FEDERAL UNIVERSITY OF SANTA CATARINA
DEPARTMENT OF AUTOMATION AND SYSTEMS ENGINEERING
AUTOMATION AND SYSTEMS ENGINEERING GRADUATE PROGRAM

BRUNO DO NASCIMENTO BESERRA

A VEHICLE COUNTER APPROACH FOR THE NIGHT PERIOD OF THE DAY BASED IN NEURAL NETWORKS

A STUDY CASE USING YOLOV4 AND DEEPSORT

FLORIANÓPOLIS/SC
2023

BRUNO DO NASCIMENTO BESERRA

A VEHICLE COUNTER APPROACH FOR THE NIGHT PERIOD OF THE DAY BASED IN NEURAL NETWORKS

A STUDY CASE USING YOLOV4 AND DEEPSORT

Dissertation presented to the Automation and Systems Engineering Graduate Program of the Universidade Federal de Santa Catarina in partial fulfillment of the requirements for the degree of Master in Automation and Systems Engineering.
Advisor: Prof. Marcelo Ricardo Stemmer, Dr.
Co-advisor: Prof. Mauricio Edgar Stivanello, Dr.

FLORIANÓPOLIS/SC
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Beserra, Bruno do Nascimento
A VEHICLE COUNTER APPROACH FOR THE NIGHT PERIOD OF THE DAY BASED IN NEURAL NETWORKS : A STUDY CASE USING YOLOV4 AND DEEPSORT / Bruno do Nascimento Beserra ; orientador, Marcelo Ricardo Stemmer, coorientador, Mauricio Edgar Stivanello, 2023.
78 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Visão Computacional. 3. YOLOv4. 4. DeepSORT. 5. Sistema de Transporte Inteligente. I. Stemmer, Marcelo Ricardo. II. Stivanello, Mauricio Edgar. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Bruno do Nascimento Beserra

A VEHICLE COUNTER APPROACH FOR THE NIGHT PERIOD OF THE DAY BASED IN NEURAL NETWORKS
A STUDY CASE USING YOLOV4 AND DEEPSORT

The present dissertation in the Master level was evaluated and approved by the examining board composed of the following members:

Marcelo Ricardo Stemmer, Dr
Universidade Federal de Santa Catarina

Mario Lucio Roloff, Dr
Instituto Federal Catarinense

Eric Aislan Antonelo, Dr
Universidade Federal de Santa Catarina

This Dissertation is recommended in partial fulfillment of the requirements for the degree of "Master in Automation and Systems Engineering", which has been approved in its present form by the Automation and Systems Engineering Graduate Program Master in Automation and Systems Engineering.

Documento assinado digitalmente
Julio Elias Normey Rico
Data: 17/10/2023 15:39:02-0300
CPF: ***.840.859-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Julio Elias Normey Rico, Dr.
Graduate Program Coordinator

Documento assinado digitalmente
Marcelo Ricardo Stemmer
Data: 17/10/2023 12:37:16-0300
CPF: ***.093.279-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Marcelo Ricardo Stemmer, Dr.
Advisor

Documento assinado digitalmente
MAURICIO EDGAR STIVANELLO
Data: 17/10/2023 13:27:59-0300
CPF: ***.390.629-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Mauricio Edgar Stivanello, Dr.
Co-advisor

Florianópolis, 19 de Julho de 2023.

This thesis is dedicated to my mom, dad and my late dog Yughi, without their support and love during my life, I would not have become the person and I am now and the completion of this project would not have been possible. Thanks for making me see this through the end. I love you.

Acknowledgements

I would like to begin by expressing my gratitude to God for granting me life and providing me with the courage and strength to overcome the challenges encountered during the completion of this project.

I would like to extend my gratitude to my parents João William Beserra Filho, Ana Cristina Silva do Nascimento Beserra and my sister Williane do Nascimento Beserra for their unwavering support, love, and guidance throughout all my decisions, providing me with the security and support I needed to overcome challenges and pursue my goals understanding that my path may not always align with their own aspirations. I am today, and I am forever indebted to them.

I also want to extend my gratitude to my late dog Yughi, who was always by my side during moments when no one else was around when he was alive. He provided unwavering companionship, love, and support throughout his journey and though Yughi is no longer physically present, his memory and the bond we shared will forever remain in my heart.

I am grateful for the guidance and expertise provided by my advisor Dr. Marcelo Ricardo Stemmer and co-advisor Dr. Maurício Edgar Stivanello. Their insightful feedback and their willingness to assist me whenever needed have been immensely helpful in shaping this research.

I would like to express my deepest gratitude to Jacqueline Ferreira Pedroso that stood by my side, providing a listening ear, a comforting embrace, and words of encouragement. I am grateful for the sacrifices she has made, the patience she has shown, and the unwavering support she has provided. Your love and care have been constant reminders of what truly matters. I love you.

I am also immensely grateful to the amazing friends Marcelo Faustino Viana Junior, Pedro Henrique Braga Moraes, Douglas Braz Maciel, Gustavo Henrique Farias Bezerra, João William Bezerra Neto and Bruno de Souza Rodrigues who have been by my side. I am grateful for the countless memories we have created together, from the outgoings to eat and laughter-filled conversations to the philosophical discussions we've engaged in about life. As I reflect on our shared journey, our friendship has stood the test of time, and I am confident it will continue to thrive and evolve in the years to come. Thank you for being there for celebrating my triumphs and comforting me during my darkest moments. Your friendship is truly a gift that I cherish with all my heart.

During times of crisis, everyone's presence and willingness to lend a helping hand have brought me solace and reassurance. Whether it was offering a listening ear, providing practical advice, or simply being there for me, your support has been invaluable. One more time, I would like to thank you all for everything.

With deepest gratitude to each one of you, Bruno do Nascimento Beserra

The only real failure in life is giving up. On looking back let it stand to our credit in life's balance sheet that at least we tried, and tried hard.

—A.G. Street

A VEHICLE COUNTER APPROACH FOR THE NIGHT PERIOD OF THE DAY BASED IN NEURAL NETWORKS

A STUDY CASE USING YOLOV4 AND DEEPSORT

Bruno do Nascimento Beserra

Abstract

There is no doubt that with our community, the number of cars in the streets is growing lately and to prevent that to launch chaos in the streets affecting both people and the economy of the cities, several companies and institutions are studying new approaches to lessen this issue adverse. Hence, counting vehicles became a trend topic around researchers of this field, since this data is highly valuable to understand the situation of the roads. The advances of technology are another key factor that helped to increase the number of Intelligent Transport Systems (ITS) that use both hardware and software to develop new ways to learn and adapt to the new situations we are facing in the traffic and many original strategies related to object detection appeared in the literature lately. These strategies focus on day time object detection on account of the higher performance of the cameras during this period when compared to the night shift. Therefore, this work studies the viability of implementing a ITS to detect and count vehicles at night time shift based on a convolutional neural network (CNN) model and the possibility to use monitoring cameras already installed in the cities to train our models for the process. To accomplish these goals this project was divided into three stages, detection, tracking and counting. The detection part is based on a YOLOv4 model and was trained with a dataset composed of 25000 night images taken from surveillance cameras of five different cities of the World. The tracking step was performed with the DeepSORT technique and the counting step was made with the virtual line procedure.

Keywords: Traffic. Object Detection. Vehicle Counting. YOLOv4. ITS. Convolutional Neural Networks. Smart Cities.

CONTADOR DE VEÍCULOS PARA O PERÍODO NOTURNO DO DIA BASEADO EM REDES NEURAIS

ESTUDO DE CASO UTILIZANDO-SE DE YOLOV4 E DEEPSORT

Bruno do Nascimento Beserra

Resumo

Não há motivos para se duvidar de que junto da nossa sociedade, o número de carros nas ruas tem crescido constantemente e para prevenir que o caos se instaure nas ruas afetando tanto as pessoas quanto a economia das cidades, diversas empresas e instituições têm estudado novas formas de minimizar essa adversidade. Consequentemente, a contagem de veículos virou um tópico importante entre os pesquisadores dessa área, já que esse dado é extremamente valioso para o entendimento da situação das estradas. Os avanços da tecnologia é outro fator importante que ajudou a crescer o número de Sistemas de Transporte Inteligentes (STI) que usa tanto hardware quanto software para desenvolver novos métodos de aprendizagem e ultimamente, diversas estratégias que se utilizam de detecção de objetos estão surgindo na academia de forma a nos adaptar às novas situações que estamos enfrentando. Essas soluções se baseiam na detecção de objetos durante o período diurno por conta da melhor performance provinda pelas câmeras durante este período do dia. Portanto, este trabalho estuda a viabilidade de implementação de um SIT para detectar e contar veículos durante o período noturno baseado em um modelo de rede neural convolucional (CNN) e a possibilidade do uso de câmeras de monitoramento já instaladas em cidades para este propósito. Para atingir esse objetivo, esse projeto foi dividido em três estágios, a detecção, o rastreio de objetos e a contagem. A etapa de detecção é baseada num modelo YOLOv4 e foi treinada com um dataset composto de 25000 imagens de veículos noturnos provindos de câmeras de vigilância de cinco diferentes cidades do planeta. A parte de rastreio foi realizada utilizando a técnica de DeepSORT enquanto a parte de contagem foi realizada utilizando-se de uma técnica envolvendo uma linha imaginária.

Palavras-chave: Transito. Detecção de Objetos. Contagem de Veículos. YOLOv4. ITS. Redes Neurais Convolucionais. Cidades Inteligentes.

CONTADOR DE VEÍCULOS PARA O PERÍODO NOTURNO DO DIA BASEADO EM REDES NEURAIS

ESTUDO DE CASO UTILIZANDO-SE DE YOLOV4 E DEEPSORT

Bruno do Nascimento Beserra

Resumo Expandido

Introdução. Ultimamente, o ITS (Sistema de Transporte Inteligente) tem demonstrado grande interesse em soluções de visão computacional. Com o aumento significativo de acidentes de trânsito, congestionamento e poluição [54], diversas empresas e instituições têm estudado novas formas de minimizar essas adversidades, onde essa nova tecnologia vem se mostrando uma grande aliada. Entretanto, o número de aplicações desenvolvidas voltadas ao período noturno do dia ainda são escassas na academia devido a melhor performance de câmeras durante o dia. Neste trabalho, é explorado a viabilidade de uma abordagem para contagem de veículos durante à noite com base em YOLOv4 e DeepSORT, que são técnicas de ponta para fins de detecção e rastreamento, respectivamente. Além disso, fornecemos uma análise da eficácia das câmeras de vigilância para fornecer dados para o banco de dados, bem como os resultados da contagem realizada sob a perspectiva delas durante a noite.

Objetivos. O principal objetivo desta dissertação é desenvolver uma aplicação robusta para detectar e contar veículos durante o período noturno do dia, dessa forma, poderemos criar um sistema de contagem que funcionaria continuamente pelo período de 24 horas seguidas ao longo do período estipulado da contagem [3]. Além do objetivo principal, este trabalho visa investigar a viabilidade de implementar este sistema computacional com uma câmera como seus olhos para esse tipo de situação e inspecionar a confiabilidade de bases de dados fornecidas pelas câmeras de vigilância das cidades.

Contribuições e Resultados. Como mencionado anteriormente, este trabalho contribui para o estado da arte ao desenvolver uma aplicação robusta para detectar e contar veículos durante o período noturno do dia. Um modelo de detecção foi treinado com base em imagens provindas de câmeras de vigilância durante o período noturno e analisado a confiabilidade de detecção deste modelo quando disposto a vídeos que ele desconhecia. Diante deste procedimento, se obteve um resultado favorável de detecção de 91.23% mAP no treinamento do modelo. Também foi adicionado ao modelo a técnica de rastreamento DeepSORT de forma a adquirir o caminho percorrido por cada objeto detectado pelo sistema e um sistema de contagem baseado em detecção de intersecção de retas de forma a obter a contagem precisa dos objetos que passaram por determinado ponto previamente marcados nos vídeos.

Cosiderações Finais. O sistema desenvolvido obteve excelente desempenho de detecção ao longo dos testes mesmo com a baixa iluminação presente nos locais. Já o rastreador não teve resultados tão animadores quanto os da detecção, devido às complexas formas de luzes e sombras presentes no ambiente. Felizmente, o baixo desempenho do rastreador não afetou o resultado do contador, que atingiu resultados superiores a 81% em todos os testes realizados, mostrando o grande potencial que este sistema pode oferecer para aplicações futuras. Ao longo desta dissertação, estão dispostos para consulta todos os passos tomados durante este estudo.

Palavras-chave: Transito. Detecção de Objetos. Contagem de Veículos. YOLOv4. ITS. Redes Neurais Convolucionais. Cidades Inteligentes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As humanity discovered the immeasurable power of computer's technology, we have been seeking to expand our horizons. With applications from the most basic of calculators to awesome prediction machine learning algorithms, it is undeniable that computers are pushing humanity's development rhythm to another level.

Nowadays, challenging as it might be, we are training computers to see using cameras as its eyes. The field called computer vision seeks to develop algorithms with the goal to enable computers to understand pictures and videos and extract useful information from it [11]. But give the sense of sight to a computer is not an easy task, the process to make it happen can be really tough due to complex dependencies among the objects in the scenes, that is why computer vision researchers often focuses on one specific aspect of the problem, such as image segmentation, object detection, or object tracking [11].

The ITS, Intelligent Transportation System, which was conceived to promote studies, projects and debates concerning the significant increase in traffic accidents, congestion and pollution [54][72], manifested considerable interest in computer vision and countless applications and studies showed up correlating these areas, such as pedestrians or cars detectors, automatic license plate recognition, traffic light detection and recognition, autonomous vehicles, and so on [84][65][74][30][33][39].

This work investigates a machine's performance when detecting and counting vehicles during the night phase of the day with the goal to study and develop a robust application based on it. We are interested in designing a robust low-cost car detector and counter with the aptitude to detect cars when no sunlight is available in order to better analyze traffic situations in streets and highways.

1.1 Project Goals

- Develop a robust application for detecting and count vehicles during night time.

- Investigate the viability of using a computational system with a camera as its eyes for this kind of situation.

- Inspect the reliability of databases provided by surveillance cameras

1.2 Study Development

To accomplish our goals, first we trained a machine learning model based in Yolov4 and as our dataset we made use of the NVD database from the Polytechnic University of Madrid [70], a database of Nighttime Vehicle Detection built with images from five different areas (California, Connecticut, Gelderland, Richmond and Walnut).

After our detector was fully working, we resorted to the DeepSORT technique to track the detected objects and implemented a virtual line technique in our video to count every car that trespasses in whatever direction they are moving forward. The figure 1.1 demonstrates well a brief picture of our project where each square will be explained in detail in the following chapters of this dissertation.

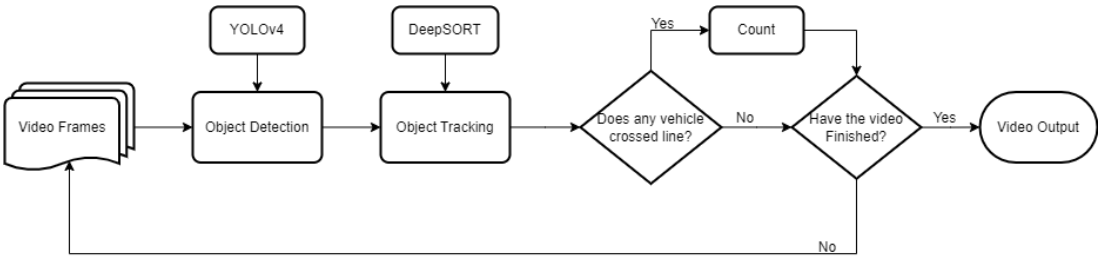


Figure 1.1: Sketch of the Project's Process
Source: Author

These workflow steps are similar to the counting system implemented in the previous study [48], but with different techniques for tracking and counting. We upgraded the previous tracking system to use the DeepSORT technique, and we upgraded the virtual line procedure to count both directions of the street simultaneously.

Alongside the counting system, we tested our systems performance on videos provided by surveillance cameras since NVD dataset is made after shots taken from these equipment. At the end of this project, we explained our results and concluded our study.

1.3 Dissertation Outline

This project is organized into five self-contained chapters. Each chapter addresses some key point of our project.

First, chapter 2 shows a friendly background about several elements correlated to the main goal of this project intending to give a brief introduction for the field that we are going to embrace for a better understanding of the following chapters. Later, chapter 3 describes every step taken regarding object detection and the results of this process. Chapter 4 continues the project with the object tracking and counting full process of this project and chapter 5 offers some thoughts and possible future approaches for this scenario.

Chapter 2

Background

In this chapter we briefly explain important terms in the computer vision literature for a better understanding of this project's procedures realized in this thesis. Therefore, our goal is to provide the essential background for a good understanding of the steps taken to accomplish our results. From Neural Networks until specific object counting technique as well as an introduction to cloud computing features that helped the development of this work are brought up in this chapter.

2.1 Artificial Neural Networks

An artificial neural network (ANN), figure 2.1, also known as a neural network, is a concept that aims to perform the human brain's behavior through artificial neurons, enabling them to process complex information and recognize patterns [45]. It contains interconnected nodes or neurons in a layered structure to process input data into a desired output, each neuron takes input from other neurons in the previous layer, and then produces an output that is sent to other neurons in the next layer. The weights on the connections between neurons determine how much influence one neuron has on another [78].

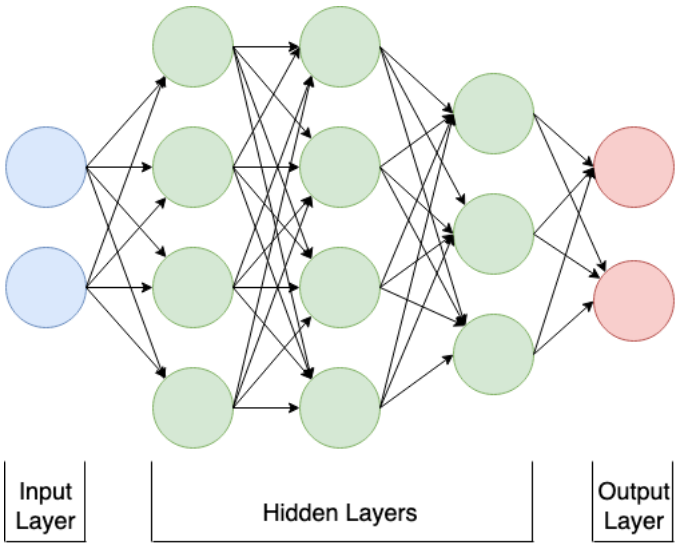


Figure 2.1: Artificial Neural Network Structure.
Source: Author

During the training of a neural network, every neuron, figure 2.2, computes the sum of all its input weights and adds the result to its own bias. The result, held by the node, is then sent to the next layer of the network [67]. This procedure helps computers to make intelligent decisions with limited human assistance, since it creates an adaptive system that learns from its mistakes and improves continuously during the training process.

