Object detection, distributed cloud computing and parallelization techniques for autonomous driving systems.

MEDINA, E.C.G., ESPITIA, V.M.V., SILVA, D.C., DE LAS CUEVAS, S.F.R., HIRATA, M.P., CHEN, A.Z., GONZÁLEZ, J.A.G., BUSTAMANTE-BELLO, R. and MORENO-GARCÍA, C.F.

2021

applied sciences

Review

Object Detection, Distributed Cloud Computing and Parallelization Techniques for Autonomous Driving Systems

Edgar Cortés Gallardo Medina [1], Victor Miguel Velazquez Espitia [1], Daniela Chípuli Silva [1], Sebastián Fernández Ruiz de las Cuevas [1,2], Marco Palacios Hirata [1], Alfredo Zhu Chen [1], José Ángel González González [1], Rogelio Bustamante-Bello [1] and Carlos Francisco Moreno-García [3,*]

[1] School of Engineering and Sciences, Tecnologico de Monterrey, Calle Puente 222, Tlalpan, Mexico City 14380, Mexico; A01336292@itesm.mx (E.C.G.M.); A01339942@itesm.mx (V.M.V.E.); A01652237@itesm.mx (D.C.S.); A01652293@itesm.mx (S.F.R.d.l.C.); A01650354@itesm.mx (M.P.H.); A01651980@itesm.mx (A.Z.C.); A01652551@itesm.mx (J.Á.G.G.); rbustama@tec.mx (R.B.-B.)
[2] Politecnico Di Torino, Department of Mechanical and Aerospace Engineering, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
[3] School of Computing, Robert Gordon University, Garthdee Road, Aberdeen AB10 7QB, Scotland, UK
* Correspondence: c.moreno-garcia@rgu.ac.uk

Featured Application: This paper contains a comprehensive review of the current state-of-the-art machine vision algorithms used to build a pipeline towards assisted driving and fully autonomous vehicle development. Key and novel concepts such as distributed systems and parallelization are explored to enhance said systems' functionality.

Abstract: Autonomous vehicles are increasingly becoming a necessary trend towards building the smart cities of the future. Numerous proposals have been presented in recent years to tackle particular aspects of the working pipeline towards creating a functional end-to-end system, such as object detection, tracking, path planning, sentiment or intent detection, amongst others. Nevertheless, few efforts have been made to systematically compile all of these systems into a single proposal that also considers the real challenges these systems will have on the road, such as real-time computation, hardware capabilities, etc. This paper reviews the latest techniques towards creating our own end-to-end autonomous vehicle system, considering the state-of-the-art methods on object detection, and the possible incorporation of distributed systems and parallelization to deploy these methods. Our findings show that while techniques such as convolutional neural networks, recurrent neural networks, and long short-term memory can effectively handle the initial detection and path planning tasks, more efforts are required to implement cloud computing to reduce the computational time that these methods demand. Additionally, we have mapped different strategies to handle the parallelization task, both within and between the networks.

Keywords: autonomous vehicle; autonomous driving system; computer vision; neural networks; feature extraction; segmentation; assisted driving; cloud computing; parallelization

check for updates

Citation: Cortés Gallardo Medina, E.; Velazquez Espitia, V.M.; Chípuli Silva, D.; Fernández Ruiz de las Cuevas, S.; Palacios Hirata, M.; Zhu Chen, A.; González González, J.Á.; Bustamante-Bello, R.; Moreno-García, C.F. Object Detection, Distributed Cloud Computing and Parallelization Techniques for Autonomous Driving Systems. *Appl. Sci.* **2021**, *11*, 2925. https://doi.org/10.3390/app11072925

Academic Editor: Manuel Armada

Received: 30 January 2021
Accepted: 18 March 2021
Published: 25 March 2021

1. Introduction

The National Highway Traffic Safety Administration (NHTSA) reported that 94% of severe crashes on the road are caused by human error [1]. In this regard, the rise of the autonomous vehicle (AV) has a huge potential to decrease these accidents and make the road much safer. Therefore, the implementation of robust and secure systems is paramount for the proper design of an autonomous driving system (ADS) pipeline. This field has been widely investigated for many years. Both academia and industry investigations have achieved breakthroughs and state-of-the-art results in the last years. In 2004, the Defense Advanced Research Projects Agency (DARPA) held the Grand Challenge competition for American AVs (DARPA, *The Grand Challenge*, DARPA, https://www.darpa.mil/about-us/

timeline/-grand-challenge-for-autonomous-vehicles); however, no team could complete the designated 150 miles route. One year later, at the same event, all but one of the finalists surpassed the best record (7.32 miles) from the previous year, and five teams were able to finish the 132 miles desert terrain route without human intervention. In 2007, during the third edition of this challenge (DARPA, *DARPA Urban Challenge*, DARPA, https://www.darpa.mil/about-us/timeline/darpa-urban-challenge), 60 miles of the urban course were proposed. Only six teams managed to complete the entire track successfully. These three competitions were very significant events up to this time, so the results have inspired universities and large corporations to improve the state of the art of in ADSs in different environments and conditions.

Apart from the DARPA challenges, many other remarkable competitions have been held until today. For instance, Hyundai Motor Company has been researching AVs for a long time. They aim to expand future vehicle research at universities and improve Korea's automotive industry's technological development, and thus they have organized four AVs competitions (Hyundai NGV Industry-Academy-Research Cooperation, *Autonomous Vehicle Competition*, Hyundai NGV, http://www.hyundai-ngv.com/en/tech_coop/sub05.do) since 2010. Similarly, the Society of Autonomous Engineers (SAE) and General Motors have partnered to sponsor the AutoDrive Challenge (SAE International and GM, *AutoDrive Challenge*, SAE, https://www.sae.org/attend/student-events/autodrive-challenge/), which consisted of a three-year (2018–2020) competition for students to complete an urban driving course. Furthermore, Indy Autonomous Challenge (Indianapolis Motor Speedway, *Indy Autonomous Challenge*, Indianapolis Motor Speedway, https://www.indyautonomouschallenge.com/) is a new high-speed autonomous race held at the Indianapolis Motor Speedway. Every team must use the specific Dallara-produced IL-15 that has been retrofitted with hardware and controls to compete. It has been noted that numerous universities and companies worldwide are very active in these kinds of events, which indubitably will accelerate research on this field.

From the academic perspective, several survey papers in this area have taken different approaches to review this field. Rangesh et al. [2] compared several representative online Multi-Object Tracking (MOT) for AVs. Later on, the authors proposed M^3OT (i.e., a 3D variation of MOT), an approach suitable for tracking objects in the real world. It is worth mentioning that M^3OT can work with any type and number of cameras as long as they are calibrated. Huang et al. [3] spanned the state-of-the-art technology in significant fields of self-driving systems, such as perception, mapping and localization, prediction, planning and control, simulation, etc. Yurtsever et al. [4] attempt to provide a structured and comprehensive overview of state-of-the-art automated driving-related hardware-software practices, high-level systems architectures, present challenges, datasets, and tools to ADS.

Badue et al. [5] published survey research on self-driving cars based on the aforementioned DARPA challenges. The authors presented a detailed description of the self-driving vehicle developed at the Universidade Federal do Espírito Santo in Brazil, named Intelligent Autonomous Robotics Automobile (IARA). This paper presents a complete overview of each of the aspects that compose the matured used architecture of self-driving cars nowadays, dividing it into two systems, i.e., perception and decision-making. At the same time, each design is composed of a variety of subsystems. Achieving its objective by describing each of the technologies superficially, this work represents a notable introduction to the interested reader. However, for a lettered person in the subject, this work's scope might fall short because it does not present the state-of-the-art technologies for driverless cars' different paradigms.

Furthermore, the study of these techniques has been extended to other application domains beyond AVs on the highway. For instance, Szymak et al. [6] presented a comparative study where different deep learning architectures were tested to perform object classification in underwater AVs' video image processing. In this study, the authors evaluated different architectures' performance to classify objects (i.e., fish, other vehicles, divers and obstacles) and detect abandoned munitions' corrosion. Interestingly, they were able to deduct that pre-trained algorithms have higher probabilities of success than tailor-made approaches. This rationale will be explained and supported throughout our work.

