, 12 2017.

[77] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[78] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

[79] Li Shen, Zhouchen Lin, and Qingming Huang. Relay backpropagation for effective learning of deep convolutional neural networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*, pages 467–482. Springer, 2016.

[80] Laurent Boué. Deep learning for pedestrians: backpropagation in cnns. *arXiv preprint arXiv:1811.11987*, 2018.

[81] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[82] Nvidia corporation. (2021). nvidia gpu architecture, 2021.

[83] Joseph Howse and Joe Minichino. *Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning*. Packt Publishing Ltd, 2020.

[84] Santanu Pattanayak. *Pro Deep Learning with Tensorflow 2.0: A Mathematical Approach to Advanced Artificial Intelligence in Python*. Springer, 2023.

[85] Daniel Lélis Baggio. *Mastering OpenCV with practical computer vision projects*. Packt Publishing Ltd, 2012.

[86] Nicholas Wilt. *The cuda handbook: A comprehensive guide to gpu programming*. Pearson Education, 2013.

[87] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*, 29, 2016.

[88] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[89] Hongkai Zhang, Hong Chang, Bingpeng Ma, Naiyan Wang, and Xilin Chen. Dynamic r-cnn: Towards high quality object detection via dynamic training. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*, pages 260–275. Springer, 2020.

[90] Adrian Rosebrock. R-cnn object detection with keras, tensorflow, and deep learning. 2020.

[91] Amlan Dutta, Ahmad Atik, Mriganka Bhadra, Abhijit Pal, Md Akram Khan, and Rupak Chakraborty. Detection of objects using a fast r-cnn-based approach. In *Modeling, Simulation and Optimization: Proceedings of CoMSO 2021*, pages 295–307. Springer, 2022.

[92] Yun Ren, Changren Zhu, and Shunping Xiao. Object detection based on fast/faster rcnn employing fully convolutional architectures. *Mathematical Problems in Engineering*, 2018:1–7, 01 2018.

[93] Qihang Wang and Junbao Zheng. Research on face detection based on fast r-cnn. In *Recent Developments in Intelligent Computing, Communication and Devices: Proceedings of ICCD 2017*, pages 79–85. Springer, 2019.

[94] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[95] Quanfu Fan, Lisa Brown, and John Smith. A closer look at faster r-cnn for vehicle detection. In *2016 IEEE intelligent vehicles symposium (IV)*, pages 124–129. IEEE, 2016.

[96] Hoanh Nguyen. Improving faster r-cnn framework for fast vehicle detection. *Mathematical Problems in Engineering*, 2019:1–11, 2019.

[97] Shaohua Wan and Sotirios Goudos. Faster r-cnn for multi-class fruit detection using a robotic vision system. *Computer Networks*, 168:107036, 2020.

[98] Jiazhi Liang. Image classification based on resnet. *Journal of Physics: Conference Series*, 1634:012110, 09 2020.

[99] Devvi Sarwinda, Radifa Hilya Paradisa, Alhadi Bustamam, and Pinkie Anggia. Deep learning in image classification using residual network (resnet) variants for detection of colorectal cancer. *Procedia Computer Science*, 179:423–431, 2021.

[100] Yoshiki Tanaka and Yuichi Kageyama. Imagenet/resnet-50 training in 224 seconds.

[101] Ashwani Kumar, Zuopeng Justin Zhang, and Hongbo Lyu. Object detection in real time based on improved single shot multi-box detector algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2020(1):1–18, 2020.

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