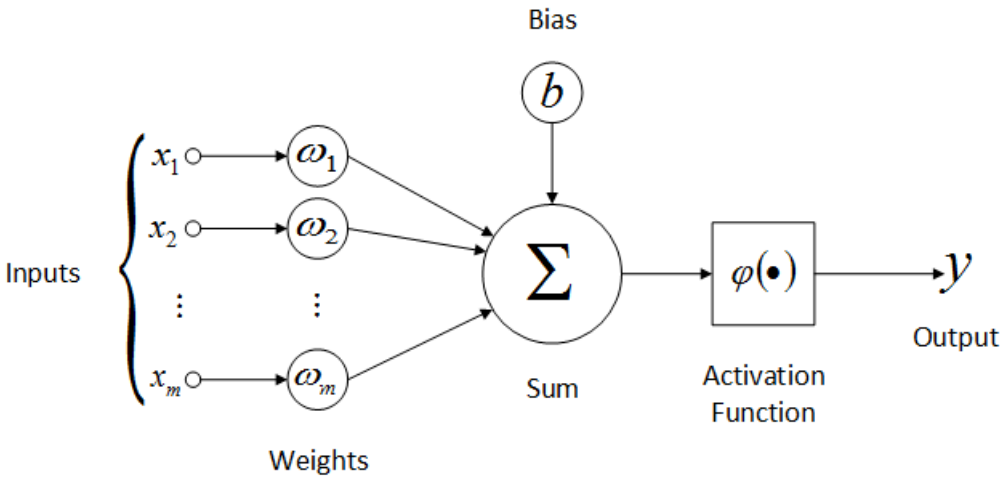


Figure 2.2: Artificial Neuron Anatomy.
Source: [21]

The training process of an ANN begins feeding it a large amount of data with specific classes labeled in a process called supervised learning, where the system learns from the ground truth provided by a supervisor [58]. With the right amount, the network will slowly build knowledge from the data given and starts to guess the right answer for problems that it has never processed before and tune the weights of each node, then it decides the value it will send to the next layer based on the results it obtained.

In 1957, the first trainable neural network was designed by Frank Rosenblatt. It was called the Perceptron and was modeled to demonstrate how the human brain processes visual data [2]. It had only one layer between the input and output layers with adjustable weights and thresholds. Nowadays, technology has advanced significantly, succeeding in numerous other applications with great accuracy. Some of the most famous applications include pattern recognition problems, like handwriting recognition for check processing, speech-to-text transcription, data analysis, weather prediction, finance, and medical diagnosis [57].

2.2 Convolutional Neural Networks

Convolutional Neural Networks are a class of deep neural networks that have been very effective in areas of image processing and analysis. Nowadays most modern object detection algorithms are built with this technology due to its lower pre-processing when compared with other detection algorithms [44].

It works taking in an input image and assigning importance to several aspects in the image in order to differentiate one from the other, with enough training, these

so called ConvNets have the ability to learn these object characteristics and have been showing to be very powerful, achieving higher accuracy and processing abilities when being used in several parts of the industry. In other words, with enough data, the CNN can be trained to understand the complexity of the image [34].

Concerning effectiveness while collecting data from images, CNNs take advantage of two properties of images through the application of relevant filters, spatial locality and translation invariance, which means that it can detect a group of neighboring pixels showing it's connectivity and showing this pattern wherever it appears in the image [14].

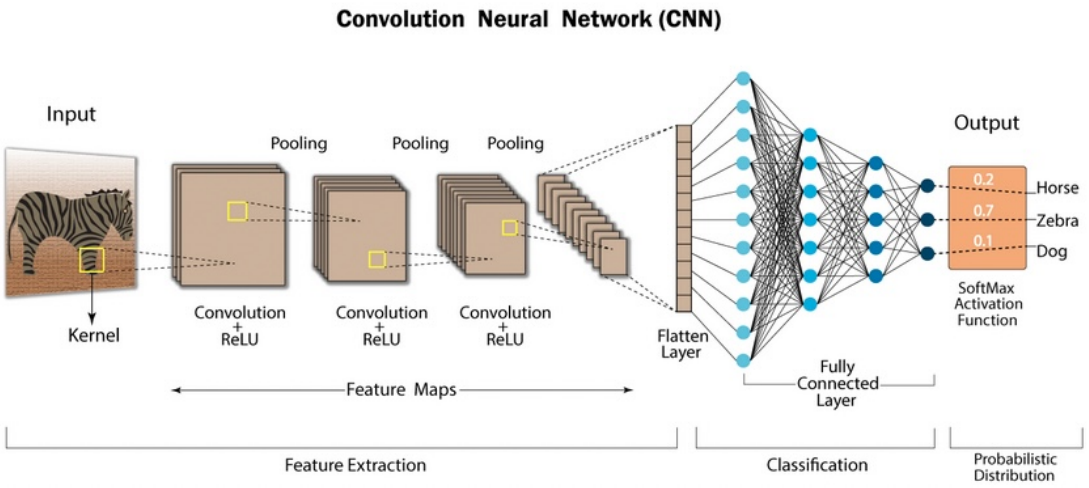


Figure 2.3: Convolutional Neural Network Architecture.
Source: [81]

The architecture of a ConvNet receive an input data and transforms it through a serie of convolutional and pooling layers. After this phase, it feeds the result into a fully connected layer and in the end it gives an output list with values for each class of it, figure 2.3 shows the details of the process. The fully connected layer is made of a bunch of hidden layers, and each one of them is made up of a set of neurons, where each one is fully connected to all neurons in the previous and following layers. There are two phases inside a CNN, the processing phase composed of convolutional layers, ReLU layers and pooling layers, and the classification phase, where the output from the previous phase is flattened and used as the input for a fully-connected layers that processes it through its hidden layers, and classify the result at the end [34].

2.2.1 Convolutional Layer

As its name proposes, Convolutional layers are the core blocks of a ConvNet and it is where most of the computation complexity occurs. It extracts features from the image by employing filters that move through the image's receptive regions. ConvNets are not limited to only one convolutional layer though, the first one is responsible for catching the low-level features such as edges and colors, the other ones adapts to check high-level features as well, giving the network more understanding about the images in the dataset. The final output of a convolution process from the input image and the filter is known as a feature map or activation map [44].

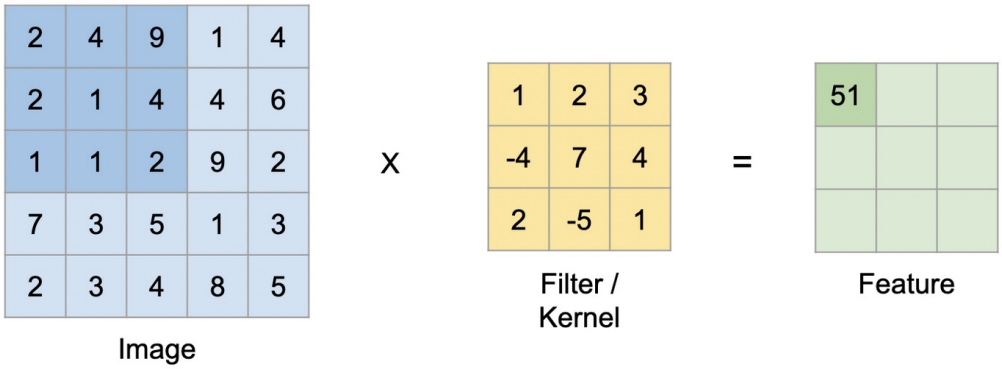


Figure 2.4: Convolutional Layer Process
Source: [50]

This operation works as shown in figure 2.4, as the kernel shifts around the image with a certain stride value, the convolution between the kernel and the image values where it overlaps occurs, performing a matrix multiplication operation between them [34]. Since RGB images have multiple channels, the kernel will have the same depth as the input image and the result of each depth multiplication will be summed as well as a bias and give as result a one-depth output as displayed in figure 2.5.

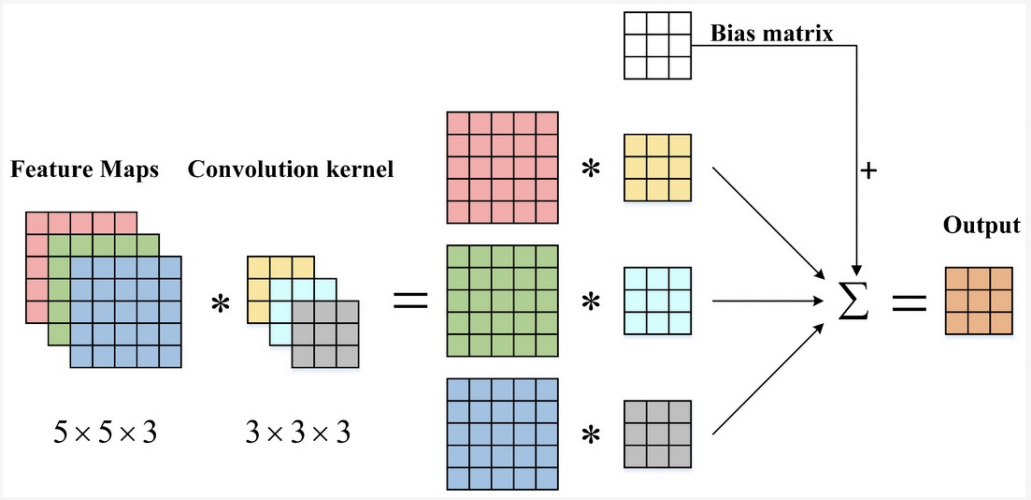
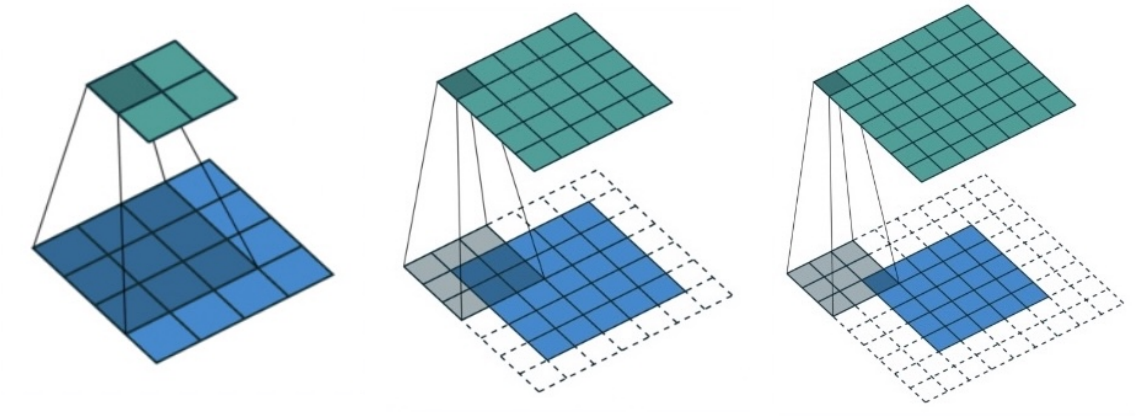


Figure 2.5: Convolutional Procedure for RGB Images.
Source: [59]

When it comes to the convolution layer, there are a few parameters which can affect the size of the output and are set before the training of the convolutional neural network [44]:

- The **Stride** defines the number of pixels that the kernel will move after each iteration with the input matrix. It is uncommon to raise its default value since it can lead to loss of information, even though it reduces overlapping and gives a smaller feature map.
- The **number of filters**, which affects the depth of the output, since each filter will originate one feature map, e.g. three distinct filters (as if applied for a RGB image) will create a depth of three feature maps.

- The **Padding**'s function is input zeros around the image's border, which means that it allow us to control the spatial size of the output that otherwise would be shrunk according with the kernel and the stride of the convolution operation, there are three types of padding:



Valid Padding | Same padding | Full padding

Figure 2.6: Operation example of distinct kinds of padding.
Source: [17]

  - **Valid Padding**: When the operation is performed without padding. In this case, the output matches the size of the kernel, causing reduction in the output dimensions if the kernel size does not correspond to the input size.
  - **Same padding**: This kind of padding ensures that the output matrix will match the size of the input image.
  - **Full padding**: This padding ensures that the kernel passes through every pixel for the same number of times, producing an output exceeding its input size.

2.2.2 ReLU Activation

The Rectified Linear Unit activation function, nowadays, is the most commonly used activation function in deep learning models. It is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input [20].

The function works by returning zero if it receives any negative input, and returning the input value back otherwise. It is mathematically written as $f(x) = max(0, x)$ and graphically looks like the example in figure 2.7.

In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed and ReLU become widely used as it is easier to train, often achieves better performance and alongside with that, it also has computational simplicity since it only requires a max() function to implement [38].

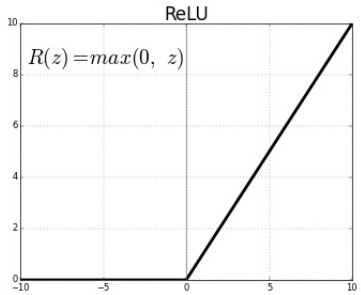


Figure 2.7: Diagram of the ReLU Function.
Source: [50]

2.2.3 Pooling Layer

Another common layer applied when it comes to convolutional neural networks is the pooling layer. It usually comes between successive convolutional layers and reduces the spatial size of the image decreasing the computational power needed to process the data while extracting dominant features which are rotational and positional invariant and improving efficiency, limiting the risk of overfitting.

It works resembling the convolutional layer as the kernel slides through the input image scanning a small region and doing some kind of operation, but instead of having weights, the kernel goal is to aggregate the values of the receptive window and fill the output matrix with the results. It requires two parameters in order to work, the stride and the size of the window [34].

There are two main types of pooling operation, the **Max pooling** which selects the larger value from the receptive window and includes it in the output and the **Average pooling** where it calculates the average value from the receptive window and fill the output with the result. The figure 2.8 shows how the process works [44].

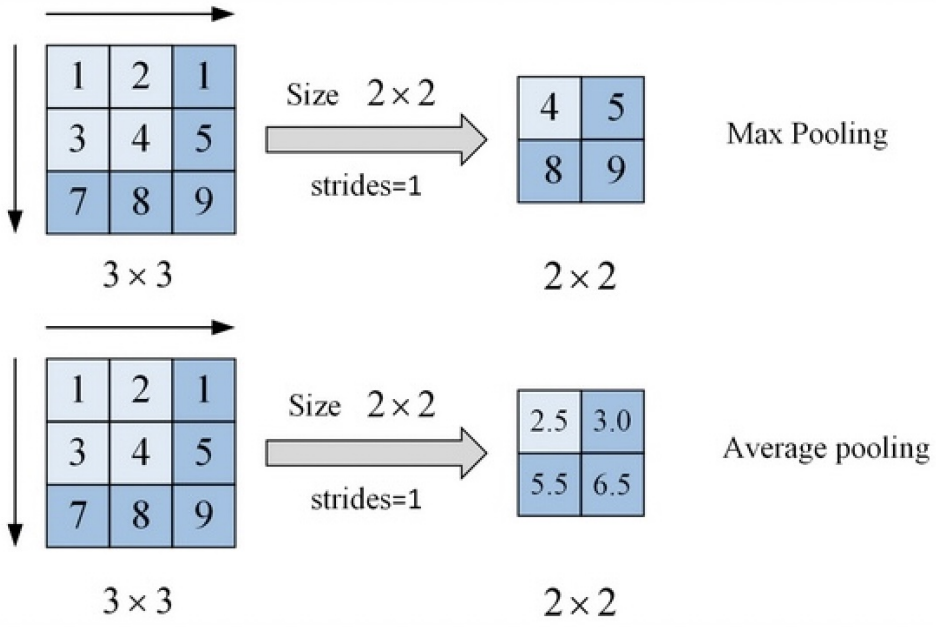


Figure 2.8: Representation of both pooling options performance.
Source: [59]

Max pooling is often the choice to work with CNNs because it has better results due to the fact that it also works as a noise suppressant, while average pooling only executes dimensionality reduction. Hence, the most common setup for pooling layers applied for CNNs is a max pooling filter of 2x2 size applied with a stride of 2, it operates independently on every depth of the input and usually does not use padding in the input before walks through it [34].

2.2.4 Flatten Process

The main goal of this process, as its name implies, is to flatten the final output of the processing phase, and feed it to the classification phase composed of a regular Artificial Neural Network, as in figure 2.3, where it will be classified into one of the classes the network supports [51].

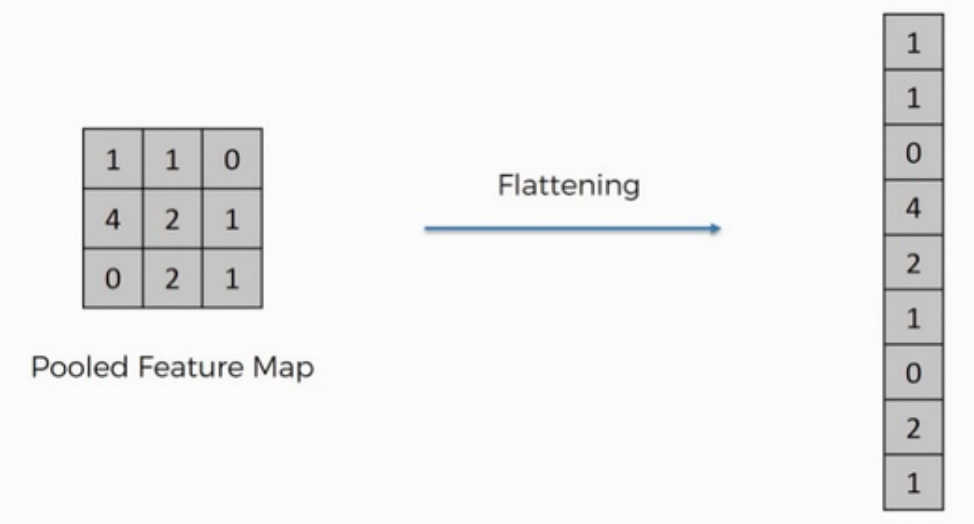


Figure 2.9: Flattening conversion process.
Source: [36]

Moreover, it basically grabs the results of the matrix row by row, and fills them into one long vector as shown in figure 2.9.

2.2.5 Fully Connected Layer

As the final layer of a convolutional neural network, the fully-connected layer, for short FC Layer, is the layer that combine the features extracted by the convolutional layers, and along with the softmax layer, it make a prediction about the input data into one of the classes provided for the CNN [44].

The FC Layer is composed for a bunch of hidden layers where each node in the previous hidden layer connects directly to all nodes in the following layer, figure 2.10. The training process involves forward and backward propagation leading to many iterations and epochs.