This paper provides a thorough state-of-the-art investigation on object detection methods for the ADS pipeline. In addition, distributed cloud computing and parallelization techniques are studied to understand how these can be incorporated to deploy these methods. Unlike other reviews, this study is centralized mainly in autonomous driving perception systems, focusing on the neural network (NN) concepts for object detection, and how the concept of a Distributed System (DS) for parallelization can be included within. The paper is structured as follows. Section 2 provides the basic definitions and background required to understand the domain, with concepts such as image acquisition, feature extraction, object detection, DS, and route calculation. Section 3 presents a thorough study of each of these concepts in light of the latest work presented in this regard. Section 4 introduces our proposed methodology to address the most recurrent issues of assisted driving and AVs based on our study and with considerable focus on parallelization for real-time deployment systems. Finally, Section 5 provides our final thoughts and comments on the subject.

2. Background

In this section, the object detection task's building blocks are explained along with their corresponding evaluation metrics and applications (i.e., path planning using the Frenét framework). This section describes feature extraction; the types and methods used to discern distinctive characteristics from the data provided by various sensors. Moreover, it will cover the discussion of multiple NN architectures and the concatenation of several deep learning techniques, which will be used for the Frenét framework generation for path tracking of vehicles on the road. DS architectures, whose components are located in different networks, are also explored in deep learning systems. These can provide an advantage when training and running computer vision architectures.

2.1. Feature Extraction Methods used for Object Detection in AVs

Feature extraction consists of gathering distinct characteristics to reduce the data needed to describe important elements. The object detection task's performance is determined not only by the learning algorithm responsible for most of the job at hand but also by the methods that select and combine sensory inputs into discernible features. The raw sensor data itself contains structural information extracted by the learning algorithm itself (such as the Convolutional Neural Network). However, there is usually a process undertaken to identify, extract and assign features so that the model learns only the most relevant characteristics. This work discusses the most common feature extraction approaches in machine learning for multiple object detection, such as camera-based, Light Detection and Ranging (LiDAR) based, and a hybrid method that uses both.

2.1.1. Camera-Based

Most standard commercial cameras can capture rich color and texture information from images used to locate objects and obstacles along the course. An image typically has a representation of three channels. On the one hand, attempting only to use these components' values is defined as a pixel feature approach; this means that the machine learning algorithm will obtain and learn the raw data features by itself. On the other hand, crafted feature representation consisting of image processing techniques to extract the features before the learning process, such as edge detection, corner detection, transformations, smoothing, or blurring (among others), can be extracted before the learning process. This process is known as an *extracted feature approach*. The latter follows a series of traditional image pre-processing methods to extract a series of key points and their respective feature descriptors. For instance, Scale Invariant Feature Transform (SIFT) [7] is one of the most widely used methods for such purpose because of its outstanding capabilities of extracting pixel colors, intensities, position, and Gaussian properties such as the Difference of Gaussian (DoG) pyramid and edge key points. All of this information is extracted while preserving scale invariance, as its name suggests. Although this is a suitable feature extractor for a camera's raw images, it is very slow in implementation and unusable in a real-time scenario.

In contrast, the Speeded-Up Robust Features (SURF) [8] method is one of the fastest and most robust feature extractors. It behaves similar to SIFT, but its outstanding accuracy in extracting important key points between matching images, its real-time performance capability, and the possibility for parallelization with GPU architectures using CUDA (a parallel computing platform developed by NVIDIA using C and C++ libraries) puts this feature extractor at the forefront [9]. SURF uses Hessian matrices to detect blobs and to find critical key points. Then, these are used to measure the focal length with each neighboring point in the image. Afterward, the sum of Haar cascades is used to obtain rotational and translational invariance. Finally, its descriptors are compared to those on another image picturing the same object but from a different angle.

2.1.2. LiDAR-Based

LiDAR is one of the most widely used sensors in robotics [10]. The classical 2D LiDAR uses a rotating mirror, which sends laser beams along the x-y plane, sensing objects within range. On the other hand, a 3D LiDAR also shoots beams along the x-z plane, capturing the distance to items along this plane. Figure 1 shows an example of this setting.

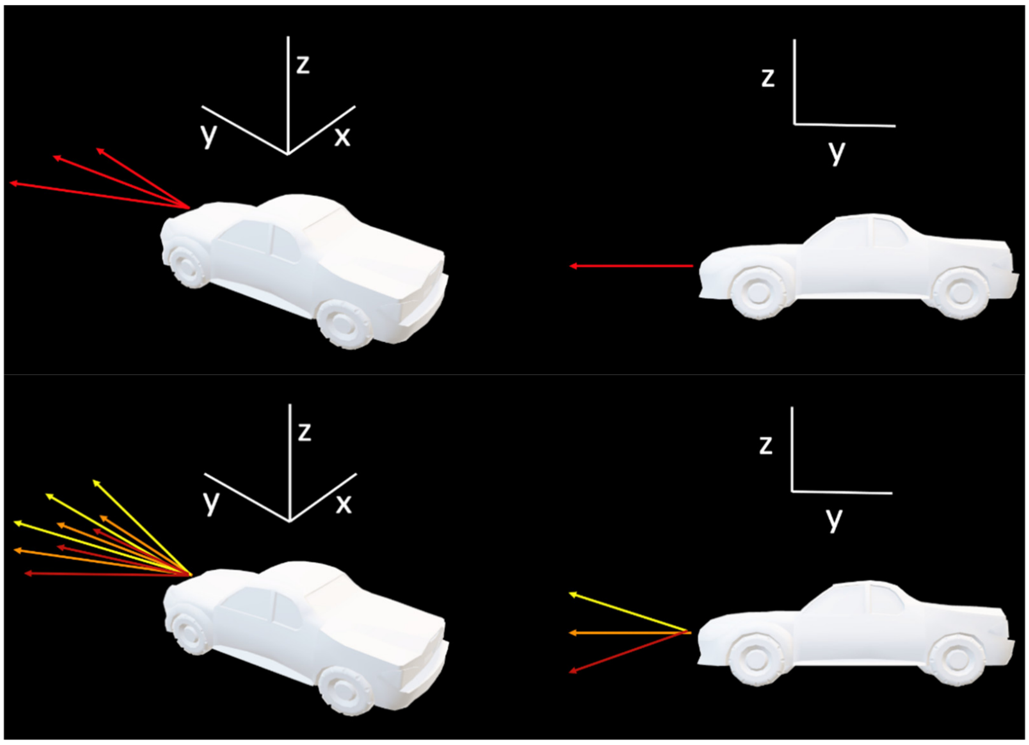

Figure 1. Diagram of the differences between the data gathered from 2D (**top**) and 3D LiDAR (**bottom**).

The features extracted from a LiDAR sensor are the so-called point clouds. Point clouds represent the surfaces in which the LiDAR laser bounced before reaching the LiDAR sensor back. These points follow a specific array of algorithms to be scale, position, and orientation invariant; all of this to perform Simultaneous Localization and Mapping (SLAM). LiDAR sensors can make a single point cloud swift in an outstanding short amount of time, which means that feature extraction is performed with a comparable speed to classical feature extraction. When it comes to object detection, a LiDAR sensor can also be used to aid AVs. BirdNet [11], for example, can detect 3D obstacles from the LiDAR's point cloud using three steps: first, a Bird's Eye View (BEV) image is created from the raw sensor data. Then, using CNNs intended initially for image processing, the position and orientation of obstacles are estimated. Finally, for the post-processing phase, predictions are corrected using statistical data, and the height of the barriers is projected on a predicted plane. This method was tested using the KITTI data set to show a state-of- the-art performance. Similar approaches include the Multi-View 3D Object Detection Network for Autonomous Driving (MV3D) [12]. MV3D fuses the camera and the LiDAR features to obtain a more rigorous representation, although the results of both methods show the state-of-the-art performance. When it comes to LiDARs in AVs, 3D LiDARs [13] are used for various reasons. First, this is a more robust sensor that includes more laser scans along the z-axis so a more extensive range of obstacles can be detected, whereas a 2D LiDAR is bounded to scanning a fixed-line along this axis. Second, the 3D point cloud produced by a 3D LiDAR can easily be transformed into the camera's coordinate system [14]. Thus, the object detection task can be performed by the camera and aided by the LiDAR, as explored in the following section.