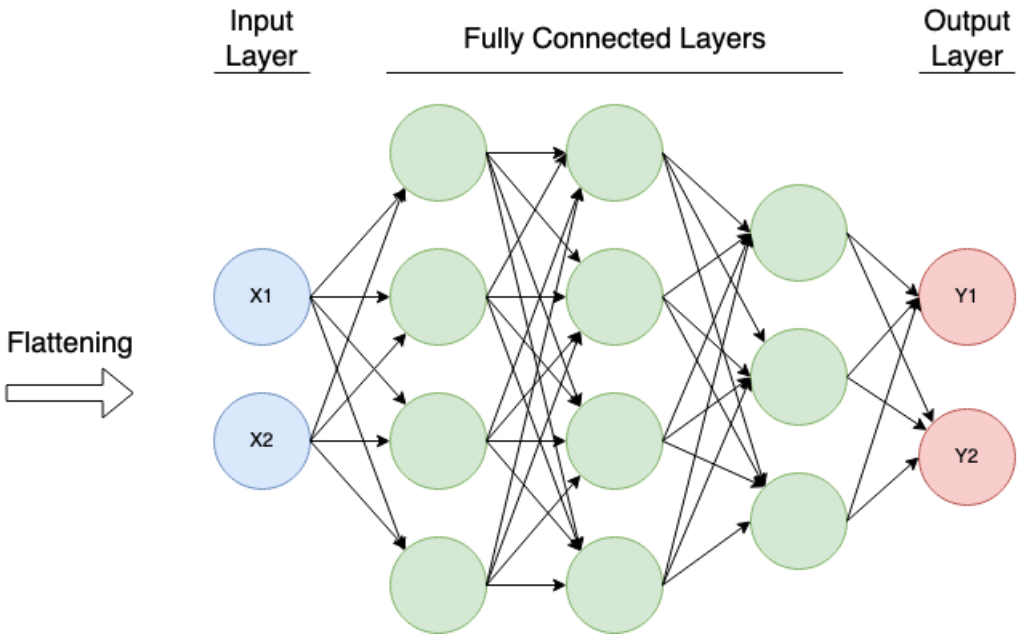


Figure 2.10: FC Layer demonstration.
Source: Author

During the training process, the model verifies the accuracy of predictions and uses a cross-entropy function to enhance network performance leading to better generalization models. With this process, the network is able to learn non-linear combinations of the high-level features of the objects and might learn a possibly non-linear function for that space as well.

After a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and using the Softmax activation function, classify inputs appropriately, producing a probability from 0 to 1 [34].

2.2.6 Softmax Activation

Softmax activation is another kind of function widely used in neural network models, however, differently from the ReLU function, it is usually used as the activation function in the output layer of neural network models that predict a multi-class probability distribution [53].

It operates taking in the output vector of the neural network and returns a vector of probability scores with the same size as the input, where the probabilities of each class are proportional to the relative value in the input vector, i.e it normalize the CNNs' output and gives each value of it a probability score in a range between 0 and 1. This allows very large values given as the weighted sum of the input to be output as 1.0 and very small or negative values to be mapped to 0.0 [43].

It is mathematically written as the equation 2.1 [53] and graphically looks like the example in figure 2.11.

$$s(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_i}} \tag{2.1}$$

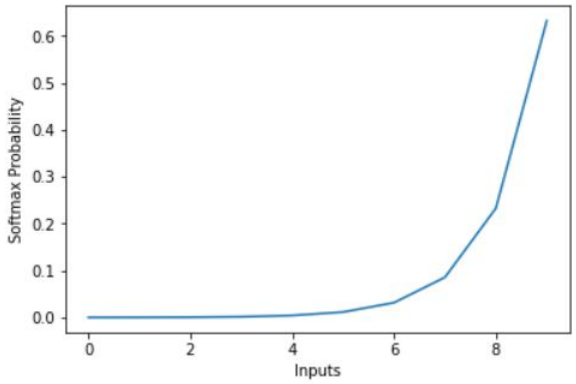


Figure 2.11: Diagram for the softmax activation function.
Source: [24]

2.3 Object Detection

As one of the most classical Computer Vision techniques, Object Detection is a technique that combines both Image Classification and Object Location. i.e. it identifies, locates and classifies objects accurately in an image or video labeling them inside a bounding box and performing a bunch of different tasks [68]. It is one of the fundamental problems of computer vision and is the basis of many other techniques, as it solves two main tasks, to find an arbitrary number of objects and to classify every single object and estimate its size with a bounding box [29].

Before the rise of deep learning methods, the named classical object detection models were mainly sliced into three stages [37]:

- Informative Region Selection: The step where the scan of the whole image occurs through a multi-scaling sliding window since different objects may appear in any positions of the image and have different sizes.
- Feature extraction: The step to extract visual features that can provide a semantic and robust representation.
- Classification: The last step, the classifier distinguishes each object from all the other categories and makes the representations informative for visual recognition.

Since the upgrade of computer vision with deep learning techniques, the so-called modern computer vision has evolved and over the past 8 years many new object detection models have been created. Those state-of-the-art methods can be sorted into two groups: two-stage detectors and one-stage detectors [76].

The main difference between these two groups is that while the two-stage detectors first proposes approximate object regions to after use this features for the classification as well as bounding box regression the one-stage detectors combines both tasks into one step, achieving higher performance at the cost of accuracy [76]. In Figure 2.12 is presented some of the most famous Object Detection Models through time they were released.

In this thesis we will focus on a one-stage model from the YOLO family. The main advantage of this kind of model is that those algorithms are faster and structurally simpler than the two-stage detectors, even though they are not as good at

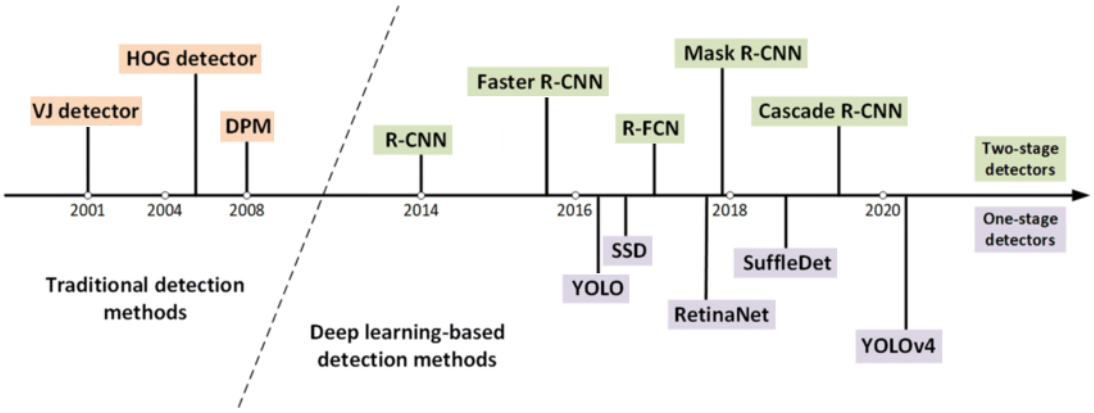


Figure 2.12: Milestone of Object Detection Models
Source: [88]

recognizing irregularly shaped objects or a group of small objects. The detector was chosen because of its balance between performance and accuracy provided and it will handle all steps regarding object detection in this project [48].

2.3.1 You Only Look Once (YOLO)

Unlike other approaches based in R-CNN from the time it was released which use region proposal methods to first generate potential bounding boxes and after run a classifier, YOLO (You Only Look Once) relabeled object detection as a single regression problem. It accelerates the task of predicting and shows what objects are present and where they are only looking once at the image[15].

Being a single convolutional network that simultaneously predicts multiple bounding boxes and class probabilities, YOLO sees the entire image during training and test steps in order to directly optimize its detection performance, unlike sliding window and region proposal-based techniques, consequently it implicitly encodes contextual information about classes as well as their appearance [15].

As said in its paper and is presented in the Figure 2.13, the model has 24 convolutional layers followed by 2 fully connected layers. It has alternating 1 X 1 convolutional layers to reduce the features space followed by 3 X 3 convolutional layers for feature extraction.

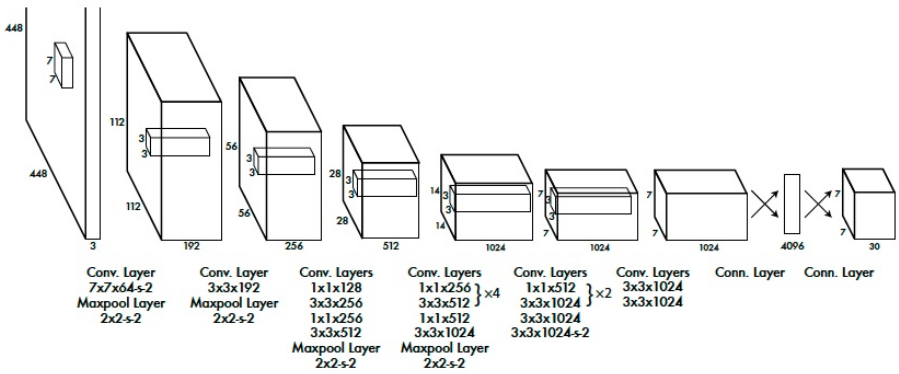


Figure 2.13: YOLO layers.
Source: [15]

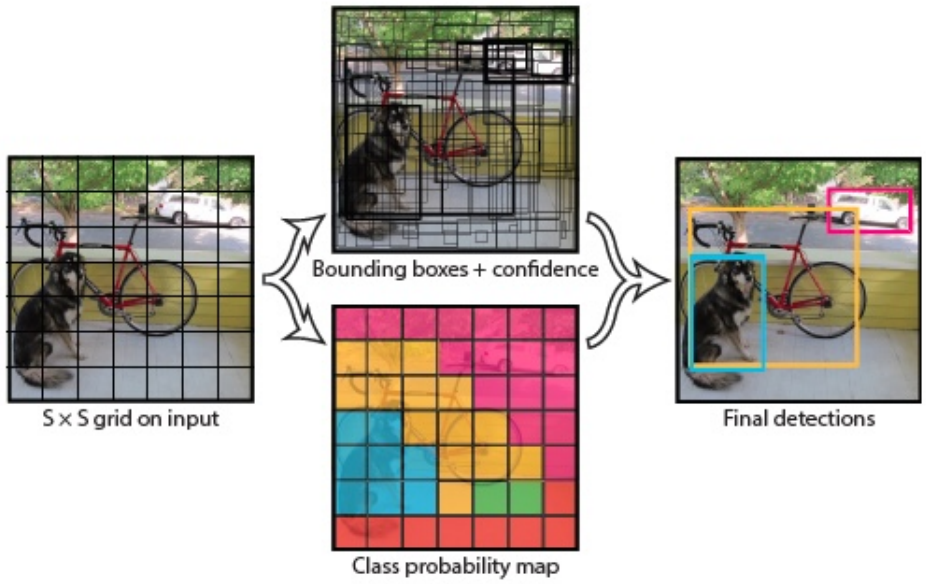


Figure 2.14: YOLO model detection workflow.
Source: [15]

The system operates dividing the image into an S X S grid, each cell will become responsible for detecting an object as well as predict bounding boxes and confidence scores for those boxes, these scores encode both the probability of that class appearing in the box and how well the predicted box fits the object. The figure 2.14 presents the Model's workflow.

2.3.2 YOLOv4

The fourth version of the YOLO family, YOLOv4 performs significantly better than its predecessors both in performance and speed as shown in figure 2.15. It was designed to aim for a fast operating speed neural network and optimize it for parallel computations [42].

To accomplish that, the team behind it chose a few different options for each of the three parts that compose the object detector (Backbone, Neck and Head) and shortlisted these candidates until they found the best choice as observed in figure 2.16.

First, the backbone, used for feature extraction, chosen was the CSPDarknet53, since it had the higher number of parameters and speed between the options as well as a good receptive field between the options [42].

Following, the neck which extract rich features that are used for accurate predictions, the most important feature is the receptive field and the chosen one was Spatial Pyramid Pooling that helped the backbone increase the receptive field and Modified Path Aggregation Networks (PANet path-aggregation) for pyramidal structure.

For the Head, the detector itself, they used the latest YOLO version so far, the YOLOv3 for loss calculations and predictions during the time of inference [55]. To sum up, the YOLOv4 consists of:

- Backbone: CSPDarknet53;
- Neck: Spatial Pyramid Pooling additional module, PANet path-aggregation;

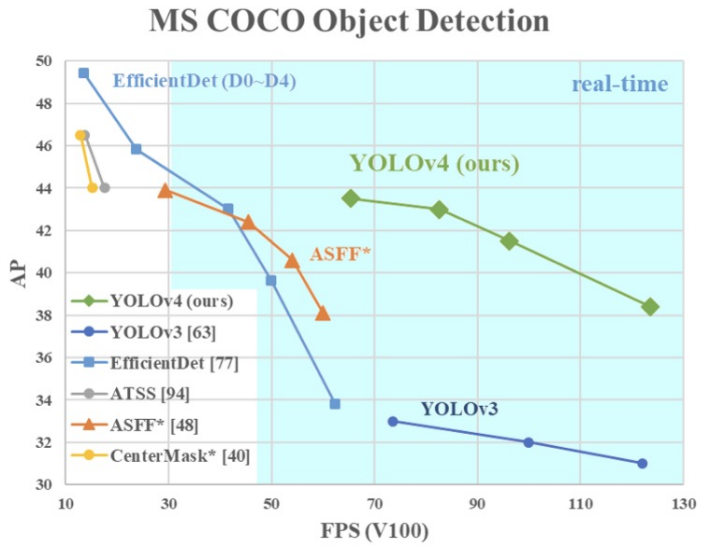


Figure 2.15: YOLOv4 performance comparison.
Source: [42]

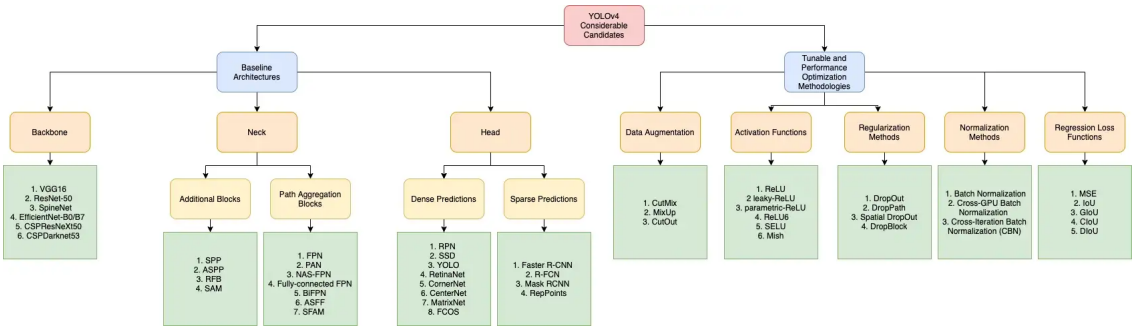


Figure 2.16: Features considered in the study for the creation of YOLOv4.
Source: [55]

- Head: YOLOv3

That alone already increases the performance of YOLOv4 when compared with its predecessor, but there is more to it. Another key point for the YOLOv4's success was the features they add to the architecture in order to optimize the process. These features can be sorted into two groups, the Bag of Freebies that improves the model's training step, with features like data augmentation, cost function and class imbalance that increase the accuracy of the model without any impact on the inference cost, and the Bag of Specials that lightly increases the inference speed, but can drastically improve the accuracy of the object detector, these plugins enhance certain attributes in the model such as increasing the receptive field size, strengthening feature integration capability and others. The addition of these features to YOLOv4 (shown in Figure 2.17) lead it to an efficiency enhanced by at least 12% when compared with older versions [55].

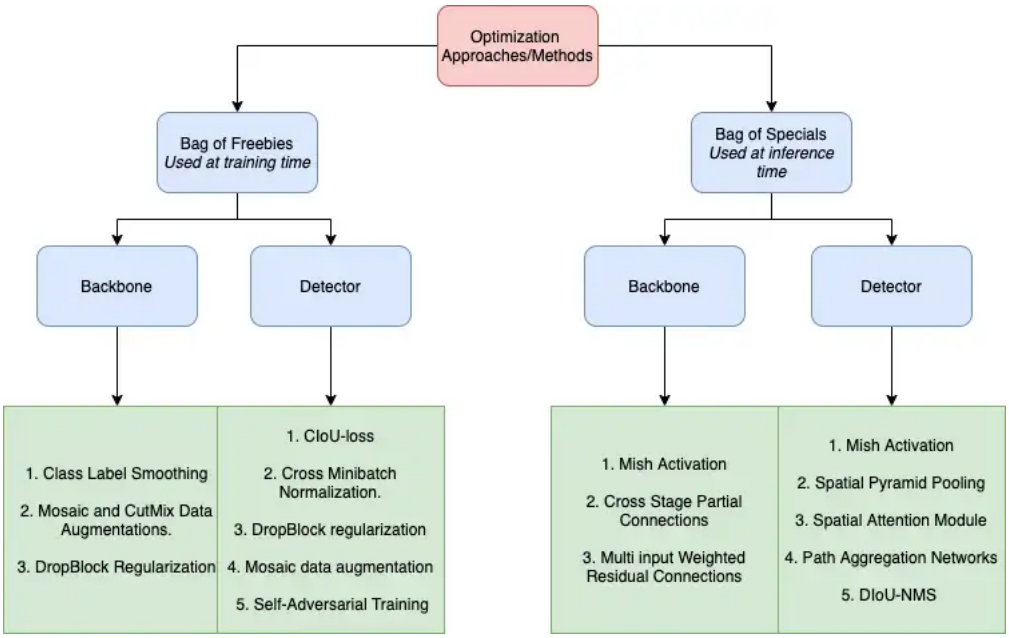


Figure 2.17: Bag of freebies and Bag of special components present in YOLOv4. Source: [55]

2.4 Object Tracking

Object tracking is a powerful computer vision technique that plays an essential role in video understanding. It is a way to follow each object detected as they move around in the frames of a video creating a unique identification for each one. It performs detection per frame and formulates the object tracking as a data association task. The figure 2.18 demonstrates how the technique works.

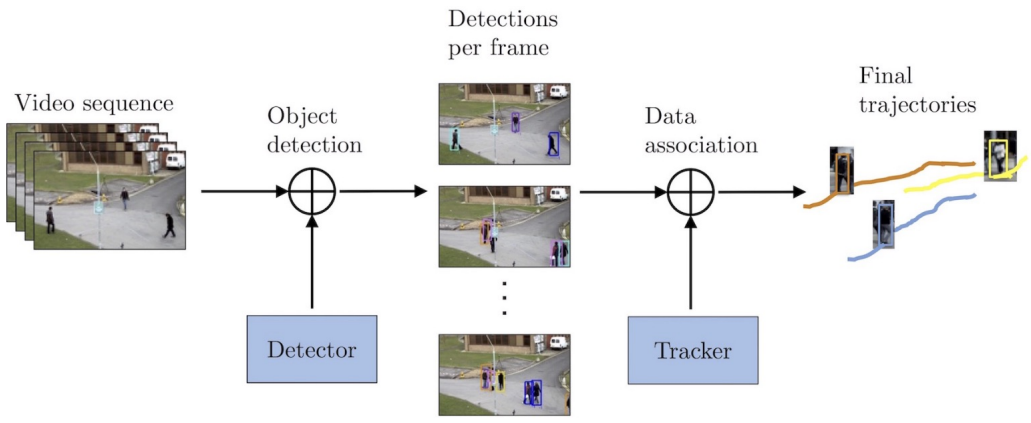


Figure 2.18: Object tracking Sample. Source: [13]

Because of the high-performance of object detection models, methods of tracking-by-detection have gained favor due to their excellent performance despite these methods usually requiring computationally expensive components, such as the detector itself and an embedding model [79].

Since it can process real-time footage, object tracking is widely used in a bunch of

applications such as traffic monitoring, self-driving cars, surveillance, human activity recognition and security and can perform at two different levels [75]:

- Single Object Tracking (SOT), which aims to track an object of a single class instead of multiple objects. Someone first has to manually provide the first bounding box to the tracker and the tracker should be able to track whatever object they are given, even if the object has no available classification.
- Multiple Object Tracking (MOT), that refers to the approach where the tracking algorithm tracks every single object in the video. First, the tracking algorithm determines the number of objects in each frame and then it keeps track of each object's identity from frame to frame until they leave the monitored space.

While implementing an object tracking algorithm, there are some obstacles that can decrease the models performance. The occlusion of objects in the image is one of the most common ones, but background distractions also affect the accuracy of the tracker as well as the variance of shape and size of the objects detected. Another challenge usually faced when implementing an object tracker, is the training time that takes a lot of time to train since the algorithm is mathematically complex and the enhancement of the tracking speed in order to better fit real-time models [85].

Equally to Object Detection, there are several methods and ideas regarding object tracking developed in the past 20 years in order to improve the accuracy and performance of the models, some using classical machine learning like K-Nearest Neighbors and some with Deep Learning such as the OpenCV Object Tracking built-in methods like MIL, KCF and GOTURN [85].

In this thesis we will work with the DeepSORT method since it is an upgrade of a popular object tracking algorithm called Simple Online Real-time Tracker (SORT). In these following subsections we will review both versions for a better understanding of this method.

2.4.1 Simple Online Real-time Tracker (SORT)

The SORT approach is composed of three steps. It first runs a detector, the default is the Faster Region CNN, but it can be swapped to others architectures to improve the detection performance as well, after the detection process, the method estimates the target's object future location making use of the Kalman Filter framework, this prediction will posteriorly be corrected with the bounding box values of the next detection. In case of occlusions the next location will be predicted without the correction of its previous state, that way the method can avoid some occlusion issues. After the prediction, the approach runs its data association, assigning detections to existing targets using the intersection-over-union distance between each detection and all predicted bounding boxes from the existing targets. This is solved optimally using the Hungarian algorithm [16].

2.4.2 DeepSORT Tracker

The SORT method accomplishes good performance in terms of tracking precision and accuracy, but it returns a high number of ID switches as well as a deficiency in tracking through different viewpoints despite the effectiveness of Kalman filter.

To overcome these issues, the Appearance feature was added to the method adding deep learning techniques to the tool. The Appearance feature allows the DeepSORT to describe the features in the image allowing it to estimate object's location through longer periods of occlusions. It works using nearest neighbor queries in the visual appearance to establish the measurement-to-track association, this association determines the relation between a measurement and an existing track using the Mahalanobis distance technique. These improvements allow DeepSORT to handle difficult scenarios and reduces the number of ID switches by over 40% [26]. The procedure of the DeepSORT technique is shown in detail in the Figure 2.19.

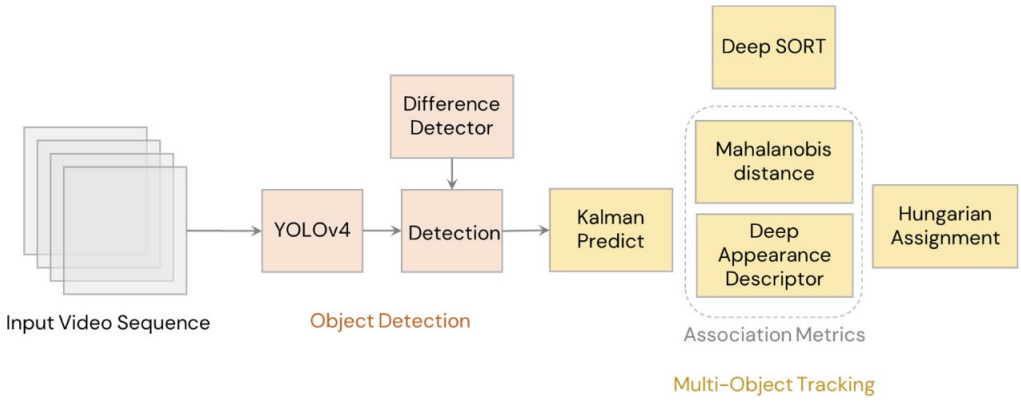


Figure 2.19: DeepSORT technique workflow.
Source: [66]

2.5 Object Counting

Counting objects has always been an important task for our society and has several fields that people make use of this tool. There is no need to say it has become an extremely important mechanism for computer vision as well.

This task involves counting the number of objects in an image or video and accomplish this goal can be a quite challenging problem since it requires not only identifying the presence and location of objects, but also keeping track and count them as they appear and disappear from the scene [64].

The applications of this technique can be applied from inventory management to traffic monitoring and upgrading this technique has become an important matter in academia since it can help in the automation of production workflows, reduction of human errors and prevent business interruptions [82]. There are two main kinds of Object Counting [63]:

- Detection-based Object Counting: Where we use a detector to identify the target objects in the frame and count how many there are. It usually uses a state-of-the-art object detection method like Faster RCNN or YOLO and returns the count of the detected objects in the image.

- Regression-based Object Counting: Where it is a full built regression method using CNNs. This method usually takes the input image and directly estimates the object count using features extracted from the ground truth of the dataset, providing the count result at the conclusion of the process.

Imaginary Line Counting Method

The Virtual Line is an approach for object counting. It is a line drawn across the frame and counts every detected object that passes through it. The position and orientation of the line are custom settled in order to better fit the project, letting it perpendicular to the objects it is going to count [8]. The figure 2.20 is composed of two images that show a glimpse of how this method works.

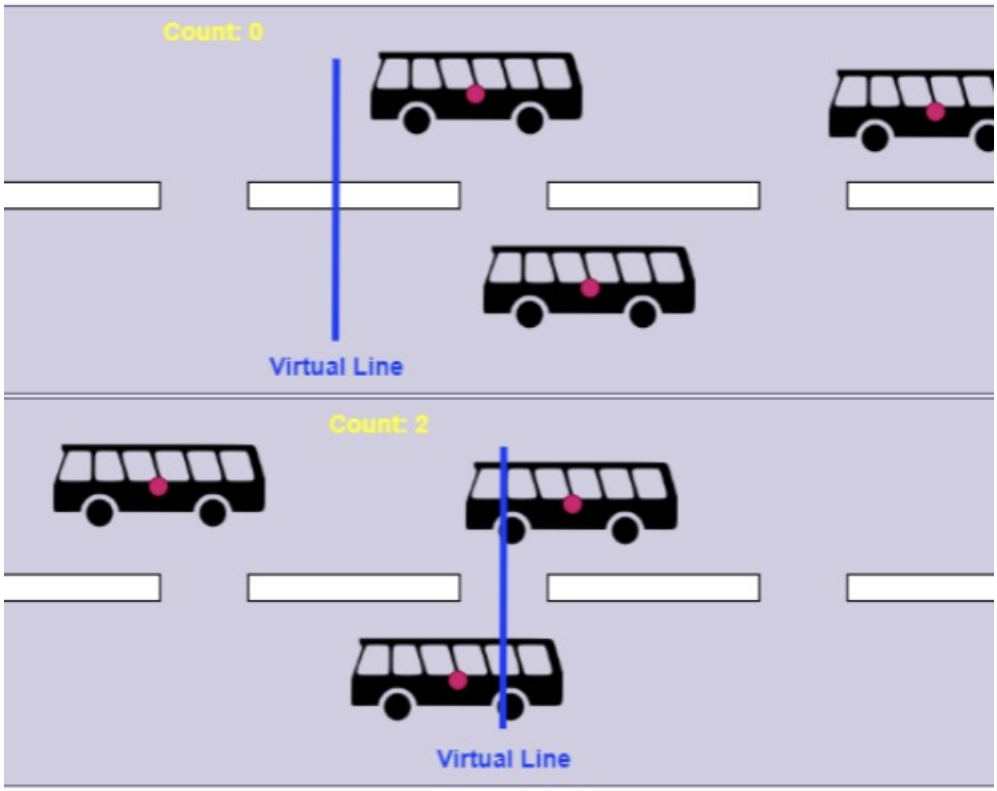


Figure 2.20: Virtual line counting approach example.
Source: Author

It works creating a two-points line with the object's centroid of both current and previous frame, then it will check if this centroid's line created intersects with any segment of the counting line in that frame. If a segment of the line is cut by an object, it will be counted by the program.

The virtual line counting can be used in a variety of applications, such as counting the number of vehicles on a road or the number of people in a crowd and it is highly recommended for its great performance yet simple design [83].

2.6 Metrics

As we train any machine learning model, we will desire to evaluate its performance, these evaluations will help the person in charge to understand how well the model has performed for the given data and improve the model's performance by tuning the hyper-parameters of the system. In this section, will be presented some metrics usually used to measure object detection and spatial localization performance.

Confusion Matrix

Confusion Matrix is not exactly a performance metric but a tabular visualization of the model's performance that shows us the training results relating the ground-truth with the model's predictions to the problem [90]. A priori, it is a matrix where each row represents the predicted values and each column shows the actual values of a class, figure 2.21, and it is often used as a basis on which other metrics evaluate these results [73].

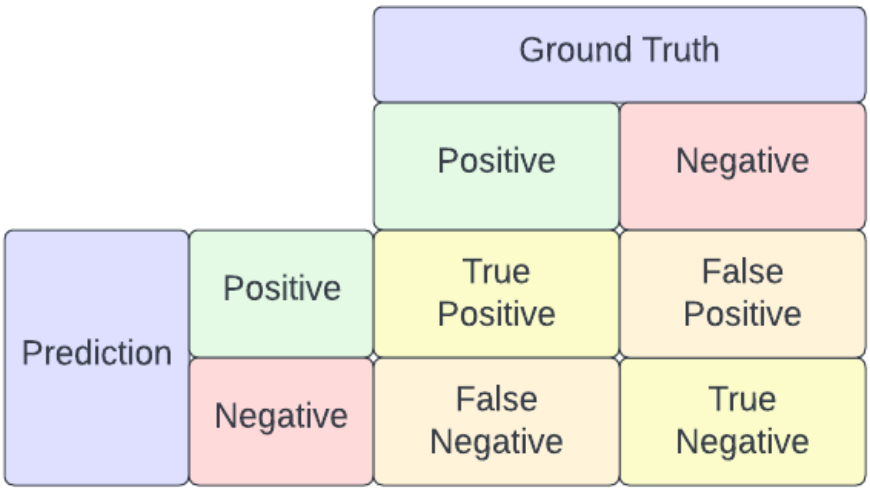


Figure 2.21: Example of the confusion matrix.
Source: Author

In figure 2.21, each cell in the matrix correspond to a different terminology, which meaning are [31]:

- True Positive (TP): Represents how many cases were predicted as true and in fact were true values.
- True Negative (TN): Stands for how many cases were predicted as false as well as its actual value.
- False Positive (FP): States how many cases were predicted as true but in reality they were false values.
- False Negative (FN): Indicates how many cases were predicted as false however they were true values.

Whenever modeling a different project, we have to check whether it is the harmless wrong response for the machine to guess, since a perfect machine is not possible most of the time. So, we need to tune our model to prioritize false positive or false negative responses in order to achieve better results.

E.g. We don't want visual systems to detect cancer cells as nevus since this disease gets worse as time passes, otherwise is acceptable though as this result would lead to other exams and would be checked if it was a mistake or not. On the

other hand, talking about criminal courts, if there aren't enough proofs, it is better to let a criminal free than convict an innocent person to the jail [91].

Accuracy

The accuracy is one of the main and most common classification metrics available to measure a model's performance. It calculates the proportion of the predictions that were in fact correct and is demonstrated in the equation 2.2 [35].

$$Accuracy = \frac{CorrectPredictions}{TotalPredictions} = \frac{(TP + TN)}{TP + FP + TN + FN} \tag{2.2}$$

Accuracy metric is very intuitive to understand as well as to implement but it is very important to take note that the accuracy metric heavily relies on data specifics, if the dataset isn't approximately balanced, the result won't be something you can trust. It works well only if there are an equal number of samples belonging to each class [90].

Precision

The precision determines the proportion of positive predictions that was actually correct. It focuses on checking how well the system performs when you can avoid False Negatives but can't ignore False Positives. For example a spam detector, it is acceptable to have a few spam letters in your inbox (False Negative) but tagging as spam an important email and moving it from folders is a problem (False Positive) [35].

$$Precision = \frac{CorrectPositivePredictions}{TotalPositivePredictions} = \frac{(TP)}{TP + FP} \tag{2.3}$$

As in equation 2.3, it can be calculated as the predictions that are actually true to the total of positive predictions. Precision usually is the best evaluation metric to deal with imbalanced data unless you need to take into account false negatives and true negatives [90].

Recall

The recall metric shows the proportion of correct positive predictions out of all positive images in the training dataset. It can be evaluated by dividing the number of true positives by the number of positive images in the training [90].

Both Recall and Precision metrics are similar, however, while precision checks how the classifier performs when you can avoid False Negatives but can't ignore False Positives, the recall metric works the other way, i.e. it focuses on checking how well the classifier perform when you can avoid False Positives but can't ignore False Negatives [35].

$$Recall = \frac{CorrectPositivePredictions}{TotalPositiveData} = \frac{(TP)}{TP + FN} \tag{2.4}$$

As expressed in equation 2.4, a good recall value means that the classifier didn't miss any true positives, it does not mean the classifier does not have any false positives though. So, if we maximize the precision metric, we will minimize the false positive errors and if we maximize the recall metric, we will minimize the false negative errors.

F1-Score

The F1 Score metric makes use of both precision and recall, it evaluates the balance between them by calculating their harmonic mean. It usually is used if both accuracy and recall are important for evaluation [40]. The equation 2.5 demonstrates how we can obtain it.

$$F1Score = 2 * \frac{precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{2.5}$$

It tells us how precise and robust our classifier is, as is a more intricate metric that allows us to get results closer to reality on imbalanced classification problems [31].

The greater the F1 Score, the better is the performance of the classifier, a low F1 Score is not very informative though, it only tells you about the performance at a threshold, but with that, you won't discover whether it is a recall error or a precision error [90].

Intersection Over Union (IoU)

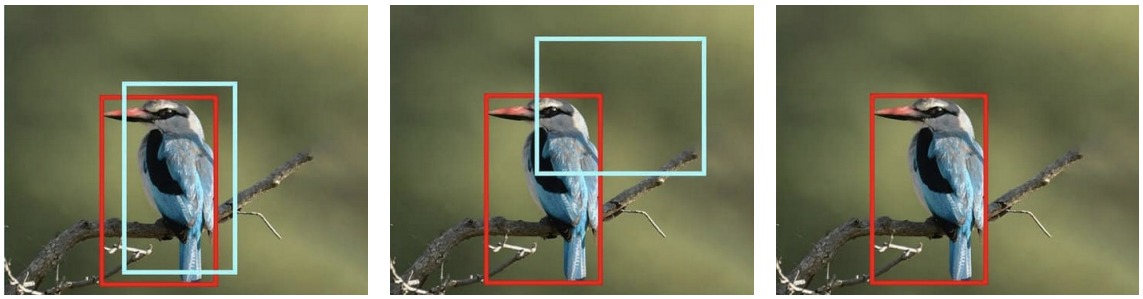
As one of most common deep learning metrics, the Intersection Over Union metric, it also can be named as Jaccard similarity coefficient in some papers, estimates how well a predicted mask matches the ground truth data, i.e. it measures the correctness of a prediction. It is calculated by dividing the overlap between the predicted and ground truth annotation by the union of these and it is represented in the equation 2.6. Its result can vary between 0 and 1 [92].

$$IoU = \frac{Prediction \cap GroundTruth}{Prediction \cup GroundTruth} \tag{2.6}$$

Even though in a real situation, it is highly unlikely that the predicted coordinates will exactly match the ground truth, we need a way to measure if our model's predictions are on spot, and this metric rewards the model whose predictions heavily overlap the ground truth [18].

A good value for this metric can change according to the precision required for the task, a standard value defined by literature for a correct detection is always higher than 0.5 though [86].

In the example of the figure 2.22 is shown a picture of a bird, where the red square signs the ground truth while the cyan square marks for the model prediction. The first image of the example presents a great example of overlap between the ground truth and the prediction, while the last image shows that even though the prediction overlaps entirely the ground truth, it does not present a good IoU since



(a) IoU = 0.96 (b) IoU = 0.22 (c) IoU = 0.00

Figure 2.22: Different examples of IoU values.
Source: [86]

union operation value is far greater than the intersection operation value of these two.

Mean Average Precision (mAP)

The Mean Average Precision (mAP), might be the most common evaluation metric to analyze the performance of object detection and segmentation systems. A lot of object detection models such as Fast R-CNN, YOLO and Mask R-CNN use this metric to check its robustness that can change from 0 to 100 while trying to strive for the best value possible [28].

For starters, the Average Precision (AP) is not the average of the precision metric, we need the results of it as well as the other metrics previously explained in this section to calculate that though.

The Average Precision is presented as the area under the precision-recall curve and is calculated for each model's class individually [56].

First, with the results of precision and recall of each object detected, we plot a table where precision values are recorded across 11 equally spaced recall values shifting from 0.0 to 1.0. Then, at each recall level, we save the higher precision value of that level and calculate as express in the equation 2.7 [27] [87]:

$$AP = \frac{1}{11} * \sum_{Recall_i} HigherPrecisionValue * (Recall_i) \tag{2.7}$$

The mAP is the average of AP, after calculating the AP of each class the average value of it will result in the mAP [89]. Equation 2.8 demonstrates this calc.

$$mAP = \frac{1}{n} * \sum_{k=1}^{k=n} AP_k \tag{2.8}$$

It is worth mentioning that mAP metric depends on IoU metric as well since it defines when a prediction is correct based on its value.

2.7 Cloud Services

As one of the most important developments in the history of the IT World [47]. Cloud Computing Services has become extremely popular among everyone

who works with computing technology. Cloud Computing allows users to host applications and files on remote servers and computers, accessible via the internet [6]. It promises improved cost-efficiencies, faster innovation, faster time-to-market, and the enhanced ability to scale applications on demand [25].

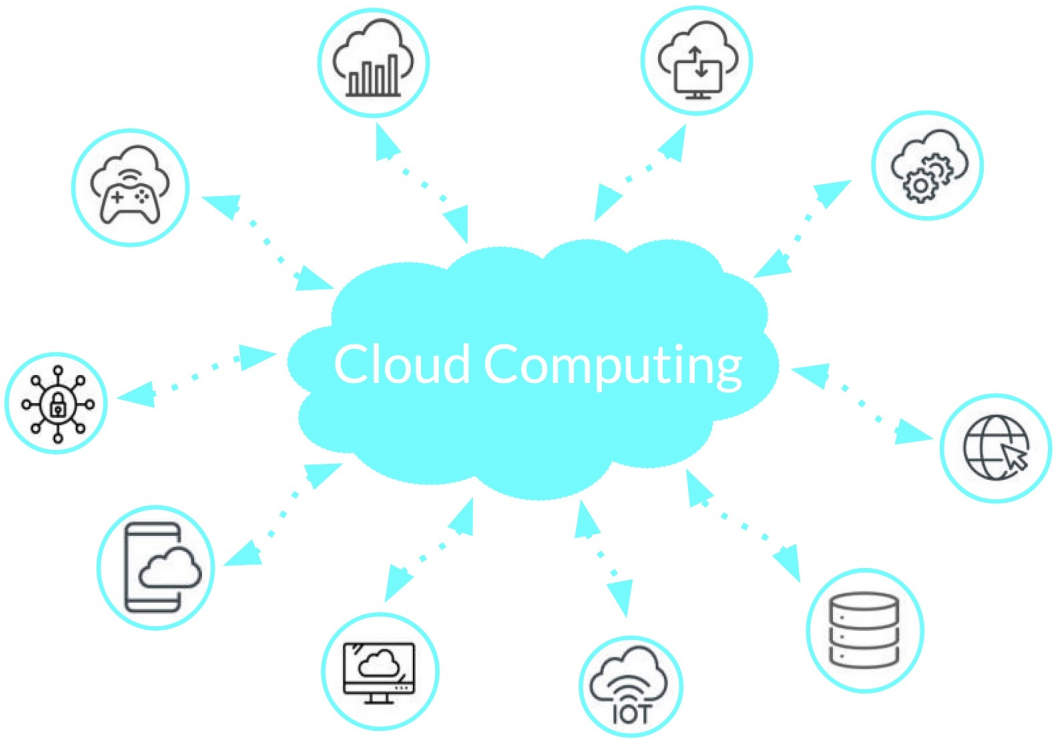


Figure 2.23: Cloud computing portrayal.
Source: Author

Numerous applications nowadays are utilizing cloud services, as shown in Figure 2.23. For this project, we will make use of two of them to accomplish our objectives, as they offer superior computational processing capabilities.