2.1.3. Hybrid Approaches

Several approaches intend to combine multiple modalities of features to provide complementary information and leverage the classification performance. The single modality of the features is not enough to solve some scenarios, such as self-driving, which involves volatile strategies and requires the utmost precision for safety reasons. In these cases, extracting features from different sensors and fusing them might improve obstacle detection effectiveness.

When it comes to self-driving cars [4], one common approach is to use the data of both cameras and LiDARs. Combining the cameras' detail to achieve reliable object detection with the laser scan reading provided by LiDAR allows us to identify obstacles and place them in the world. As discussed in Section 2.1.2, a 3D LiDAR enables the fusion of a 3D point cloud and a 2D image, combining image detail and object positioning, as shown in Figure 2.

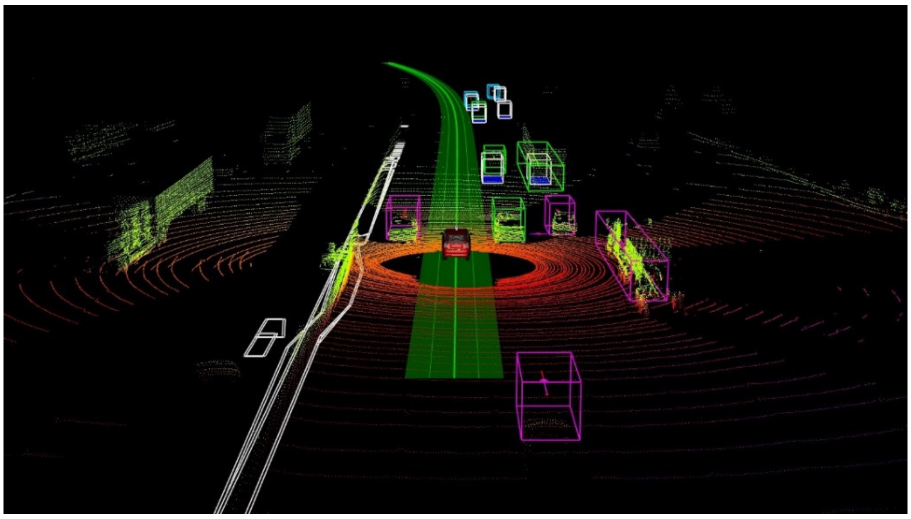

Figure 2. LiDAR point cloud projection (image under creative commons license).

To validate this, tests were carried out inside a gazebo simulator. The point cloud from a 2D LiDAR (mounted on a 2-wheel robot) was projected onto an image produced by a camera on the same robot. Data were projected from one sensor onto another, thus proving this method to be a low computation and low-cost procedure to accurately detect the distance to objects identified by computer vision algorithms. However, for the robustness needed in ADSs, the 3D LiDAR is still the more reliable option as the 2D LiDAR is bounded to a single line along the x-y plane.

It is essential to point out that using a LiDAR sensor is not the only way to obtain point clouds. Using a stereo camera and performing SIFT-like algorithms, point clouds can also be generated for autonomous driving applications. The only issue is the frame rate, which is not as good as when using a LiDAR swift. Moreover, this alternative would require image pre-processing to acquire the point clouds properly. Nonetheless, this alternative is much less expensive, and even some GPU-based architectures of this kind have shown a very competitive point cloud generation compared to LiDAR sensor-based architectures [15].

2.2. Object Detection

Object detection is the action of recognizing and locating every single object within an image. In contrast, image classification detects all items in an image but only displays those with the highest probability of being the one of interest. Therefore, object classification is commonly used for classification tasks, while object detection is demanded in AV scenarios.

2.2.1. Convolutional Neural Network (CNN)

CNN is one of the most widely used image classification methods, object detection, and semantic image segmentation because of its robust feature learning and classification capabilities [16,17]. Their primary purpose is to reduce the images into a more straightforward process without losing features relevant for the right prediction. In other words, the objective is to extract the high-level features from the input image.

A typical CNN consists of an input layer, a convolutional layer, a pooling layer, a fully connected layer, and an output layer. The convolutional layer contains feature detectors, best known as kernels or filters, followed by a normalization layer and a non-linear activation function to extract feature maps of the input image. These feature detectors swiftly through the input image and filter the critical parts to scan images and categorize them more accurately. These feature maps are then down-sampled by pooling layers, removing redundant features, and reducing their dimensions from their previous layers to improve the statistical efficiency and model generalization. Finally, fully connected layers are applied to classify the extracted features and obtain an input-based probability distribution. After that, the network's output is a fixed-sized vector [18,19]. Figure 3 shows the classical CNN architecture example.

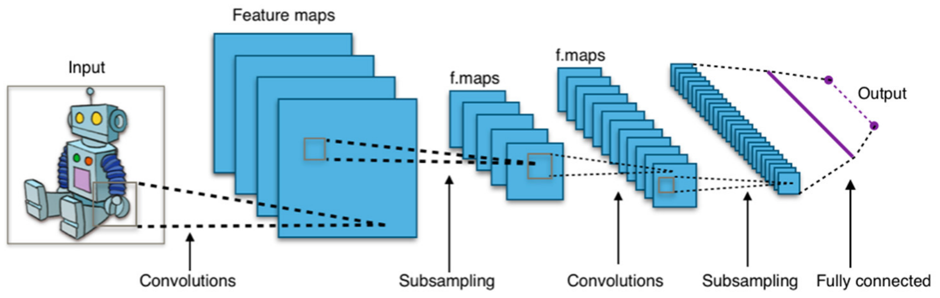

Figure 3. Convolutional Neural Network (CNN) architecture with an input, convolutional, subsample, fully connected, and output layers (image under creative commons license).

In terms of accuracy, the CNN is superior to traditional image classification methods, such as Directed Acyclic Graph-Support Vector Machine (DAG-SVM) [20] or Constrained Linear Discriminant Analysis (CLDA) [21], and are the most developed networks for image classification tasks nowadays [17,18]. The structure of early CNNs was relatively simple, such as the classic LeNet-5 model [22] (shown in Figure 4), which was mainly used in handwritten character recognition and image classification. However, with the continuous deepening of research and the need for more complex and extended applications, CNN's structure is continuously optimized. For example, the Convolutional Deep Belief Network (CDBN) [23] is an unsupervised generative model resulting from combining a CNN and a Deep Belief Network (DBN). Successful applications in the field of object detection (discussed in detail in the following section) were presented in [24] with the intent of performing region feature extraction based on a CNN.

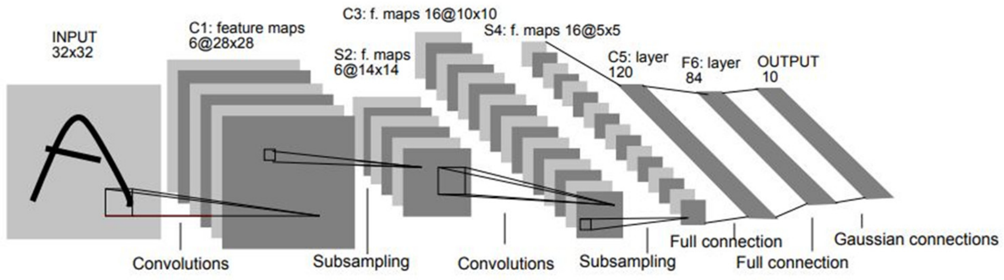

Figure 4. LeNet-5 architecture, which contains five first-level features (image under creative commons license).

2.2.2. R-CNN

The Region-Based Convolutional Neural Network (R-CNN) model is an object detection architecture. This model allows selective search, which is based on proposing ≈2000 regions per image, which are then passed through the convolutional layers for the classification of the areas and features differentiation, finally output the boundary box and the label of the object's type. [25] This method's principal disadvantage is that it is very slow, taking around 50 s for object detection. Another consideration is that it is memory-consuming. Additionally, the authors considered a supervised pre-training stage for the convolutional network, as shown in Figure 5.