2.7.1 Google Drive

The Google Drive (2.24), is the google "Cloud-based" storage system and provides users with the ability to store, create and share files online [32]. In spite of everything it also allows users to integrate files from other google services.

With 15GB free storage available for its users, Google Drive has become a great tool for everyone who wants to store your files in a safe place. It also lets you view different file formats without buying extra software and access your files from any device [80].

The main role of Google Drive in this project will be pivotal. It will serve as the primary storage for all files, including the dataset and algorithms used throughout the project. We will also integrate our dataset with Colab to avoid the need to upload it every time we use it.

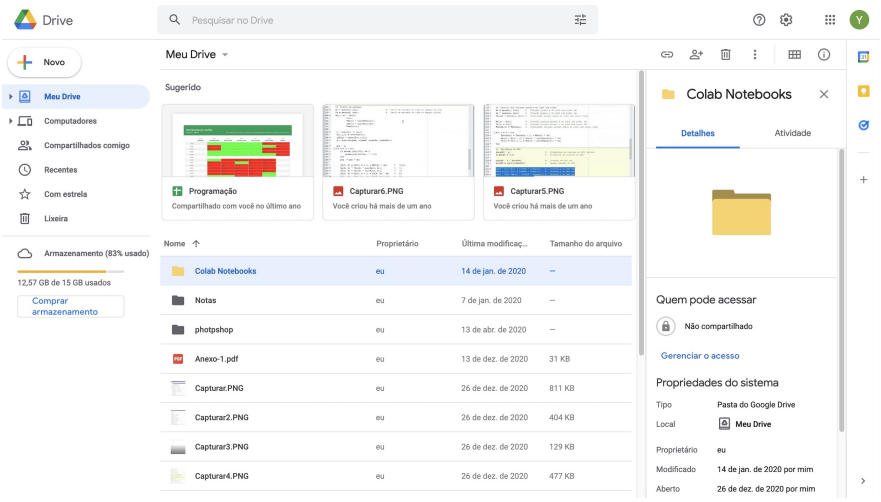


Figure 2.24: Google Drive homepage
Source: Author

2.7.2 Google Colab

Google Colab (figure 2.25), is a free Cloud Computing service which allows users to program Jupyter Notebooks and offers some advantages while performing this task. It is hosted by Google itself which makes it work extremely fast due to Google's network speed. They also allow users to Access their GPU, a single 16GB NVIDIA Tesla K80 [22], as well as their TPU, which are great for projects which require high computer processing.

Additionally, Google Colab includes many popular libraries for deep learning, machine learning, and data science, making it unnecessary to set up the environment and it integrates with Google Drive, allowing users to store notebooks online, share them with others, and avoid losing their files [52].

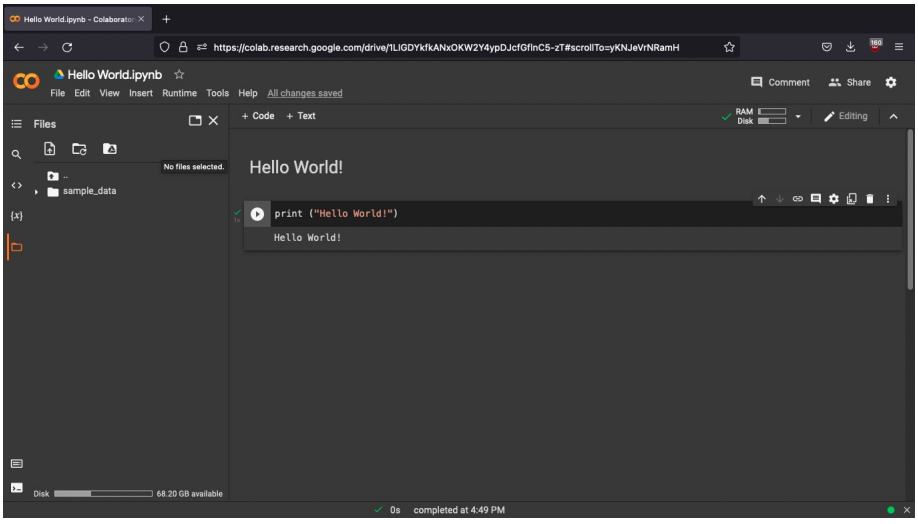


Figure 2.25: Google Colaboratory homepage.
Source: Author

This project will use Google Colab to handle all the computer processing phases, such as training the algorithm for car detection, tracking and counting, and gener-

ating results for each step.

2.8 Related Works

There is no doubt that since our community is growing, the number of cars in the streets have been increasing lately [60][61][77] but counterbalance to that, the number of traffic jams and accidents are escalating as well [4][69][41] and these complications have direct impact in the economy of the country [62]. As stated [7] we can distinguish two ways that traffic jams affect the economy. The first is due to the lost time a person spent on a roadblock that could be used in a productive activity and the second is the delay of goods and the increase of fuel and pollution that are caused by this disorder.

Added to that, traffic jams have significant influence on people's health as well [10][9], many doctors affirms that roadblocks cause chronic stress to the drivers, who might feel scared, insecure, helplessness and anxious. This stress provokes the adrenaline in the body, speeding up the heartbeats, raising the blood pressure and sometimes elevating even the sugar rate in blood. Congestion also increases chances of car accidents as well, according to data [46], pedestrian fatalities can raise up to 8.5 times in these circumstances.

Therefore, counting vehicles on a daily-basis aiming to learn about the local traffic and the best way to handle it is the fast and economical way, to not say the most important [1]. With a good understanding about the proportion of cars in the roads and avenues we are studying [5], we can have a better analysis of it and decide on a way to solve or at least mitigate the problem, solutions that can alter from improving traffic lights or public transports to set up new roads [4][19].

Some of the first strategies used to identify vehicles [12] made use of approaches to filter different features present in the image in order to detect the objects, the most common was the background subtraction that detach moving objects from the scenery to identify it, another technique detect the vehicle based on the shadow created underneath it. While some of the first approaches for tracking were based in the contour method, checking the boundaries of the vehicles and the pattern method which uses the YCrCb color space to construct an initial background and consequently find vehicles location and keep track it [12].

As the technology advanced, other work started an analysis about the topic [48], he developed a study over some detection models based on CNN with the goal to find the best model to detect vehicles. According to the outcomes of the study, the researcher found that the Yolov4NCIoU performed the best among the different options. The Yolov4NCIoU is a modified version of Yolov4, where the author made some improvements to the loss function. When tested on the researcher's dataset that consists of pictures taken from the roads of Florianopolis during the daylight, it showed a result of 88,2% mAP and 18 FPS of speed when tested in his dataset.

Another paper [49] did a similar analysis with four different detectors and four different trackers to compare the best among the 16 combinations they implemented would have a greater performance. The YOLOv4 detector combined with the deep-SORT tracker turned out to be the best model for counting cars. According to their findings, the counting results were approximately 107%. This calculation was made by dividing the number of cars the model identified by the actual number of vehicles in the test scenario. Their models sometimes incorrectly identified objects as cars

as well as double counted them, creating what we call "false positives". These false positives caused the count to be higher than expected.

In this study, they developed a huge database with 98856 images of roads scenes featuring vehicles at night. Additionally, they showcase a novel framework based on a grid of foveal classifiers which handled unshaped light patterns. This method presented greater accuracy results and faster operation time than the YOLOv3 and R-CNN when the comparison was made.

Jhon's paper [48], also developed the tracking and counting system. For the tracker he used the deepSORT method and implemented a count system based on a middle virtual line which detects and counts every detected vehicle that crosses it. His work declares an average accuracy of 97.8% for this method [48].

In an effort to contribute to these presented projects, our work implemented a system with the finest attributes highlighted in these previous studies. The main objective of our system is to detect and count vehicles during the nocturnal phase of the day since automatic counting systems are recommended to perform in the course of 24 hours [3]. We aim to make this dissertation a complement study of these previous ones and advance the capabilities of vehicle detection and counting in low-light conditions.

We are also interested in evaluating datasets provided by surveillance cameras in order to reduce hardware's cost. If we have these cameras to do uninterrupted work for us, we can gather an immeasurable amount of data which can lead us to another level of expertise in respect to the increase of traffic.

Chapter 3

Car Detection

This chapter presents every information related to the object detection process done in the project of this thesis. We start with the overview of our object detection approach in the first section and the dataset generation process is presented in the second section. In the third section we show our results and do an analysis about the models we tested and finish this chapter with some notes about all the activity done so far in the fourth and last section.

3.1 Project Overview

The first step of this project is the car detection procedure. To accomplish that, this thesis will focus on training a YOLOv4 based solution which presents good detection results as well as good speed for real time applications according to the literature [42], [48]. We decided to use this version since the newer ones available in the beginning of this study, like YOLOv5 and YOLOv6 were not developed by the official authors and presented equivalent performance as YOLOv4. In the middle of 2022, the official authors released the official YOLOv7 which presented a good upgrade in performance [93], but since we were near the end of the study we decided to stick with the fourth version.

To achieve a good detector we prepared a database, which will be explained in the section 3.2, and compounded several models under individual conditions and different datasets to accomplish our goal.

The workflow displayed in the figure 3.1 indicates the procedures implemented in our detection process including the YOLOv4 training part. Our detector will work reading the input video, it will detach each frame and detect any object available, every detection made will be pointed and drawn in the image as well as written and saved inside a reference list that can be used for tracking purposes (which will be explained later on into the next chapter).

Training Outline

The training processes were executed in the Google Colaboratory, see section 2.7, using its pro version to handle powerful operations faster than its free version.

The algorithm code used to train the dataset was based on the main article published by the original author [42], adapting the code to run the specific situation present in this thesis which is detecting vehicles.

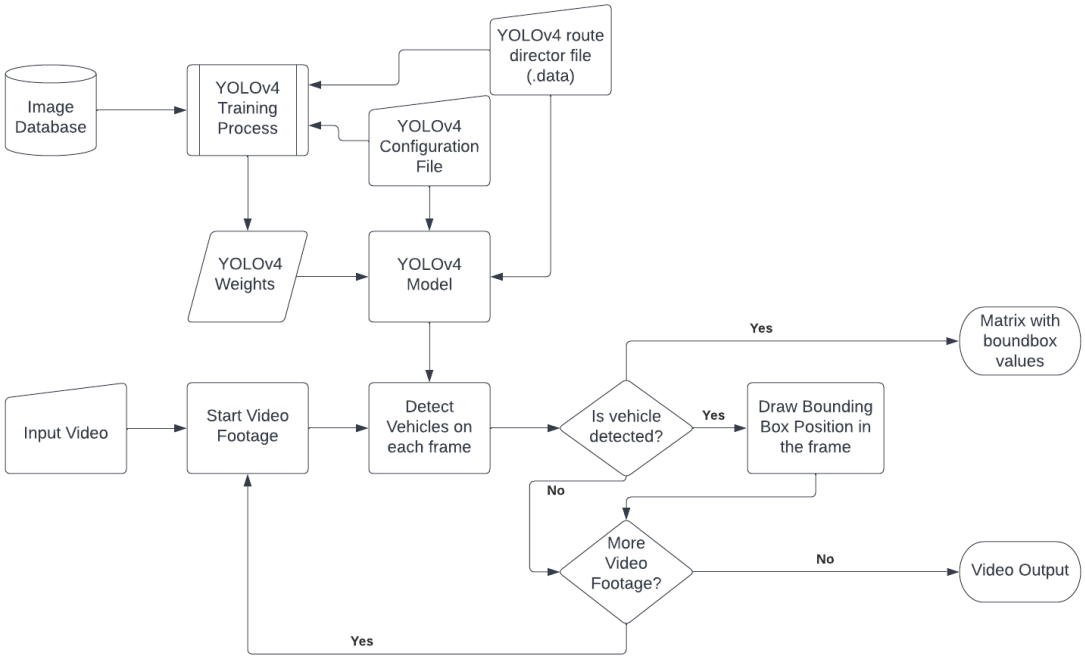


Figure 3.1: Detection phase workflow.
Source: Author

A few different batch sizes and learning rates were chosen in the training process to contrast the effectiveness of these choices in the final result taking into account some discoveries of previous studies [48] [23]. With this range of variables we can obtain a review of whether they interfere in the model generalization or if there's a combination which could give us better results.

3.2 Data Collection

This project's data collection process was created after a free publicly available dataset labeled Nighttime Vehicle Detection database (NVD Dataset). This database was created by a research group, called Image Processing group, focused on applications of Digital Images and Video Processing based on the Madrid Polytechnic University [71].

Scenario	Resolution	Number of Sequences	Number of Frames	Frames per Second
Richmond	450 × 800	11	10279	2
Walnut	480 × 640	29	28910	8
California	480 × 640	20	19998	13
Connecticut	480 × 704	21	20699	5
Gelderland	720 × 1280	15	14970	25

Table 3.1: The NVD Database Summary
Source: [70]

The NVD Dataset is composed of 96 sequences consisting of 94856 images of real traffic surveillance cameras from five different locations on highways and details can be found in table 3.1 [70]. The cities addressed in this database are Richmond, Wal-

nut, California, Connecticut and Gelderland and all images gathered were recorded in monochrome video at night as it is displayed in Figure 3.2.

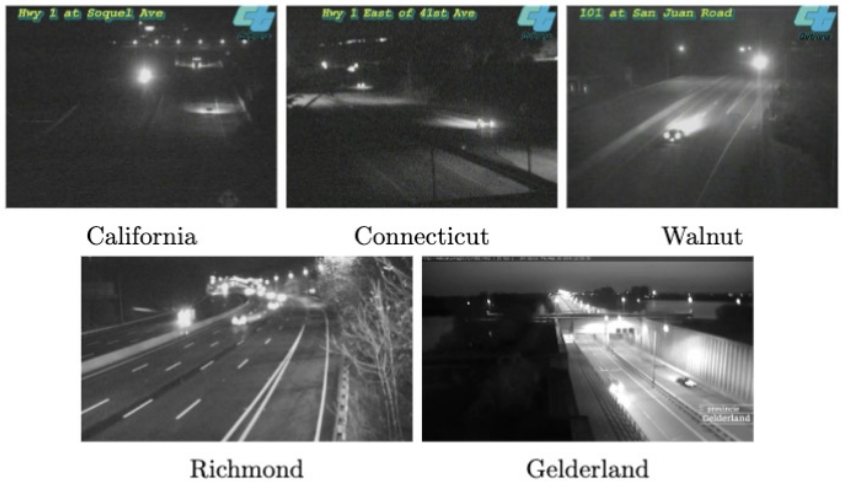


Figure 3.2: Samples of the Nighttime Vehicle Detection database (NVD). Source: [70]

As shown in the examples, this dataset has dissimilarities between figures resolutions as well as complex illumination patterns manifested by the vehicles which amplify the detection process complexity.

The ground-truth information was provided along with this database for every object of all images. The method used to accomplish that was the point-based annotation technique, which consists of annotating the central point of each object. They also provided conversion algorithms to convert the ground-truth information to other formats. Using that, we gathered the ground-truth in COCO format and simply reshaped it into YOLO format to work on our project. Below in the figure 3.3 we label the previous samples and show the difference between both the annotation's types we mentioned, the red squares representing the bounding boxes annotation and the yellow dots showing the point-based annotation.

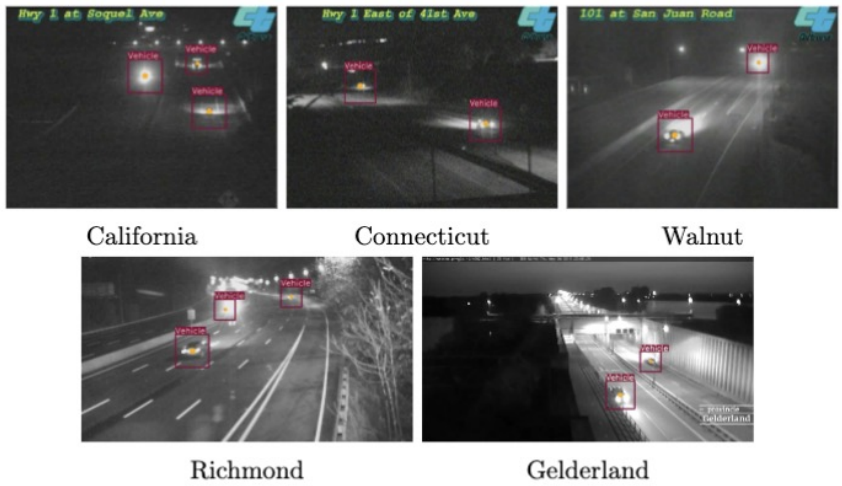


Figure 3.3: NVD database with both annotation methods examples. Source: Author

Due to hardware processing limitations, we opted to restrict the number of images utilized in the training phase rather than training every single image within this database. The selection was made according to the number of vehicles detected across the whole sequence, aiming to gather an appropriate set of training objects. The chosen ones are presented on the table 3.2 as well as some important numbers from this training procedure in the table 3.3. It is important to note that the sequences shown in the table for the test and validation sets were split randomly, with the goal of having the validation set contain 65% of the images and the test set containing the remaining 35%.

City	Training				Test and Validation	
California	seq43	seq44	seq50	seq56	seq42	seq58
Connecticut	seq104	seq107	seq113	seq118	seq102	
Gelderland	seq122	seq126	seq132	seq135	seq123	
Richmond	seq2	seq4	seq6	seq9	seq10	
Walnut	seq13	seq18	seq24	seq33	seq34	

Table 3.2: Sequences numbers implemented on detection phase
Source: Author

Parameter	Dataset
Total Image Number	26000
Training	20000
Validation	3900
Test	2100
Total Number of labelled vehicles	54100

Table 3.3: Detection data overview.
Source: Author

In our analysis, we split each line of the table 3.2 as one different model and along with these five models we added another one named AllCities model, which is a composition of all the sequences appearing in this table. These six distinct datasets will enhance our study to define a good detector that aligns with our requirements. In order to evaluate our trained approaches, we are going to use the unused sequences from the NVD Database, which were not utilized in the training of our network.

3.3 Results

In this section we find every knowledge we learned about the detection process and seeking for a better way to handle the information we decide to tear it up into some subsections, first we are going to talk about the output files that the YOLOv4 provide us in this procedure, after we are going to discuss about the results obtained in the training process and end talking about the results of our detection experiments in the videos of the roads of the NVD Database.

3.3.1 Result Files

The YOLOv4 detector provides two files after the process of detection. The first one is the video itself with the bounding boxes drawn around the objects detected.

The second one is a text file that shows two informations, the FPS speed of the detection for each frame that will not be our main focus since it is not vital for our project to work on a real time environment even though it is a good option for the future and a vector with detection and location values of each object detected in the frame like the example 3.1:

$$Data = [Class, Confidence, left\_x, top\_y, width, height] \tag{3.1}$$

This data vector presents six arguments to each object detected in the frame, the first gives us the machine class' prediction while the second one gives the level of conviction it has about the class. These two first variables give us the machine's classification information while the last four shows the location coordinates of the object in the image, presenting the x and y coordinates in the image as well as its width and height.

With the classification and localization of each object in the image, we can enhance our system and implement a huge variety of applications and that is why the detection process of a computer vision feature, which provides this information, is so important.

3.3.2 Training Results

To bring out a better analysis of the results acquired in the training process, we first need to know what our goal is and determine the best way to achieve it with the best results possible.

First things first, this thesis' chapter focuses on identifying vehicles, while in the night shift of the day where the main goal is to count them. In order to accomplish that, we need our model to correctly identify all the vehicles present in the video tests. That is not an easy task though, since there is no thing as a perfect system which classifies every object correctly.

A good way to deal with that case, is by minimizing the False Negative index as low as possible, i.e. maximize our Recall metric. That way we might have more vehicles identified in the video than there really are, but since our main goal is to count them when traveling across a virtual line instead of only detecting them, it is acceptable to deal with some false positives as light poles or light reflections that might occur in the process.

Both tables 3.4 and 3.5 presents the metric results obtained after the training process of our six different datasets, considering a learning rate of $1e^{-3}$ and $1e^{-5}$, respectively, and two different batch sizes for each one. Each dataset had at least 32 epochs while training and that value was enough to converge the accuracy results through this course. At first, it is notable when comparing the two tables that the one with better performance regarding the F1-Score (see section 2.6) is the table 3.4 with results up to 98%, while the table where the learning rate is lower, the better F1 Score was around 90%. Another comparison observed is that the models with 32 batch size tend to have a better performance when compared with the 16 batch size version.

In order to cut down our approaches list and since the models with a higher learning rate showed better results so far, we are going to continue our analysis and check the mAP and IoU metrics of only these models to find if there is a model that

Learning Rate = 10^{-3}							
Training Dataset	Batch Size	TP	FP	FN	Precision	Recall	F1-Score
California	16	3239	105	109	0.97	0.97	0.97
	32	3242	93	106	0.97	0.97	0.97
Connecticut	16	1906	66	127	0.97	0.94	0.95
	32	1910	49	123	0.97	0.94	0.96
Gelderland	16	787	33	22	0.96	0.97	0.97
	32	795	21	14	0.97	0.98	0.98
Richmond	16	3755	1607	375	0.70	0.91	0.79
	32	3772	1527	358	0.71	0.91	0.80
Walnut	16	1317	119	68	0.92	0.95	0.93
	32	1316	137	69	0.91	0.95	0.93
AllCities Model	16	9041	2410	676	0.79	0.93	0.85
	32	9058	2142	659	0.81	0.93	0.87

Table 3.4: Validation results obtained for learning rate of $1e^{-3}$.
Source: Author

Learning Rate = 10^{-5}							
Training Dataset	Batch Size	TP	FP	FN	Precision	Recall	F1-Score
California	16	3046	607	302	0.83	0.91	0.87
	32	3098	475	250	0.87	0.93	0.90
Connecticut	16	1770	383	263	0.82	0.87	0.85
	32	1799	359	234	0.83	0.88	0.86
Gelderland	16	746	240	63	0.76	0.92	0.83
	32	742	191	67	0.80	0.92	0.85
Richmond	16	3584	2491	546	0.59	0.87	0.70
	32	3579	2417	551	0.60	0.87	0.71
Walnut	16	1267	336	118	0.79	0.91	0.85
	32	1270	298	115	0.81	0.92	0.86
AllCities Model	16	8486	3283	1231	0.72	0.87	0.79
	32	8654	3260	1063	0.73	0.89	0.80

Table 3.5: Validation results obtained for learning rate of $1e^{-5}$.
Source: Author

won't be a good fit for our project. The table 3.6 presents the mAP and IoU metrics of these models.

With mAP values from around 97.6% until around 80% all of these candidates seem to fit our project well even though the models with 32 batch size value show slightly better results in comparison with its 16 batch size. Some of them might have presented a poor IoU metric but it is worth highlighting that it is due to the conversion process of the point-based annotation technique, the figure 3.3 shows a good example of this fact. So even though the predictions and labels might be on point for a human perspective, the ground truth might be in a different way than the prediction, that way the union value becomes higher than the intersection, decreasing the IoU metric of these models. (check section 2.6).

Learning Rate = 10^{-3}			
Training Dataset	Batch Size	IoU (%)	mAP (%)
California	16	80.56 %	97.62 %
	32	80.90 %	97.64 %
Connecticut	16	82.30 %	94.83 %
	32	83.10 %	94.65 %
Gelderland	16	83.27 %	93.70 %
	32	85.73 %	94.51 %
Richmond	16	52.08 %	80.35 %
	32	53.16 %	81.27 %
Walnut	16	77.91 %	95.65 %
	32	77.32 %	95.71 %
AllCities Model	16	63.47 %	90.50 %
	32	65.52 %	91.23 %

Table 3.6: mAP and IoU metrics for a learning rate of $1e^{-3}$.
Source: Author

3.3.3 Experiment Results

From this section we will only continue our comparison with the models with learning rate of 0.001 and batch size of 32 which were the ones with better results so far. In the figures 3.4 and 3.5, we tried on our detectors in the five landscapes already mentioned in this thesis, these frames from the videos show how well our models performed in the simulation in the streets using videos that weren't present in the training phase.

At first sight, in the figure 3.4, the models trained after the datasets of California, Connectituc and Walnut seem to have greater bounding boxes than the vehicles expect, being the Connecticut dataset the worse one in comparison. Looking back to the table 3.6 it explains why the IoU metric of the others seems to be low in contrast and that was anticipated as well as the reason besides this fact.

Following the figure 3.5, the second point worth mentioning is how poorly the Gelderland model performed in the other environments besides its own. The reason that leads to this behavior is because of the lack of true positive samples in the training phase. The roads of the Gelderland dataset seem to not be very used during night time and that lead to a lack of true positive cars fed into our model of this city, leading it to a lack of generalization and not being able to detect cars in the other roads consistently.

The Richmond model gave better results when compared with the previous models mentioned, presenting only some false negatives and some misplaced bounding boxes as demonstrated in the image 3.5, but overall the best model was, the already expected, AllCities detector since it was fed up with more images than any other detector enhancing its generalization, even though this fact was not present in the training results. The AllCities Model did great in all of the scenarios tested, showing good results as well as a better confidence value when compared to the others.

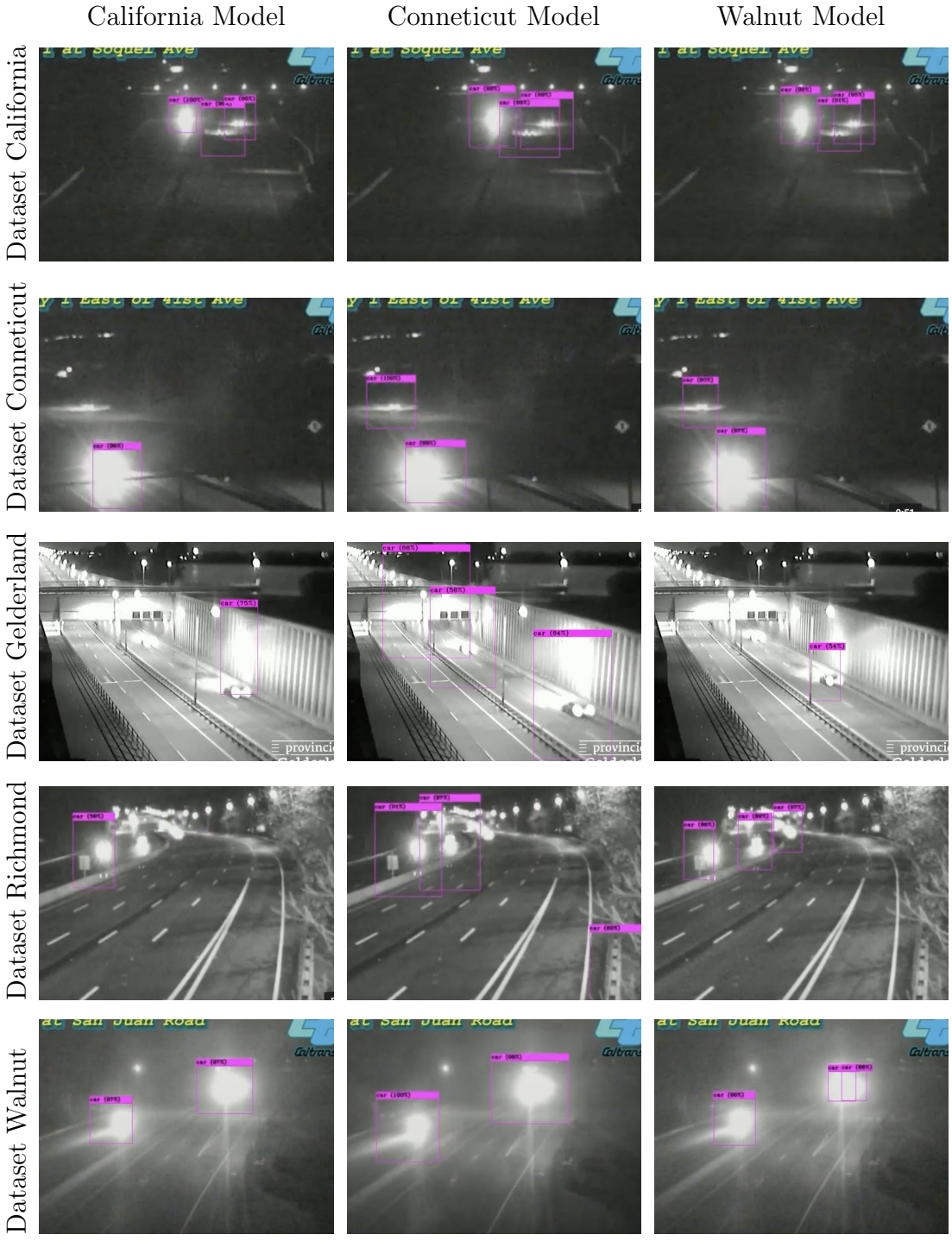


Figure 3.4: Experiments' detection results for California, Conneticut and Walnut models.

Source: Author

Figure 3.5: Experiments' detection results for Gelderland, Richmond and AllCities models.

Source: Author

In the figures 3.4 and 3.5, we show a result comparison between the models we trained, but from this point we are going to add up other interesting results that appeared while we were doing our analysis. These following images focus on the mistakes made by the models during the tests of the detection phase but show some good results as well.

(a) (b) (c)

Figure 3.6: Interesting results that the California model presented during the tests.
Source: Author

Beginning with the California Model in the Figure 3.6, it showed inconsistency during detection, missing a lot of objects when they were far from the camera and occluded by some nearby vehicles, it also presented some shifted bounding boxes during the tests. The Connecticut model had huge bounding boxes in comparison to the vehicles as already said as well as some false positives in light poles and misplaced bounding boxes in some cases as shown in figure 3.7.

(a) (b) (c)

Figure 3.7: Interesting results that the Connecticut model showed during the test phase.

Source: Author

(a) (b) (c)

Figure 3.8: Interesting results that the Gelderland Model demonstrated in the course of the tests.

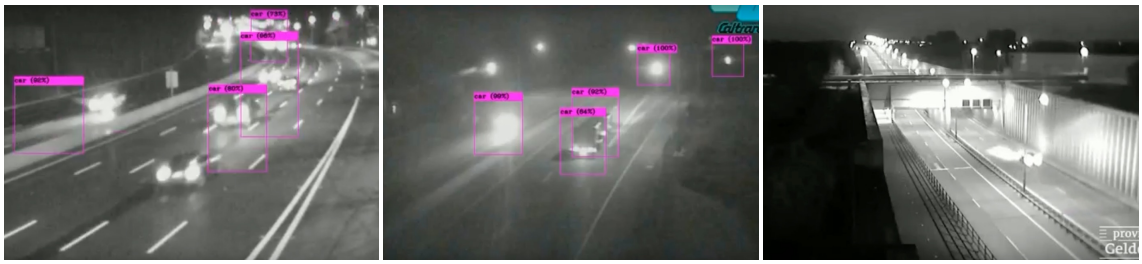
Source: Author

The Gelderland Model was, without a doubt, the worst detector around the models. It does detect some vehicles in the others scenarios though as well as track them for a couple of frames, despite not being enough to be considered as an useful detector. As it shows in the figure 3.8, it only detects one vehicle when this event happens.

(a) (b) (c)

Figure 3.9: Intriguing results that the Richmond Model presented during the tests. Source: Author

The Richmond model handled multi-object detection well, but presented low accuracy or didn't detect objects that moved far away from the viewpoint, in some cases it also created a new bounding box between two different vehicles. The figure 3.9 shows in the details these results.

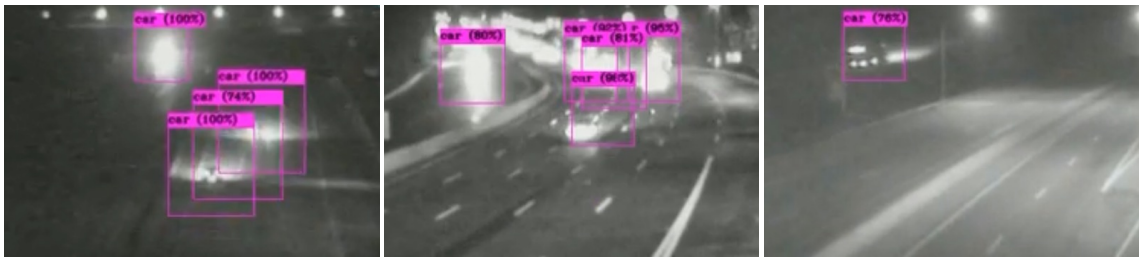


(a) (b) (c)

Figure 3.10: Interesting results that the Walnut Model manifested during the tests. Source: Author

Like some other models, the Walnut model presented bounding boxes shifted from the real object as well as occlusions from near vehicles or light poles, as we can take a glimpse in figure 3.10. It also showed double bounding boxes for the same vehicle when it was a big size one.

Lastly, The AllCities model showed a few false positives in some cases when cars were approaching each other in the roads for a couple of frames as demonstrated in the figure 3.11, but it handled well the multi object detection task, even though lose track of far vehicles sooner than when they show up one at each time, that occurred because of the different kinds of illumination conditions presented when more cars are in the image. It also detected objects that were traveling in the side road for a short time despite weren't enough to track them.



(a) (b) (c)

Figure 3.11: Captivating results that the AllCities model exhibited in the test phase. Source: Author

3.4 Final Thoughts

In the course of this chapter we showed the dataset gathering process and tried to implement it in different ways regarding vehicle detection at night time.

Our results expressed before revealed some object detection classic issues as occlusions between the objects or with the scenery and different illumination patterns interfering in the detection. It also exhibited some shifted bounding boxes in the process due to the conversion process from the point based annotation to the YOLO supported version, that happened because the point label marks weren't always in the same spot of the vehicle and were placed in the front of the vehicles whenever a new one entered the sight view.

After this detailed analysis of the criteria, we chose to use the named AllCities Model, with a learning rate of 10^{-3}, and a batch size of 32 in the next steps, as it proved to be the design that best met our expectations among the options presented in this thesis, with a mAP value of 91.23%.

This outcome is slightly better when compared with the results of other studies, such as the one presented by Andrés [70] and Jhon [48]. It is valid to highlight that even though we used the same database as Andrés, we did not use the same training and test sets. His study also used the predecessor model of ours, the YOLOv3. As for Jhon, he achieved an 87% mAP performance and used the same detection and tracking tools as our study.

The AllCities Model might have shown some detection errors around the process but in the following chapter we will analyze and check if these issues will have a great impact on our results.

Chapter 4

Car Tracking and Counting

Now that we have our detector, we are going to focus on the last two features we need to finish our project, the tracking and counting steps. Even though they are different techniques, the counting technique heavily relies on the tracking algorithm and that is why they are explained in the same chapter.

In this chapter we show all details regarding the object tracking and object counting steps of our project, we start by presenting the way we are going to embrace these steps in the first section and talking about the method we will use to analyze our approach in the second one. After in the third section we present the results we got during the process of both tracking and counting steps and finalize with some thoughts about the chapter in the fifth section.

4.1 Proposed Solution

To achieve our goal, we need to complement our detector with both tracker and counter techniques. Our detector already identifies the presence and location of the objects in the image and with the text file that our algorithm provides us, we handle this data to track these detected objects around the frame.

To do this track, we decided to make use of the DeepSORT feature. In a loop, it first runs the detector to find the objects, after we have them defined, the framework estimates these object's centroid future location making use of the kalman filter. Next, the tracker will make a data association between the two following frames using a threshold value to do it. This threshold will decide if these centroids of both frames are from the same object or not. More material of this tracker can be found in the section 2.4.2 where we comment on it in more details.

For the counting step procedure of this project, we choose to use the virtual line counting technique. In order for it to work, at the start of the process, a line will be set across the road. As the video goes on and the algorithm keeps track of the vehicles, whenever a detected object moves through any section of the line it will be counted and sum up to the total, giving the full amount of vehicles that went north and the ones which moved south at the end. Check 2.5 for more details about this counting method.

The figure 4.1 demonstrates the workflow of the system proposed for this chapter. Starting from the output of the detector until the result video after all the track and count procedure.

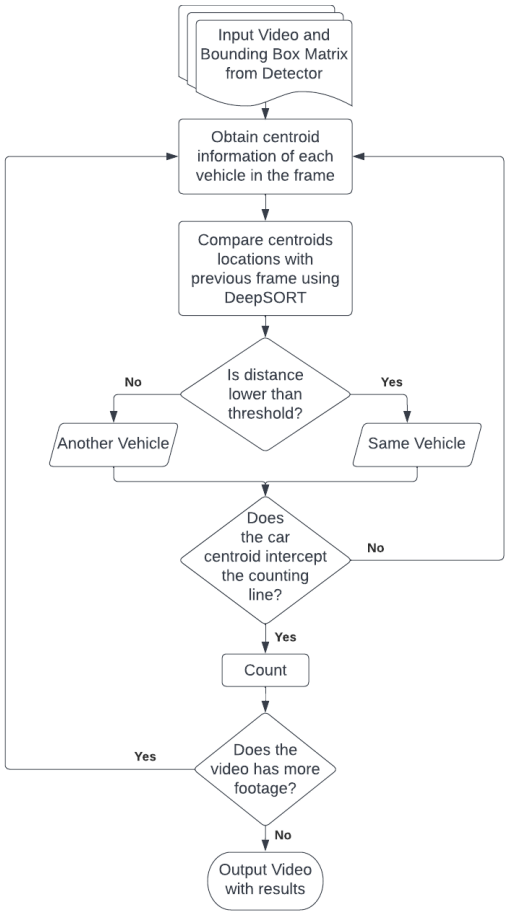


Figure 4.1: Tracking and counting workflow
Source: Author

4.2 Analysis Overview

In this chapter, we decided to make our analysis for both track and count techniques with the sequences from the NVD Dataset [71] that were not used previously in this thesis in an effort to represent real deal videos where the model has never faced before. In the table 4.1 we show the sequences we generated the videos.