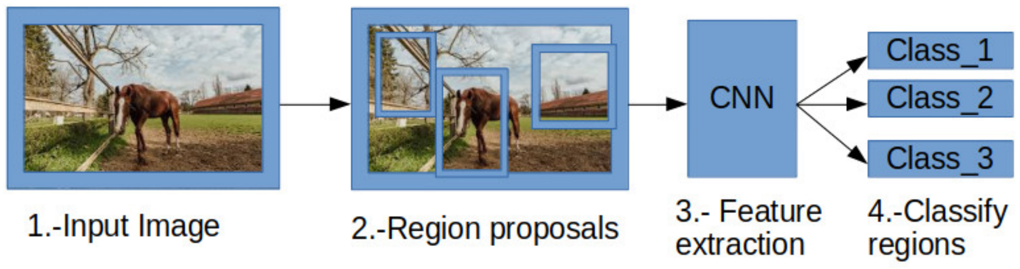

Figure 5. Basic Region-Based Convolutional Neural Network (R-CNN) algorithm.

R-CNN architecture appeared as an innovative method, combining region proposal and convolutional networks approaches to produce the bounding boxes. This result is a promising approach if disadvantages could be solved, thus developments in the study have been done generating improved versions of this architecture. The following subsections are devoted to discussing the most popular R-CNN-based methodologies presented in the literature that solved the mentioned issues. Moreover, architectures similar in purpose but different in nature will be discussed, such as the renowned You Only Look Once (YOLO) architecture.

2.2.3. Fast R-CNN

Similar to the standard R-CNN, this method inputs regions of an image into the convolutional layers but at the same time inputs the whole image. Besides, it combines different architectures to process the data, such as ConvNet (a sequence of convolutional layers to pre-process the input image), Region of Interest (RoI) pooling layer for region proposal and a classification layer. These architectures reside in the same principal, as shown in Figure 6.

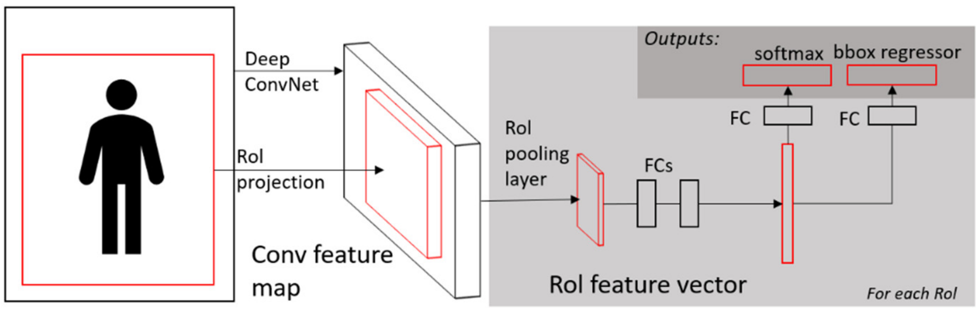

Figure 6. Fast R-CNN architecture.

This architecture dramatically improves the training time, over ten times the normal R-CNN, and achieves higher precision. Another contribution is that no storage is required for feature acquisition, reducing the memory needed. Although it drastically improves the time to about 2.5 s, it is still considerably slow for real-time applications; the selective search region proposal generation algorithm takes the most of the time. A newer architecture called Faster R-CNN attempted to overcome the problem mentioned above.

2.2.4. Faster R-CNN

With aims to real-time object detection, Faster R-CNN is a consolidated algorithm based on a Region Proposal Network (RPN). RPNs start by taking an image as input. As output, they provide a set of different rectangles of object proposals. In this architecture, the selective search algorithm is different with respect to the previous R-CNN architectures. RPNs share full-image convolutional features with the detection network. It is a fully convolutional network (FCN) that simultaneously predicts object bounds at each position. This network is trained in an end-to-end base to learn the task of generating detection proposals. First, it generates a convolution feature map, which then can be passed into the RPN. Finally, it is passed into a classification layer responsible for determining whether an object or not is present [25]. The mentioned process can be visualized in Figure 7.

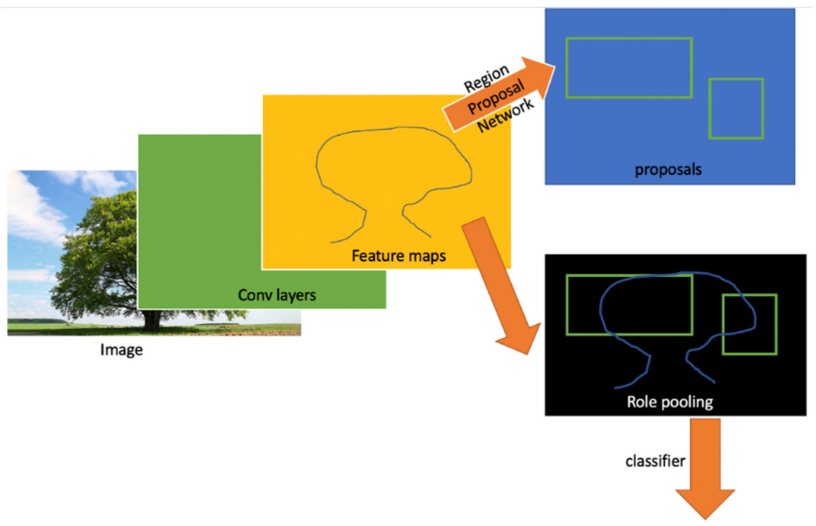

Figure 7. Faster R-CNN algorithm.

As the original paper's title proposes, this architecture's main objective was to get as close to a real-time object detection algorithm as possible. This architecture has achieved outstanding performance and time response (between 0.05 and 0.2 s). The proposal's accuracy increased critically in various applications, being its most significant advantage over other analog methods. It uses a reduced number of needed rectangle region proposals to 300, minimizing the computer processing need. One drawback compared to its most similar architecture (i.e., YOLO) is that all R-CNN architectures are two-stage algorithms, with YOLO being a single-stage one. Hence, YOLO is faster. YOLO is proposed as the faster approach in the literature, while Faster R-CNN is more accurate. The YOLO algorithm will be explained later in this same section.

2.2.5. Mask R-CNN

This method extends the Faster R-CNN concept by adding a branch for predicting an object segmentation mask in parallel with the bounding box prediction. In that case, the two units work for class prediction, both within an RoI. Segmentation is widely used for estimating poses, although it requires a much more refined spatial layout of an object. This method harnesses the velocity and, principally, Faster R-CNN's accuracy to get over the precision requirements and get over the known image segmentation approaches [25].

A mask encodes an input object spatial layout, and thus, a mask is predicted for each RoI using an FCN. This mask maintains the spatial layout without collapsing into vector representations. Instance segmentation is challenging since it requires correct object detection and precise segmentation to classify each pixel into a fixed set of categories. Recalling, this method adds a segmentation mask on each RoI. The added branch for segmentation comprises a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. This scenario only adds a small computational extra requirement, non-considerable compared to the needs of the standard R-CNN methods and other segmentation algorithms. The process is shown in Figure 8.

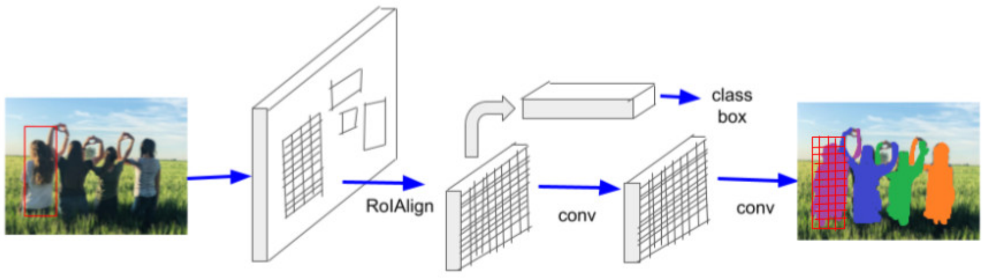

Figure 8. Mask RCNN architecture.