The data study we developed consists of two parts, the tracking and the counting steps. In the first one, we are interested in discovering if the tracker can handle the vehicles at night checking if it can keep up with the objects without losing track of it, to accomplish that we are going to check if it keeps the same object ID as the vehicle moves through the video. In the second part we test the counting algorithm, to investigate if the system shows a good counting performance and if there were problems that we faced along this process.

For all the comparisons with the algorithm results, we manually counted all vehicles presented in each video to have a ground truth metric and this data will be shown in the result tables of the next section as well as the machine results.

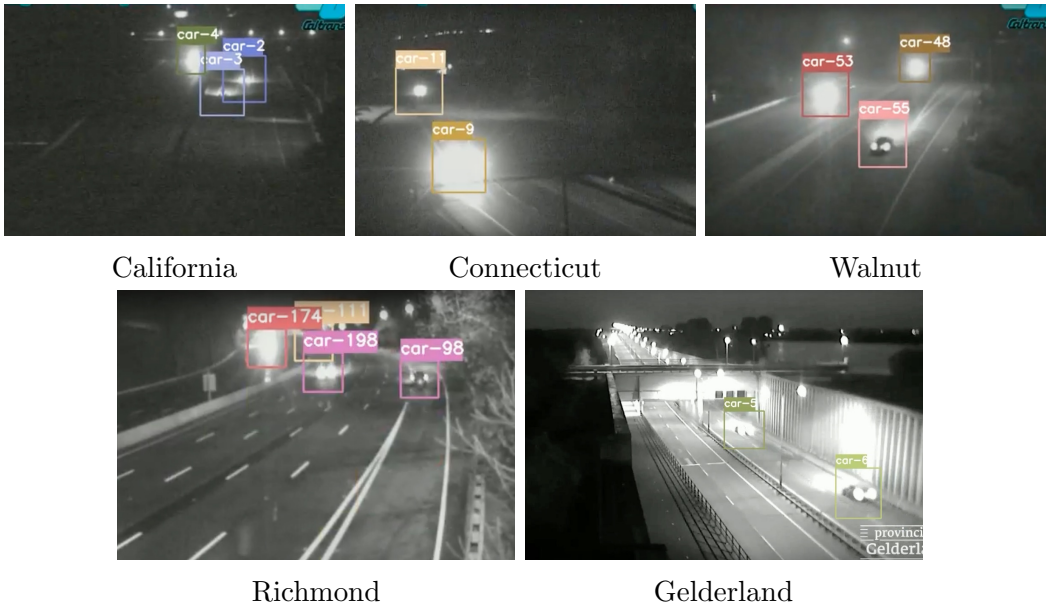
Scenario	Video Samples				
California	seq41	seq46	seq48	seq52	seq54
Connecticut	seq100	seq106	seq109	seq110	seq115
Gelderland	seq125	seq128	seq129	seq130	seq134
Richmond	seq1	seq3	seq7	seq8	seq11
Walnut	seq12	seq20	seq22	seq26	seq27

Table 4.1: Sequences number chose for tracking and counting test
Source: Author

4.3 Results

4.3.1 Vehicle Tracking

After implementing the DeepSORT technique and running it through our videos, we obtain our output files with our detected objects and each of them receive a different ID. The figure 4.2 gives us some examples of the results.



California Connecticut Walnut

Richmond Gelderland

Figure 4.2: Tracking samples of the Nighttime Vehicle Detection database (NVD).
Source: Author

The following table shows the ID assignments that the DeepSORT technique made on the objects that appeared in the videos for each landscape. The first five tables show the results for each one of the five landscapes we are working on and the last one shows a summary of the results for each city. Each video is named after the sequence that generated it, we also included to the table the ground truth provided by a manually count by a person.

California Sequences	Ground Truth	ID Numbers Tracked	Accuracy (%)
seq41	6	6	100%
seq46	9	10	90%
seq48	6	8	75%
seq52	5	5	100%
seq54	9	16	56,25%
Total	35	45	77,78%

Table 4.2: California tracking results.
Source: Author

Connecticut Sequences	Ground Truth	ID Numbers Tracked	Accuracy (%)
seq100	15	23	65,21%
seq106	12	23	52,17%
seq109	14	23	60,87%
seq110	11	18	61,11%
seq115	10	11	90,90%
Total	62	98	63,26%

Table 4.3: Connecticut tracking results.
Source: Author

Gelderland Sequences	Ground Truth	ID Numbers Tracked	Accuracy (%)
seq125	3	4	75%
seq128	5	7	71,43%
seq129	5	5	100%
seq130	2	2	100%
seq134	2	2	100%
Total	17	20	85%

Table 4.4: Gelderland tracking results.
Source: Author

Richmond Sequences	Ground Truth	ID Numbers Tracked	Accuracy (%)
seq1	123	237	51,90%
seq3	119	248	47,98%
seq7	254	469	54,15%
seq8	299	487	61,39%
seq11	106	236	44,91%
Total	901	1677	53,72%

Table 4.5: Richmond tracking results.
Source: Author

Walnut Sequences	Ground Truth	ID Numbers Tracked	Accuracy (%)
seq12	25	53	47,17%
seq20	15	32	46,87%
seq22	13	36	36,11%
seq26	21	69	30,43%
seq27	13	39	33,33%
Total	87	229	37,99%

Table 4.6: Walnut tracking results.
Source: Author

Overall Results	Ground Truth	ID Numbers Tracked	Accuracy (%)
California	35	45	77,78%
Connecticut	62	98	63%
Gelderland	17	20	85%
Richmond	901	1677	53,73%
Walnut	87	229	37,99%

Table 4.7: Tracking results summary.
Source: Author

We can see from the tables above that the landscapes where our tracker most struggled working on were the ones that had more traffic on it while in the other places it showed a great performance keeping the same ID for most of the vehicles, but as stated we faced some issues during this process. In the following images we exhibit the track problems that we experienced during our tests.

Starting with the figure 4.3, it display a change of ID after our detector loses the track of the vehicle with ID 2 for a period of time due to the light patterns that it experienced, when the detector recognize the object again it gave another ID for the same object.

Figure 4.3: Tracking error sample: Lost of bounding box.
Source: Author

In the figure 4.4, we expose an issue caused by the occlusion of two vehicles. As the first vehicle was going north, when it went across the second vehicle and left the image, the tracker thought of the second vehicle as the first one, assigning the same ID value to it.

Figure 4.4: Tracking error sample: Same ID for two different vehicles.
Source: Author

The last type of issue we found during our tests was caused by occlusion from another object, like other vehicles or even the watermark from the video, like demonstrated in the figure 4.5, when it happened the algorithm usually assigned another ID for the object right after the occlusion.

Figure 4.5: Tracking error sample: Occlusion by objects
Source: Author

4.3.2 Vehicle Counting

There are several ways to implement a virtual line, in this project we decided to use the line segment intersection method. Two lines intersect if and only if the extremities of each are separated for the other's segment and that happens when the orientation of the three ordered points created in the plane has different orientations.

To find the orientation of an ordered triplet of points we calculate the slope of them using the formula 4.1:

$$slope = (y_b - y_a) * (x_c - x_b) - (y_c - y_b) * (x_b - x_a) \tag{4.1}$$

Depending on the outcome of this expression, we have different kinds of orientation that can vary as below:

- If the outcome is zero, then $\theta = \varphi$. Hence the orientation is collinear.

- If the outcome is negative, then $\theta < \varphi$. Hence the orientation is anti-clockwise.
- If the outcome is positive, then $\theta > \varphi$. Hence the orientation is clockwise.

With these orientations known, we discover if these endpoints intersect, if so, the counter will sum up one, to the final count.

Another matter that has to be settled before the process is to define the place where the line will be settled. In the figure 4.6 we present our landscapes and show the places where we set the virtual line for each of them. These places were chosen in a way that helps the object count for both directions.

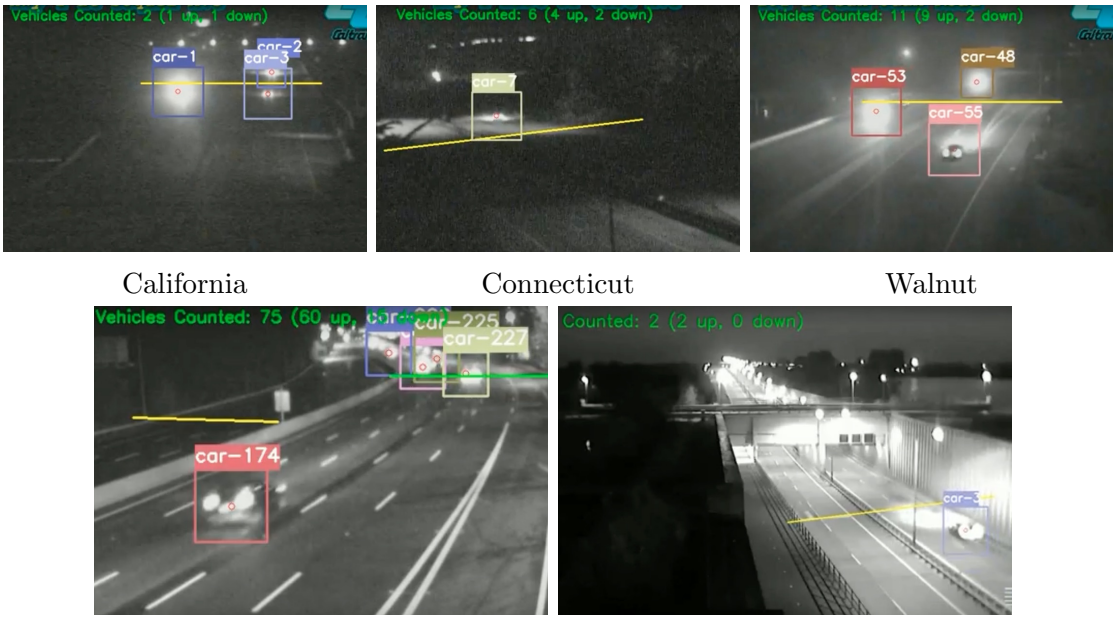


Figure 4.6: Virtual line places for each landscape.
Source: Author

The following five tables present the counting results obtained during our project for each one of the cities. The sixth one will summarize the results from all the cities present in this work that will be used to end our discussion. The counting was done by car's direction and we are going to compared this data as well.

California	Vehicles going North			Vehicles going South			Total	
Sequences	A.C.	G.T.	ACC (%)	A.C.	G.T.	ACC (%)	AC/GT	ACC (%)
seq41	5	5	100%	1	1	100%	(6/6)	100%
seq46	5	5	100%	1	1	100%	(6/6)	100%
seq48	1	1	100%	2	2	100%	(3/3)	100%
seq52	2	1	50%	4	3	75%	(6/4)	66,7%
seq54	1	1	100%	4	6	75%	(5/7)	77,77%
Total	14	13	92,85%	12	13	84,21%	(26/26)	86,66%

Table 4.8: California counting results.
Source: Author

Connecticut	Vehicles going North			Vehicles going South			Total	
Sequences	A.C.	G.T.	ACC (%)	A.C.	G.T.	ACC (%)	AC/GT	ACC (%)
seq100	12	12	100%	5	1	20%	(17/13)	76,47%
seq106	4	3	75%	8	8	100%	(12/11)	91,67%
seq109	10	10	100%	6	2	33,33%	(16/12)	75%
seq110	8	8	100%	5	3	60%	(13/11)	84,61%
seq115	3	3	100%	7	7	100%	(10/10)	100%
Total	37	36	97,3%	31	21	67,74%	(68/57)	83,82%

Table 4.9: Connecticut counting results.
Source: Author

Gelderland	Vehicles going North			Vehicles going South			Total	
Sequences	A.C.	G.T.	ACC (%)	A.C.	G.T.	ACC (%)	AC/GT	ACC (%)
seq125	1	1	100%	2	2	100%	(3/3)	100%
seq128	3	3	100%	1	1	100%	(4/4)	100%
seq129	3	3	100%	2	2	100%	(5/5)	100%
seq130	2	2	100%	0	0	100%	(2/2)	100%
seq134	1	1	100%	1	1	100%	(2/2)	100%
Total	10	10	100%	6	6	100%	(16/16)	100%

Table 4.10: Gelderland counting results.
Source: Author

Richmond	Vehicles going North			Vehicles going South			Total	
Sequences	A.C.	G.T.	ACC (%)	A.C.	G.T.	ACC (%)	AC/GT	ACC (%)
seq1	80	85	94,44%	18	35	67,3%	(98/120)	84,5%
seq3	70	80	88,89%	20	37	68,52%	(90/117)	81,25%
seq7	180	209	87,81%	21	48	64%	(201/257)	82,11%
seq8	188	228	85,07%	44	71	72,45%	(232/299)	81,69%
seq11	57	72	82,76%	11	30	61,22%	(68/102)	75%
Total	575	674	87,19%	114	221	67,38%	(689/895)	81,29%

Table 4.11: Richmond counting results.
Source: Author

Walnut	Vehicles going North			Vehicles going South			Total	
Sequences	A.C.	G.T.	ACC (%)	A.C.	G.T.	ACC (%)	AC/GT	ACC (%)
seq12	6	6	100%	9	11	84,61%	(15/17)	89,47%
seq20	8	6	75%	10	9	90%	(18/15)	83,33%
seq22	5	4	80%	3	3	100%	(8/7)	87,5%
seq26	14	13	92,86%	5	5	100%	(19/18)	94,73%
seq27	9	10	90,9%	5	4	80%	(14/14)	87,5%
Total	42	39	88,63%	32	32	88,89%	(74/71)	88,75%

Table 4.12: Walnut counting results.
Source: Author

Overall	Vehicles going North			Vehicles going South			Total	
Results	A.C.	G.T.	ACC (%)	A.C.	G.T.	ACC (%)	AC/GT	ACC (%)
California	14	13	92,85%	12	13	84,21%	(26/26)	86,66%
Connecticut	37	36	97,3%	31	21	67,74%	(68/57)	83,82%
Gelderland	10	10	100%	6	6	100%	(16/16)	100%
Richmond	575	674	87,19%	114	221	67,38%	(689/895)	81,29%
Walnut	42	39	88,63%	32	32	88,89%	(74/71)	88,75%

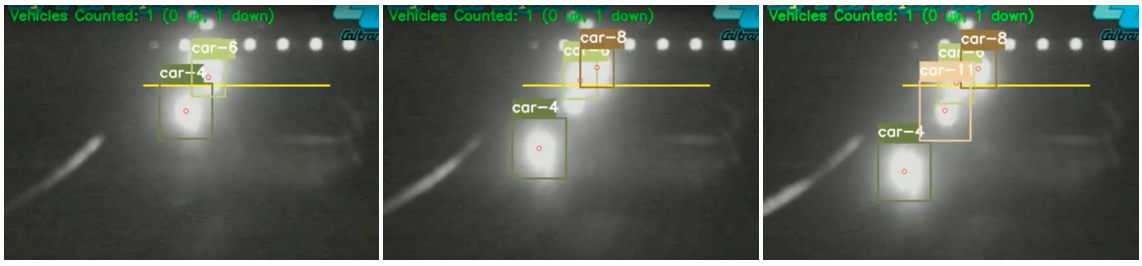
Table 4.13: Overall counting results.
Source: Author

As presented in the tables, for all cities of this work, the counter had an overall result over 81%, even reaching 100% in Gelderland. These results were a satisfactory outcome but it is noteworthy that we encountered some mistakes during our analysis as well. Most of them were because of detection irregularities that prevented the tracker from following the object trace, i.e. losing the data it produced. An example of this situation is demonstrated in the figure 4.7 where the car next to the bus was not detected, hence, it was not count.

(a) (b)

Figure 4.7: Example of detection error during count tests.
Source: Author

We also faced some issues with full beam headlights in some cases, that the highlight occluded other vehicles nearby preventing the detector from finding them. It resulted in the line missing them and not counting these affected objects as they passed through it. The figure 4.8 shows an example of this situation.



(a) (b) (c)

Figure 4.8: Example of vehicle occlusion caused by beam headlights of another one.
Source: Author

4.4 Final Thoughts

Through the course of this chapter we presented every step taken regarding the tracking and counting parts as we said in the beginning.

Our tracker faced difficulties to follow the objects because of the lack of descriptors for our appearance vector caused by the low illumination of the places and the complex light and shadows patterns caused by the headlights of the vehicles.

On that account, we decided to not limit our counter by ID, fearing that this could lead to missing counts. Our decision gave us a few errors, but not enough to regret the decision, because considering this case scenario, we rather have a false positive than a false negative.

It is worth mentioning that we removed from the ground truth, cars that did not cross the line during the video as it was not an algorithm's fault.

With that in mind, the virtual line counter did great overall. In the scenarios with a more stable detection, as Gelderland and California it got even results with 100% accuracy in some sequences, for the landscapes with more instability by the tracker, like Connecticut and Walnut, it was required to place our line in a good spot where the tracker seemed to be more reliable to follow, in the end, it performed great as well, showing 100% accuracy in some cases.

For the city of Richmond, since it was the city with most traffic as well as most unreliability from the detector, it did not performed good at first, because there was not a spot good enough to check both sides of the road, so the line for this city was splitted up for each orientation of the venue. With that upgrade, the counter did better and the results were around 81% as well.

If we use other related studies as references [48] [49], we discover that our outcomes were indeed satisfactory. We can see that our model's performance aligns well within the expected range of results. Although our approach slightly underperformed in comparison to their methodology due to the reduced lighting conditions in our terrain, which affected the reliability of our detector, the consistent results across various scenarios provides validation for the effectiveness of our approach.

Chapter 5

Conclusion and Future Works

The aim of this project was to develop a vehicle counter approach that works in the night period of the day. This system was developed in three different components: detector, tracker and counter. For the detector, this project used the YOLOv4, a neural network based technique, for the tracker, it applied the deepSORT approach and as the counter, it was developed a virtual line method. This research achieves the goal to count vehicles during the night time of the day of five different cities around the world.

Another key point that is worth mentioning in this project was the use of images from surveillance cameras in all steps of it, in order to evaluate the possibility of future implementation in these systems that are all over the world nowadays.

During the tests, after a broad analysis considering different databases and several parameters, the YOLOv4 detector chosen for our model has 32-batch size and 10^{-3} learning rate, the model achieved an overall performance of 91.23% mean Average Precision (mAP) and a recall value of 89% on the validation set using the trained dataset. While conducting the counting process, the model also demonstrated reliable detection capabilities in the utilized videos.