Furthermore, it is essential to remark that the authors proposed the RoI-Align layer, a quantization-free layer instead of RoI-Pooling that has misalignments between the RoI and extracted features. RoI-Pooling does first the same step for all RoI abstraction methods, dividing each coordinate by K, from the new relative coordinates the RoI is cropped. Finally, with quantization, the cut part is divided into bins that give an $n \times n$ RoI. RoI-Align presents a standard operation to extract small feature maps, properly aligning with the input to show the per-pixel spatial correspondence by defining the RoI with four points from the feature's division coordinates map is then bilinear interpolated to result in the final RoI, as shown in Figure 9. This process results are critical for the mask prediction, and has an enormous impact, improving the mask accuracy between 10–50%.

The model outperformed the baseline of other state-of-the-art instance segmentation models. The authors reported the inference time of architecture, capable of running at 195 ms per image on Nvidia Tesla M40 (plus 15ms CPU by resizing outputs). This method has been satisfactorily used in different applications. However, it presents a slightly higher accuracy than other mature architectures; there is a dilemma because faster architectures are capable of real-time working, such as YOLO (explained in the next section), which shows an adequate working velocity.

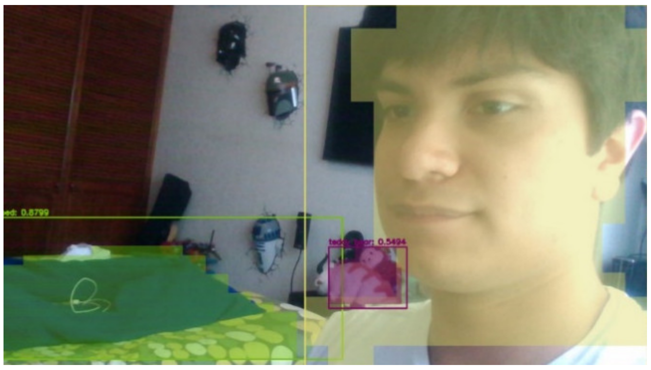

Figure 9. Segmentation results of Mask R-CNN.

2.2.6. You Only Look Once (YOLO)

YOLO deserves an analysis of its own, as it happens to be one of the most used architectures to perform recognition tasks. YOLO is an algorithm that uses a deep NN architecture output for making bounding boxes around all the objects that the algorithm recognizes in an image. The actual architecture used in YOLO is called DarkNet [26]. First, the image is divided into a grid, and in the last convolution layer of DarkNet, a sliding window analyzes each of the grid's dimensions at once. DarkNet uses a VGG-like [27] architecture with leaky ReLU activations in each layer, but in contrast to other architectures, it has no fully-connected layers at the end. Instead, it uses a $5 + C$ vector. That is, concatenated to a C channel vectors, being 5 the key components for detecting all objects in an image, i.e., object presence, objects localization in x and y coordinates, and objects dimensions, and C corresponding to the number of classes that DarkNet is trained on (20 in the original paper [27]). This vector is then convolved into a final net output with the width and height of the grid dimensions chosen for a specific image. The last dimension uses an arrangement, where B is the number of boxes per grid that the net is capable of handling. Once this last layer is done, then the YOLO algorithm works its task by sliding a window through each pixel of the final convolution layer, which then traduces into analyzing each of the cell grids of the original image at once, hence the name of the architecture. Once a simultaneous sliding window in each grid cell is produced, YOLO searches for an object that DarkNet can recognize, derived from the training data. In the original paper, each grid cell has two boxes, meaning that $B = 2$, giving a channel dimension of 30. These boxes exist because it could happen that two objects share a single grid cell, and so because there are two boxes, then two bounding boxes can be drawn and recognize both objects. B can increase so more objects are recognized per grid cell. In further YOLO applications B is commonly set to 5. This makes YOLO a very fast and accurate algorithm, being very suiting for real-time applications.

Despite this, YOLO does not cope well in very crowded scenarios. Given the grid approach, a minimum-maximum number of elements can be found in each grade depending on the setting. To address this issue, the last layer would be needed to be retrained with a finer grid and more boxes per grid cell, thus complicating the architecture, as not only YOLO would need to detect more classes, but also needs to be retrained to see more objects of the same classes it has already been trained for. It is important to state that YOLO has been experiencing several updates, which lead to improved versions. For instance, YOLO v4 is the best one, exceeding the 40 FPS in and NVIDIA GTX 1080 Ti GPU architecture, having a 65.7% mAP. It uses only three boxes for the B variable. Its training uses weighted-residual connections, cross mini-batch normalizations, and other techniques that make YOLO v4 very fast and accurate compared to different object detection algorithms.

2.2.7. RNN

A Recurrent Neural Network (RNN) is another type of architecture designed to model time series. A multilayer perceptron has the distinction of allowing connections among the hidden layers to describe a time delay. This architecture is known as the short-term memory of the NN. Because of the kind of relationships mentioned before, the model can retain information about past states, allowing it to understand temporal correlations between events distant in time to each other [28]. A basic example is depicted in Figure 10.

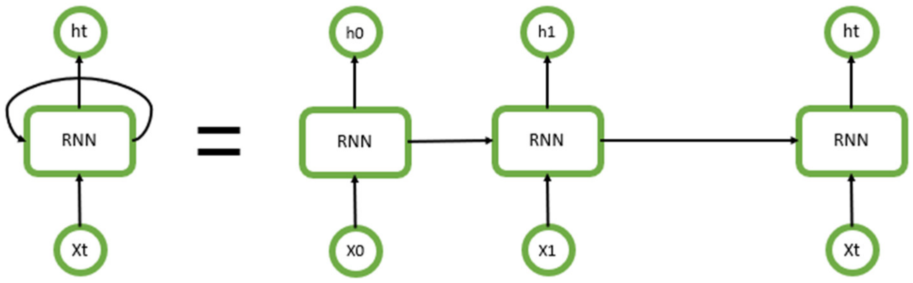

Figure 10. Recurrent Neural Network (RNN) Architecture.

In principle, RNN presents a robust and straightforward solution to predicting possible future scenarios based on previous events allowing information to persist. Its initial architecture presents two main problems at the update stage of backpropagation; The vanishing gradient problem and the exploding problem. Both stand for improper training or wrong updating. On the one hand, the vanishing gradient describes that the neurons at the very back take too much time to update or even stop being updated. Nonetheless, the exploding gradient problem represents the closest to present neurons, which tend to have the highest weights, which gives less importance to farther weights. Both issues have been solved in more advanced architectures derived from the classical RNNs. RNNs are brought to this work because they have been used as an extra tool for object detection, taking advantage of its memory capacity; this can be seen in the next section.

2.2.8. CRNN

The Convolutional Recurrent Neural Network (CRNN) is the combination of the two previously mentioned kinds of NNs; the CNN (Section 2.2.1) and the RNN (Section 2.2.7). The CRNN architecture has been widely used in several applications such as sound recognition and classification [29–31], where they have found particular success. Figure 11 shows the schematic of a CRNN.

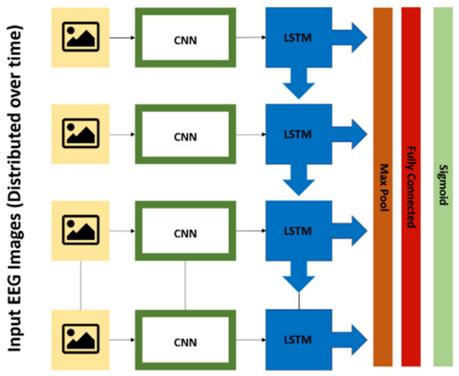

Figure 11. CRNN Architecture, in which a CNN and a type of RNN are present among other layers.

CRNN based methods were developed to generate more profound and accurate image analysis. Given that convolutional layers have mainly been used to obtain image representations and features, these methods were enough to get acceptable results in many cases. However, as the need for a better quality of analysis has increased (due to the high demand for accurate classifications), recurrent layers have become a promising solution in combinations with the ones mentioned above. This setting offers the possibility to remember and connect independent features.

Zuo et al. [32] proposed a CRNN model consisting of a CNN as a first step to extract middle-level image features and general image analysis. In series to the convolutional layers, the data must be adapted to suit the recurrent layers; this can be done by converting the regions previously obtained into a matrix of the total amount of areas. After this step, the data can be analyzed in a parallel recurrent layer of 1-dimension for each one. From this output, dependencies and correlations are generated from the recurrent layers, taking advantage of the memory capacity. Finally, the parallel layers must converge in a final layer that will merge all the gathered information.

This method's improvement was presented by Hu et al. [33], doing the same procedure of starting with a convolutional layer to extract image features and then feeding the data into the recurrent layers. This method proposes the implementation of the CRNN module in between any of the convolutional layers. What differentiates this module from a classic RNN is that the connection between layer and layer is convolutional and not fully connected. This approach demonstrated better performance, reliability, and accuracy, although they consume more computer capacity for each of the modules used.

2.2.9. LSTM

Long Short-Term Memory (LSTM) is a recurrent-type cell for regressive NN. LSTMs are explicitly designed to avoid long-term dependency. This architecture consists of three gates, mainly intended to solve gradient disappearance and explosion in the long sequence training process. Put merely, LSTM can perform better on longer sequences than normal RNNs. An example of the LSTM architecture is shown in Figure 12.

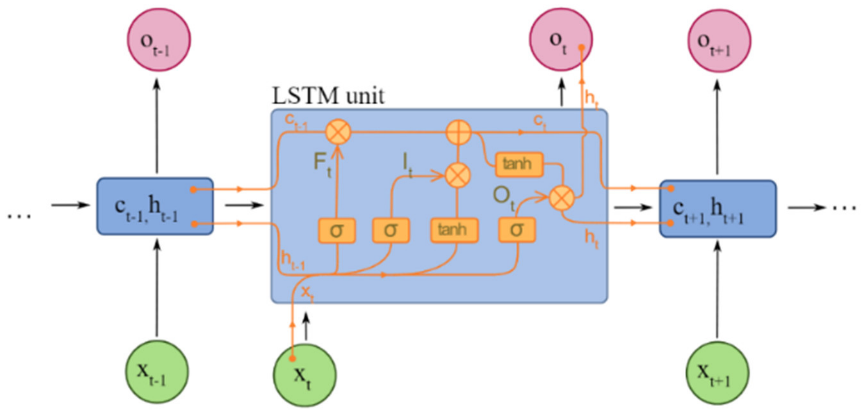

Figure 12. Long Short-Term Memory (LSTM) architecture, with three gates composed of four NN layers (image under creative commons license).

The previous image shows in more detail the structure of an LSTM, which, contrary to the standard RNN, the repeating modules have four neural network layers instead of a single *tanh* layer. The principal idea behind the LSTMs is to have the ability to add or remove information to the cell state, which is the horizontal line running through the top of the diagram, where the data could flow along with it unchanged. This ability is regulated by structures called gates composed of a sigmoid neural net layer and a pointwise multiplication operation. Depending on the digital output state, the information would be let through. LSTMs have three of these gates. The first one decides which inputs are going to be ignored by the cell state. The second one determines which values will be updated. With the help of a *tanh* layer, which creates a vector of new candidate values, an update to the state will be produced to decide which further information will be stored in the cell state. Finally, the third decides what parts of the cell state will be the output [34].

2.2.10. GAN

A Generative Adversarial Network (GAN) is a type of NN that consists of two architectures: a discriminator D and a generator G. In principle, the idea is that based on a minor amount of training data Z, D is capable of generating a set of artificial data X_{fake}. Afterward, D is trained on X_{fake} and real data X_{real} to recognize the difference by estimating losses from prediction and ground truth. An example of this architecture is shown in Figure 13. Notice that, in contrast, G estimates the log-likelihood of the distributions of the data. The early version of this architecture, often called Vanilla GAN [35], uses the *Kullback-Leibler* divergence as the distribution similarity to produce false images. Some other architectures presented in the literature are C-GAN [36], S-GAN [37], AC-GAN [38], and F-SGAN [39], with an ever-growing market in the research community (Hindupuravinash, 'The GAN zoo', *Github*, hindupuravinash/the-gan-zoo: A list of all named GANs! (github.com)).

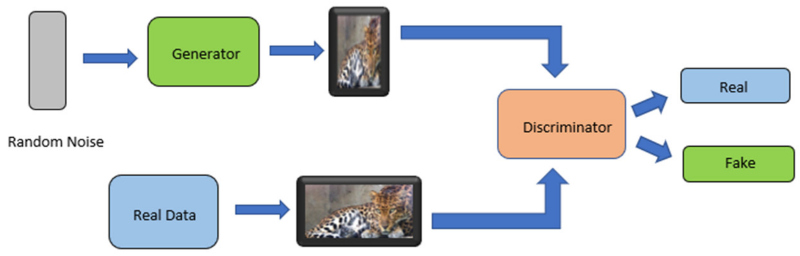

Figure 13. Generative Adversarial Network (GAN) architecture.

These networks are typically used to generate more data samples so that the data used to train classification systems can be improved. Most notably, this has been applied for industrial applications such as face detection [40], fish classification [41], and symbol recognition in engineering drawings [42].

2.3. Distributed System (DS)

According to Coulouris et al. [43], the concept of a DS refers to different agents connected working for a standard system. Each node communicates through messages to achieve a common goal. A DS in the cloud has the following characteristics:

- *Concurrency*: Each node can perform different tasks or parts of the same task-sharing resources, allowing parallel computing.
- *Quality of service*: this concept refers to the idea that a specific type of message's sensibility depends on the type of information transmitted. For example, if you download a file, it may not be vital if it delays for some seconds. However, in a phone call, or a video call, any delay may result in a loss of data that troubles communication.
- *Scalability*: Any network must be built, having in mind that it may expand.
- *Security*: With the constant growth of devices connected to the cloud, and the high amount of information transmitted, safety must be implemented on every network designed.
- *Failure handling*: In a DS, there must be a plan for possible failures. If a computer fails, the others will keep working.
- *No global clock*: This happens because of the communication through messages. Each node may have its frequency of operation as each one has its own hardware. This situation may difficult the transmission because there is a limit to computer synchronization accuracy in a network. Therefore, a bottleneck in communication, where the best bandwidth that may be reached is the lowest in the network, may be caused. This allows each node to operate independently with its requirements of hardware.

However, it's important to note that each node is a different device because, in edge computing, they will process the information without sending it to the servers. The nodes that are closest to the sensors can be the ones that process the information because they will have a processing unit, memory, and the hardware needed resulting in less latency, less energy consumption, more privacy, and a faster process overall. When deep NN grow in complexity, usually the training time increases as well. CNNs, as well as other architectures mentioned above, might face this challenge, especially when several convolutional layers are required. With the current average computer systems, a CNN might end up needing a couple of hours to train, not to mention the time it takes to process the input data, along with any other specific operation.

Due to this hindrance, several attempts in the literature to develop DS applied on NNs, now referred to as Distributed Deep Neural Network (DDNN), have been presented. DDNNs implemented through DS offer the same advantages with regards to the ones mentioned above for DSs. In [43,44], the authors modified the raw data before sending it to the NN in the cloud. This prototype is an example of edge computing applied in DDNN to avoid bandwidth limitations in network communications. Additionally, this helps with training time because the information will be filtered. As a result, the system's privacy is strengthened because all information packets will only be available locally on the nodes that gather data. It is expected that, similar to image pre-processing, NNs in the cloud need to modify data before feeding it to the NN. One might even say it is necessary because of the limitation of bandwidth. Usually, gathered data needs to be pre-processed before uploading it to the cloud as input to the NN, as shown in [43,44].

Usman et al. [44] implemented a cloud-based video analytics system. They used several layers in the system to process the information. They used parallel model training, which trains several partial models and combines them in a central one. The information received was filtered by separating consecutive frames that lacked a significant change in the object. These characteristics reduced resource usage and training time by filtering video frames that lacked an essential change in the image observed.

Teerapittayanon et al. [45] proposed a DDNN, which works over the cloud, edge, and end devices. This system was used to analyze different sensor data, minimizing resource usage for devices by processing a portion of the information recovered in the end devices and an edge layer. This separation also allowed system fault tolerance and enhanced data privacy because each end device did not offload raw sensor data to the cloud directly.