The tracker did not perform as well as the detector though. To measure it, this thesis used the IDs generated by the tracker and compared these results with the manual counting provided by a human being, the tracker results ranged from 38% until 85%, showing a high rate of IDs being changed in the midst of the object's way. The deepSORT technique has the appearance vector to help the tracker in the process but it was not effective because of the low illumination present in the videos, making it challenging for the tracker to differentiate a vehicle from another.

The poor results of the tracker did interfere, but was not the main factor for the results of the counter technique, which achieved an overall success rate over 81%, reaching 100% on some occasions. It was implemented to count every object which passes through the line, that way, even if the object's ID changes, they will be counted if they cross the line.

All the steps of this project were realized using cloud features, such as Google Colab for all the code processing making use of its video cards and Google drive for storage and easy access of files. They proved their reliability and were extremely important for the realization of this project.

Despite the good outcome, there were some limitations to achieving better and more reliable results, so there are some upgrades that have to be done before implementing this application in a complete computer vision system.

The key feature that requires enhancement is the image database used for the detector. The efficiency and accuracy of both the tracking and counting procedures are linked to the detector, given that these processes deeply rely on the data extracted from detections to carry out their respective functions. Furthermore, many images have poor focus and the places where the cameras were set presented deficient illumination, resulting in some identification errors by the detector. Also, it is important to note that the annotation initially provided from the ground truth was in point-based format, and upon realizing the conversion process of this annotation, it might have had an impact on the overall dataset reliability. Lastly, even though the deepSORT had features like the appearance vector and the kalman filter it had issues following the objects.

These following recommendations for future works concerns the limitations listed above:

- More data from other cities and different vehicles perspectives should be gathered to enhance the detector generalization and success rate.
- Even though YOLO already performs data augmentation during the dataset training, more advanced data augmentation techniques should be implemented to enhance the the existing provided dataset.
- So far, the system only detects if there is a vehicle in the image. We did not implement the classification feature because of the lack of available images from other classes such as trucks and motorcycles. These increases would be great for the system and add more data for precise analysis.
- Substituting the YOLO detector with an alternative that utilizes point-based annotations as ground truth during the training phase might present an optimal choice and is highly recommended to unlock the full potential of the provided database.
- The deepSORT handled the task, but presented some issues in the process, mostly because of occlusions and we suggest the viability study of another tracker's adoption, we suggest the BYTE method since it has different association methods.

Bibliography

[1] Joe Fletcher. *BASICS OF TRAFFIC COUNTING PROCEDURES*. Acessado em: 18 ago. 2021. Highway Extension, Research Project for Indiana Counties, and Cities (HERPICC), 1992. URL: https://docs.lib.purdue.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=3614&context=roadschool.

[2] Alexx Kay. *How-To Artificial Neural Networks*. 2001. URL: https://www.computerworld.com/article/2591759/artificial-neural-networks.html/.

[3] BOWLING GREEN KENTUCKY. *TRAFFIC DATA COLLECTION PROCEDURES*. Acessado em: 18 ago. 2021. The City of Bowling Green Public Works Department, 2002. URL: https://www.bgky.org/files/hSLFx00I.pdf.

[4] Anthony Downs. *Traffic: Why It's Getting Worse, What Government Can Do*. Acessado em: 18 ago. 2021. Brookings, 2004. URL: https://www.brookings.edu/research/traffic-why-its-getting-worse-what-government-can-do/.

[5] DNIT. *MANUAL DE ESTUDOS DE TRÁFEGO*. Acessado em: 18 ago. 2021. DNIT, 2006. URL: https://edisciplinas.usp.br/pluginfile.php/5068343/mod_resource/content/0/DNIT%5C%202006_manual_estudos_trafego.pdf/.

[6] Shivaji P. Mirashe and N. V. Kalyankar. "Cloud Computing". In: *CoRR* abs/1003.4074 (2010). arXiv: 1003.4074. URL: http://arxiv.org/abs/1003.4074.

[7] Mobilize. *Os custos do congestionamento na cidade de S.Paulo*. Acessado em: 18 ago. 2021. Mobilize, 2010. URL: https://www.mobilize.org.br/midias/pesquisas/custos-do-congestionamento-em-sp.pdf.

[8] Zhe Liu and Renaud Marlet. "Virtual Line Descriptor and Semi-Local Matching Method for Reliable Feature Correspondence". In: *British Machine Vision Conference 2012*. United Kingdom, Sept. 2012, pp. 16.1–16.11. URL: https://hal.science/hal-00743323.

[9] Tiago Dantas. *Perigos do trânsito para a saúde*. Acessado em: 18 ago. 2021. Uol, 2013. URL: https://mundoeducacao.uol.com.br/saude-bem-estar/perigos-transito-para-saude.htm.

[10] São Paulo G1. *Congestionamento provoca estresse e pode ocasionar crises de ansiedade*. Acessado em: 18 ago. 2021. G1, 2013. URL: http://g1.globo.com/sao-paulo/anda-sp/noticia/2013/07/congestionamento-provoca-estresse-e-pode-ocasionar-crises-de-ansiedade.html.

[11] Jason Chang. *Sampling in computer vision and Bayesian nonparametric mixtures*. 2014.

[12] Raad Ahmed Hadi, Ghazali Sulong, and Loay Edwar George. "Vehicle Detection and Tracking Techniques : A Concise Review". In: *Signal &amp Image Processing : An International Journal* 5.1 (Feb. 2014), pp. 1–12. DOI: 10.5121/sipij.2014.5101. URL: https://doi.org/10.5121%2Fsipij.2014.5101.

[13] Laura Leal-Taixé. *Multiple object tracking with context awareness*. 2014. DOI: 10.48550/ARXIV.1411.7935. URL: https://arxiv.org/abs/1411.7935.

[14] CS231n Course Materials. *CS231n: Convolutional Neural Networks for Visual Recognition*. 2015. URL: https://cs231n.github.io/convolutional-networks/.

[15] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[16] Alex Bewley et al. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, Sept. 2016. DOI: 10.1109/icip.2016.7533003. URL: https://doi.org/10.1109%5C%2Ficip.2016.7533003.

[17] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *ArXiv e-prints* (Mar. 2016). eprint: 1603.07285.

[18] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. 2016. URL: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[19] Smarter Cambridge Transport. *Reducing Traffic Congestion and Pollution in Urban Areas*. Acessado em: 18 ago. 2021. smartertransport, 2016. URL: https://www.smartertransport.uk/smarter-cambridge-transport-urban-congestion-enquiry/.

[20] Dan Becker. *Rectified Linear Units (ReLU) in Deep Learning*. 2017. URL: https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning/notebook.

[21] Rodrigo De Oliveira et al. "A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges". In: *Journal of Microwaves, Optoelectronics and Electromagnetic Applications* 16 (Sept. 2017), pp. 628–645. DOI: 10.1590/2179-10742017v16i3854.

[22] Google. *Colaboratory Frequently Asked Questions*. 2017. URL: https://research.google.com/colaboratory/faq.html.

[23] Priya Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". In: *CoRR* abs/1706.02677 (2017). arXiv: 1706.02677. URL: http://arxiv.org/abs/1706.02677.

[24] Shivlu Jain. *When To Use Softmax Activation Algorithm In Deep Learning*. 2017. URL: http://www.mplsvpn.info/2017/12/when-to-use-softmax-activation.html.

[25] Jaydip Sen et al. "Cloud Computing - Architecture and Applications". In: *CoRR* abs/1707.09488 (2017). arXiv: 1707.09488. URL: http://arxiv.org/abs/1707.09488.

[26] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *CoRR* abs/1703.07402 (2017). arXiv: 1703.07402. URL: http://arxiv.org/abs/1703.07402.

[27] Jonathan Hui. *mAP (mean Average Precision) for Object Detection*. 2018. URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

[28] Lars Hulstaert. *A Beginner's Guide to Object Detection*. 2018. URL: https://www.datacamp.com/tutorial/object-detection-guide.

[29] Jeremy Jordan. *An overview of object detection: one-stage methods*. 2018. URL: https://www.jeremyjordan.me/object-detection-one-stage/.

[30] Rayson Laroca et al. "A Robust Real-Time Automatic License Plate Recognition based on the YOLO Detector". In: *CoRR* abs/1802.09567 (2018). arXiv: 1802.09567. URL: http://arxiv.org/abs/1802.09567.

[31] Aditya Mishra. *Metrics to Evaluate your Machine Learning Algorithm*. 2018. URL: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

[32] Erica Mixon. *Google Drive*. 2018. URL: https://www.techtarget.com/searchmobilecomputing/definition/Google-Drive.

[33] Julian Müller and Klaus Dietmayer. "Detecting Traffic Lights by Single Shot Detection". In: *CoRR* abs/1805.02523 (2018). arXiv: 1805.02523. URL: http://arxiv.org/abs/1805.02523.

[34] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[35] Java T Point Team. *Performance Metrics in Machine Learning*. 2018. URL: https://www.javatpoint.com/performance-metrics-in-machine-learning.

[36] SuperDataScience Team. *The Ultimate Guide to Convolutional Neural Networks (CNN)*. 2018. URL: https://www.superdatascience.com/blogs/the-ultimate-guide-to-convolutional-neural-networks-cnn.

[37] Zhong-Qiu Zhao et al. "Object Detection with Deep Learning: A Review". In: *CoRR* abs/1807.05511 (2018). arXiv: 1807.05511. URL: http://arxiv.org/abs/1807.05511.

[38] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. 2019. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[39] Lucas C. Possatti et al. "Traffic Light Recognition Using Deep Learning and Prior Maps for Autonomous Cars". In: *CoRR* abs/1906.11886 (2019). arXiv: 1906.11886. URL: http://arxiv.org/abs/1906.11886.

[40] Tavish Srivastava. *11 Important Model Evaluation Metrics for Machine Learning Everyone should know*. 2019. URL: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/.

[41] Tina Bellon. *As global traffic jams mount, cities try new ways to ease congestion: study*. 2020. URL: https://www.reuters.com/article/us-autos-congestion-idUSKBN20W18E.

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

[43] Jason Brownlee. *Softmax Activation Function with Python*. 2020. URL: https://machinelearningmastery.com/softmax-activation-function-with-python/.

[44] IBM Cloud Education. *Convolutional Neural Networks*. 2020. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks#toc-types-of-c-yL2bT7qZ.

[45] IBM Cloud Education. *Neural Networks*. 2020. URL: https://www.ibm.com/cloud/learn/neural-networks.

[46] Redação Garagem360. *Acidentes de trânsito aumentam quando há congestionamento*. Acessado em: 18 ago. 2021. Garagem360, 2020. URL: https://garagem360.com.br/acidente-de-transito/.

[47] Amin Keshavarzi, Abolfazl Toroghi Haghighat, and Mahdi Bohlouli. "Research Challenges and Prospective Business Impacts of Cloud Computing: A Survey". In: *CoRR* abs/2005.01475 (2020). arXiv: 2005.01475. URL: https://arxiv.org/abs/2005.01475.

[48] Jhon Majin. *DESENVOLVIMENTO DE UM SISTEMA DE CONTAGEM E CLASSIFICAÇÃO DE VEÍCULOS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS*. Universidade Federal de Santa Catarina, 2020.

[49] Vishal Mandal and Yaw Adu-Gyamfi. "Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis". In: *CoRR* abs/2007.16198 (2020). arXiv: 2007.16198. URL: https://arxiv.org/abs/2007.16198.

[50] Shriyashish Mishra. *Object Detection*. 2020. URL: https://github.com/shriyashish/objectdetection.

[51] Shriyashish Mishra. *Object Localization*. 2020. URL: https://github.com/shriyashish/objectdetection.

[52] Tutorials Point. *Google Colab Tutorial*. 2020. URL: https://www.tutorialspoint.com/google_colab/google_colab_introduction.htm.

[53] Bala Priya. *Softmax Activation Function: Everything You Need to Know*. 2020. URL: https://www.pinecone.io/learn/softmax-activation/.

[54] Sistemas Inteligentes de Trasporte Brasil. *A ITSB*. 2020. URL: https://www.itsb.org.br/a-itsb.

[55] Shreejal Trivedi. *YOLOv4 — Version 0: Introduction*. 2020. URL: https://medium.com/visionwizard/yolov4-version-0-introduction-90514b413ccf.

[56] Shivy Yohanandan. *mAP (mean Average Precision) might confuse you!* 2020. URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2.

[57] Deep AI. *Neural Network*. 2021. URL: https://deepai.org/machine-learning-glossary-and-terms/neural-network.

[58] Ed Burns. *What is a neural network? Explanation and examples*. 2021. URL: https://www.techtarget.com/searchenterpriseai/definition/neural-network.

[59] Leiyu Chen et al. "Review of Image Classification Algorithms Based on Convolutional Neural Networks". In: *Remote Sensing* 13.22 (2021). DOI: 10.3390/rs13224712. URL: https://www.mdpi.com/2072-4292/13/22/4712.

[60] Guilherme Fontana. *Produção de veículos no Brasil cai 31,6% em 2020 e tem pior resultado desde 2003*. Acessado em: 17 ago. 2021. G1, 2021. URL: https://g1.globo.com/economia/noticia/2021/01/08/producao-de-veiculos-no-brasil-cai-316percent-em-2020-diz-anfavea.ghtml.

[61] Bernardo Gonzaga. *Produção de veículos cresce 57,5% no 1º semestre...* Acessado em: 17 ago. 2021. poder360, 2021. URL: https://www.poder360.com.br/economia/producao-de-veiculos-cresce-575-no-1o-semestre/.

[62] Nizar Hamadeh et al. *Intelligent Transportation Systems to Mitigate Road Traffic Congestion*. 2021. DOI: 10.48550/ARXIV.2106.02315. URL: https://arxiv.org/abs/2106.02315.

[63] Yash Khandelwal. *Crowd Counting using Deep Learning*. 2021. URL: https://www.analyticsvidhya.com/blog/2021/06/crowd-counting-using-deep-learning/#:~:text=It%5C%20is%5C%20one%5C%20of%5C%20the,counting%5C%2C%5C%20and%5C%20traffic%5C%20flow%5C%20monitoring.

[64] Hui Lin, Xiaopeng Hong, and Yabin Wang. "Object Counting: You Only Need to Look at One". In: *CoRR* abs/2112.05993 (2021). arXiv: 2112.05993. URL: https://arxiv.org/abs/2112.05993.

[65] Zebin Lin et al. "Pedestrian Detection by Exemplar-Guided Contrastive Learning". In: *CoRR* abs/2111.08974 (2021). arXiv: 2111.08974. URL: https://arxiv.org/abs/2111.08974.

[66] Addie Ira Parico and Tofael Ahamed. "Real Time Pear Fruit Detection and Counting Using YOLOv4 Models and Deep SORT". In: *Sensors* 21 (July 2021), p. 4803. DOI: 10.3390/s21144803.

[67] AWS Team. *What is a Neural Network?* 2021. URL: https://aws.amazon.com/what-is/neural-network/.

[68] Fritzs Team. *Object Detection Guide: Almost everything you need to know about how object detection works*. 2021. URL: https://www.fritz.ai/object-detection/.

[69] TomTom. *TomTom Traffic Index Ranking 2021*. 2021. URL: https://www.tomtom.com/en_gb/traffic-index/ranking/.

[70] Grupo de Tratamiento de Imágenes. "A Novel System for Nighttime Vehicle Detection Based on Foveal Classifiers with Real-Time Performance". In: *IEEE Trans. Intelligent Transportation Systems* xx.x (2021), pp. xxx–xxx. DOI: https://doi.org/10.1109/TITS.2021.3053863.

[71] Grupo de Tratamiento de Imágenes. *Nighttime Vehicle Detection database (NVD)*. 2021. URL: https://www.gti.ssr.upm.es/data/NVD_database.

[72] Wen-Kai Tsai and Hung-Ju Chen. "High-accuracy vehicle lamp detection for real-time night-time traffic surveillance". In: *IET Intelligent Transport Systems* 14 (Jan. 2021). DOI: 10.1049/iet-its.2020.0063.

[73] Aayush Bajaj. *Performance Metrics in Machine Learning [Complete Guide]*. 2022. URL: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide.

[74] Abhishek Balasubramaniam and Sudeep Pasricha. "Object Detection in Autonomous Vehicles: Status and Open Challenges". In: *CoRR* abs/2201.07706 (2022). arXiv: 2201.07706. URL: https://arxiv.org/abs/2201.07706.

[75] Nilesh Barla. *The Complete Guide to Object Tracking [+V7 Tutorial]*. 2022. URL: https://www.v7labs.com/blog/object-tracking-guide.

[76] Gaudenz Boesch. *Object Detection in 2022: The Definitive Guide*. 2022. URL: https://viso.ai/deep-learning/object-detection/.

[77] Best Selling Cars. *2021 (Full Year) International: Worldwide Car Sales*. 2022. URL: https://www.best-selling-cars.com/international/2021-full-year-international-worldwide-car-sales/.

[78] James Chen. *What Is a Neural Network?* 2022. URL: https://www.investopedia.com/terms/n/neuralnetwork.asp.

[79] Yunhao Du et al. *StrongSORT: Make DeepSORT Great Again*. 2022. DOI: 10.48550/ARXIV.2202.13514. URL: https://arxiv.org/abs/2202.13514.

[80] Google. *Google Drive training and help*. 2022. URL: https://support.google.com/a/users/answer/9282958?hl=en.

[81] Chirag Goyal. *20 Questions to Test your Skills on CNN (Convolutional Neural Networks)*. 2022. URL: https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-cnn-convolutional-neural-networks/.

[82] Viso AI Group. *Object counting*. 2022. URL: https://viso.ai/application/object-counting/.

[83] Tao Han et al. *DR.VIC: Decomposition and Reasoning for Video Individual Counting*. 2022. DOI: 10.48550/ARXIV.2203.12335. URL: https://arxiv.org/abs/2203.12335.

[84] Irtiza Hasan et al. "Pedestrian Detection: Domain Generalization, CNNs, Transformers and Beyond". In: *CoRR* abs/2201.03176 (2022). arXiv: 2201.03176. URL: https://arxiv.org/abs/2201.03176.

[85] Nico Klingler. *Object Tracking in Computer Vision (Complete Guide)*. 2022. URL: https://viso.ai/deep-learning/object-tracking/.

[86] Kukil. *Intersection over Union (IoU) in Object Detection and Segmentation*. 2022. URL: https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/.

[87] Kukil. *Mean Average Precision (mAP) in Object Detection*. 2022. URL: https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/.

[88] Bharat Mahaur and K. Mishra. "Road object detection: a comparative study of deep learning-based algorithms". In: *Multimedia Tools and Applications* 81 (Apr. 2022). DOI: 10.1007/s11042-022-12447-5.

[89] Deval Shah. *Mean Average Precision (mAP) Explained: Everything You Need to Know*. 2022. URL: https://www.v7labs.com/blog/mean-average-precision.

[90] AltexSoft Team. *Machine Learning Metrics: How to Measure the Performance of a Machine Learning Model*. 2022. URL: https://www.altexsoft.com/blog/machine-learning-metrics/.

[91] Exploring Our Fluid Earth Team. *Practices of Science: False Positives and False Negatives*. 2022. URL: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide.

[92] Hasty Team. *Intersection over Union (IoU)*. 2022. URL: https://hasty.ai/docs/mp-wiki/metrics/iou-intersection-over-union.

[93] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. DOI: 10.48550/ARXIV.2207.02696. URL: https://arxiv.org/abs/2207.02696.