Although this approach may come with several advantages, it also brings a handful of problems to the table. Unlike centralized systems, information transferred faces the problem of bandwidth. Data cannot travel at the same speed. In the Internet of Things (IoT) systems, the filter can be local, right before traveling to the cloud. This way, useless or repetitive information can be disposed to reduce time and space for training, allowing the part of the DS in which the NN is present to focus all its resources to train on sufficient data. In [44], a CNN was trained to classify objects. Although videos were set to 25 frames per second, the system only used five frames each second.

2.4. Metrics for Evaluation

Several metrics exist to evaluate the performance of object detection and segmentation tasks. These are useful to assess the behavior of a new model and to compare it to existing ones. Furthermore, there are competitions, such as the Common Objects in Context Object Detection Challenge (COCO-ODC) or the PASCAL Visual Object Classes (VOC) Challenge [46], that use a combination of metrics to serve as benchmarks of quality for object detection models. Before explaining said metrics, some concepts need to be defined. Firstly, the Intersection over Union (IoU) or Jaccard Index is defined as the intersection between the predicted bounding box and the ground truth bounding box over the union of those two, or simply put, the similarity between the prediction and the ground truth. This score ranges from 0 to 1; the higher the score, the closer the prediction is to the ground truth. This metric can generate four outcomes: (1) True Positive (TP) the model detects an object where, indeed, an object exists, (2) False Positive (FP) the model detects an object where there is no object, (3) False Negative (FN) the model detects no object where there is an object present and (4) True Negative (TN) the model detects no object where there is no object. The next essential concepts are precision and recall. Precision refers to the probability of the prediction matching the ground truth. It can be defined as $\frac{TP}{TP+FP}$ or $\frac{P}{ALL\ DETECTED\ BOXES}$ [47,48]. In counterpart, recall refers to TP's rate or the probability of a TP to be detected. The recall can be defined as $\frac{TP}{TP+FN}$ or $\frac{P}{ALL\ GROUND\ TRUTH\ BOXES}$. Adding on, the precision-recall curve is a metric that summarizes the two previous concepts and is especially useful to assess the performance of models with imbalanced datasets [30]. In this curve, precision is plotted on the -axis and recall on the x-axis. Ideally, a model should deliver both high precision and recall, although this is not always the case. When comparing models, generally, the one with the highest curve indicates the best performance. Another way to gauge these two metrics is the F-measure or F1-score, defined as $F1 = 2 \times \frac{precision\ \times recall}{precision+recall}$.

Furthermore, it is also possible to calculate average precision (AP), which, as the name indicates, is the average precision among the precision-recall curve. Mean average precision (mAP), on the other hand, involves the AP among several classes, whereas the AP only covers one. The mAP is defined as $mAP = \frac{\sum_{I=1}^{K} AP_i}{K}$. Regarding the competitions mentioned above, the COCO-ODC uses 12 metrics, where mAP is the main one. On the other hand, PASCAL VOC uses two metrics: the precision-recall curve and AP. It is also worth mentioning that different challenges use different datasets with different scenarios in mind. For example, the KITTI dataset consists of hundreds of hours of driving systems gathered from an on-board camera on a car, which is especially useful when working with AVs.

On the other hand, the COCO-ODC dataset contains many frequently found objects (80 different classes) such as animals, tools, people, etc. The PASCAL VOC data set is very similar, and both are commonly used when training object detection architectures. However, there are some key differences worth noting. For instance, annotations in COCO are facilitated by employing a JSON file, while PASCAL uses the XML format. Furthermore, the bounding box format is different: COCO uses $xmin$ top-left, $ymin$ top-left, width, height, while PASCAL uses $xmin$ top-left, $ymin$ top-left, $xmax$ bottom-right, $ymax$ bottom-right [46]. Although these differences may sound minimal, some researchers decided to focus on specific challenges based on the formats that they are more familiar with. Thus, there are no common grounds to compare every single algorithm ever made.

2.5. Frenét Motion Planning

The Frenét frame method consists of transforming the Cartesian coordinates x and y to a more dynamic reference frame n and t, which are normal and tangential to the road vectors, respectively [49]. If a motion planning algorithm in a highway uses the traditional cartesian coordinate plane, it would have to parametrize x and y coordinates positions concerning time and solve for polynomials with significant coefficients. This is a challenging and computationally expensive task. Thus, the Frenét coordinate system uses vectors projected on the road's curves, treating the road as if it was all forward plane with no curves. Hence, by changing these vectors' magnitude, lane changing and braking and acceleration happen to be an isolated problem from the road following task. It helps to have two different easier problems: lane keeping and lane changing, instead of one more difficult situation. Frenét coordinate systems are preferred since these allow for lane-keeping using minimal computing effort. They also predict and calculate lane deviation in dodging for static or dynamic obstacles to then return to the lane's boundaries. The Frenét algorithm does not execute the prediction task, but its output can be easily used in many prediction algorithms. The Frenét coordinate system can also be taken for peripheral vehicles other than the ego. Still, its original purpose remains for the car to perform inertial odometry tasks. Much more can be exploited, but to get it to work, many sensor fusion tasks need to be performed to state the vehicle's center of mass since it will be the origin of the Frenét coordinate system. For example, in Figure 14, a route in which x and y coordinates are the curve's points, the vehicle doesn't see those coordinates. Instead, it transforms its coordinates to vectors T and N, so the car perceives itself as traveling in a straight line, only changing the N vector to keep the lane and vector T changing its speed. To update its N and T vector values, the following linear algebra equation is used:

$$\begin{bmatrix} T' \\ N' \end{bmatrix} = \begin{bmatrix} 0 & \delta \\ -\delta & 0 \end{bmatrix} \begin{bmatrix} T \\ N \end{bmatrix}.$$

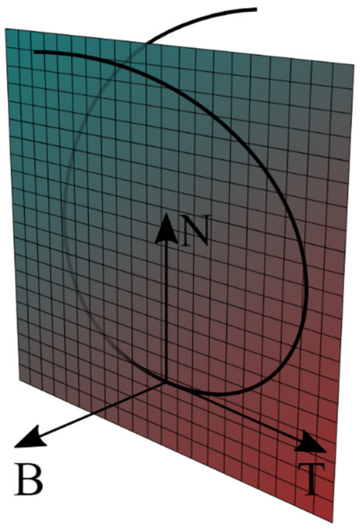

Figure 14. Frenét frame visualization (image under creative commons license).

Since each car's Frenét route needs to be predicted, the T and N vectors are given by combining many algorithms, as discussed later in Section 3. The T and N derivatives are calculated with respect to the arc length, and thus the only missing value, and the one that needs to be predicted, is δ. Once δ is predicted, the Frenét Frame can be calculated by solving for the x and y curve which is tangent to T and normal to N. The updated frame for each point of δ will be 35 Hz as it is the vision algorithm frame rate and T and N will be captured using computer vision algorithms.

It is essential to state that the Frenét framework is mainly used for motion and path planning, and so the equation given above is not the only one.

3. Methods to Address the Pipeline of End-To-End ADS

This section will discuss the latest literature methods to address the different stages that constitute an end-to-end ADS. This section is divided into four main categories. The first one presents image pre-processing and instance segmentation, in which the main aim is to locate all legal boundaries and other objects of reference. It is inferred that although one vehicle is physically capable of driving on a sidewalk, it is not correct for it to do so, and therefore the ADS needs to recognize the road lanes. Nonetheless, not all road lanes are used for driving in the required direction (i.e., the single direction only, cycling/pedestrian lanes). Thus, this step plays a crucial role in detecting which road lanes shall the vehicle occupy. The second one focuses on object detection within the semantic segmenting boundaries of the route. For instance, algorithms such as YOLO are used to recognize each object, and this data is fused with LiDAR point clouds using an MV3D-like architecture [12]. Processes such as these help the vehicle draw prism points on each detected object and know where its center of mass relies upon and the volume being handled within the lane space. Human and sentiment detection techniques (for driver, passengers, and pedestrians) are discussed in the third section. Finally, in the fourth category, cloud computing and parallelization are discussed to speed up and improve the pipeline. For instance, these techniques can train all object detection models, communication of different vehicles, etc.

3.1. Image Pre-Processing and Instance Segmentation

Chen et al. [18] proposed a CNN model for object localization using camera and LiDAR data for 3D multi-view object detection, or MV3D. In this method, three CNNs are deployed in parallel. Two of them are fed with the point cloud LiDAR data (one obtained from a BEV and the other from the front view). The third CNN uses raw RGB images from a camera located at the front, on the sides, and the car's back for a 360-degree view. Once each architecture finishes, a deconvolution process occurs to identify where the tracks are located and their shapes. Like an RCNN architecture, a decoder-like function outputs decoding fed to the architecture performing the localization task and shared between the other parallel architectures. This process generates a 3D object proposal of the classes, combining a region-wise feature obtained via ROI pooling caused by each architecture. The output is a 3D bounding box around the recognized objects. It is important to note that this method can realize the item's class, position, and volume. This method performs with a 14.9% higher accuracy than state-of-the-art methods, comparable in runtime. As a result, this method is computationally demanding, and deducting the 3D bounding box will require a specific and robust hardware architecture for the algorithms to be deployed correctly and in real-time scenarios.

Zhou et al. [50] proposed VoxelNet, an end-to-end trainable deep architecture for point cloud-based 3D detection. In contrast to the previous two methods, this is a LiDAR-only based detection, which relies on the object size, occlusion state, and other superficial information. To improve the localization and detection, the authors proposed the Multimodal Voxel for 3D Object Detection Network (MVX-Net) framework. In this method, fused augmented LiDAR point clouds with semantic image features are used at early stages to improve 3D object detection. Moreover, two fusion techniques were developed to extend the VoxelNet. The first one is PointFusion, which combines features at early stages to add 3D points to an image feature and capture dense context, extracting features from a 2D detector (i.e., a camera) and then jointly inputting them into the VoxelNet model. The second technique is called VoxelFusion and employs non-empty 3D voxels to be projected on the image plane. Features are extracted and fed into the voxel feature encoder used by a 3D RPN to deduce the bounding boxes. Authors state that the VoxelFusion technique delivers a slightly inferior performance compared to the PointFusion one; however, it has more efficiency in memory consumption. The authors evaluated the KITTI 3D object detection benchmark model and concluded that their approach demonstrated better results than the single modality VoxelNet [51].

Avoiding person-to-person contact is the best way to control and prevent the virus' transmission amidst the COVID-19 epidemic that affected the world in 2020. Still, people require food and other basic goods to address their essential needs. Liu et al. [52] developed Hercules, an autonomous logistic vehicle used for contactless delivery, to address this issue. It has an autonomous navigation capability and a stable hardware-software system to manage the operations. Its maximum payload and capacity are 1000 kg and 3 m^3, sufficient for most of the people delivery's requirements, and uses 3D object detection with LiDAR point clouds to recognize and classify objects. Multiple calibrated LiDARs provide data that are fused as input into the 3D object detector of the VoxelNet. Authors consider that the real-time performance deserved much more attention than accuracy for this particular application. To improve efficiency, they replaced dense convolutional layers with spatially sparse convolutions to obtain inference time boosting. Another critical part of Hercules' perception task was to build the 3D map of the environment with 3D point clouds from the LiDARs and the readings from the inertial measurement unit. Objects in motion should be treated differently than static entities. Thus, this task is very crucial for motion planning in AVs. Hercules' solution is called Ego-motion Compensated Moving-object Detection, which detects objects based on the consecutive point cloud frames. The autonomous navigation technology was reported to work well on the state-of-the-art scenarios.

The Intelligent Autonomous Robotic Automobile (IARA) was presented in Moraes et al. [53] and Cardoso et al. [54]. It is a vehicle that uses a suite of sensors (including LiDAR and odometry sensors) to perform various tasks. The two papers present an NN-aided approach to state-of-the-art navigation, mapping, and localization methods. The first publication presents a novel, image-based real-time path planner. Using a CNN, the algorithm can deduce a path from images the on-board camera sends. This path (a cubic spline) is then transformed to the car's coordinates, and thus the route can be followed. The second publication presents a new approach for real-time inference of occupancy maps using deep learning. The network proposed in this paper, NeuralMapper, takes the LiDAR data as input and creates an occupancy grid map. It further uses a NN to infer certain parts of the route where LiDAR data is not very accurate, creating a completer and more efficient map.

If a car detects an obstacle, it brakes and corrects its course, but it does so when the interruption has already occurred. Analyzing human gestures can help avoid sudden collisions; however, deep learning models do not have enough data to train them. For this reason, Cruise's AVs [19] use motion capture to understand human gestures technology, which is a technique that video game developers use to create and animate characters. With this technology, the necessary characteristics would be extracted to train the deep learning models and be one step closer to predicting pedestrians' or drivers' sudden movements.

There are two types of mo-cap systems optical and non-optical. The visual uses cameras distributed over a sizeable grid-like structure that surrounds a stage; the video streams from these cameras can triangulate the 3D positions of visible markers on a full-body suit worn by an actor. In a non-optical way, this system [55] uses a sensor-based version of motion capture instead, which relies on microelectromechanical systems (MEMS) [56,57], which are portable, wireless, and do not require dedicated studio space. That gives a lot of flexibility and allows to take it out of the studio and collect real data of world locations.

Barea et al. presented an integrated CNN for multisensory 3D vehicle detection in a real autonomous driving environment [58]. As the title expresses, the proposal of their paper aims to present an outstanding architecture based on the combination of state of the art methods for object detection such as YOLO and Mask R-CNN for 3D segmentation besides a LiDAR point cloud. Figure 15 shows the results achieved in this paper, integrating Mask R-CNN, YOLOv3 and cloud point LiDAR. It can be seen that an accurate bird eye view, which corresponded with the object detection in the superior image, is created. Moreover, an accurate mapping of the road's current state, with the vital information of the 3D sizing of the moving and static objects in the ego car's peripheric view, is obtained with complete navigation details. The unique architectures applied in this proposal achieved high performance in the KITTY test set.

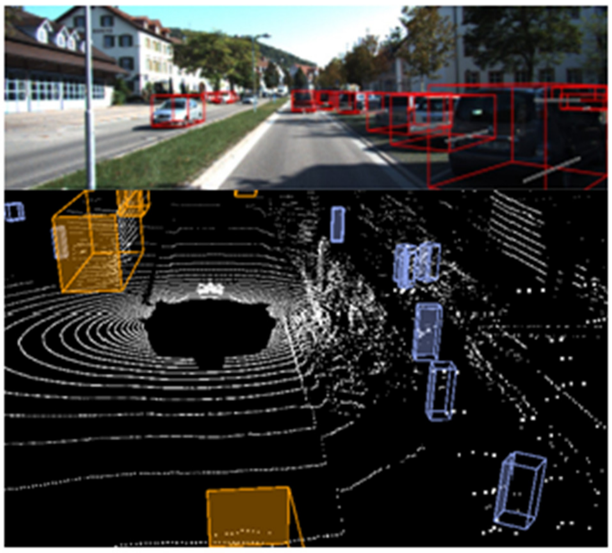

Figure 15. LiDAR's birds-eye view plus boxing object detection.

This work proves that individually each architecture cannot overcome itself. Still, in the correct combination, new opportunities might be found. For the reader, in machine learning and similar technologies, the novel results are achieved based on experimenting and not in a fixed formula. Thus, the following question arises: What would be the best set combination for acquiring the correct data for a completely autonomous car?

As mentioned previously, autonomous driving is a challenging problem addressed with various technologies, methods and algorithms. Still, it seems that there's always space for more efforts due to the rigorous task of taking decisions in an always-changing and mostly unpredictable environment, especially on busy streets and highways. The following paper [59] discusses how image segmentation allows the class to classify the road's drivable and non-drivable regions. To do so, a Mask R-CNN architecture was trained to differentiate these possibilities. The architecture was trained and tested in the Berkeley Deep Drive (BDD100k) dataset, with 100k on road pictures, achieving a final mAP of 0.79 (a sample of the results is provided in Figure 16). To identify the drivable areas, the model determines the ego car's current line, making the difference with the alternatives. The implementation problem was broken into various subproblems; road object detection to declare a line as non-drivable, segmentation breaking a 2D Image into depth-based layers, and lane maker detector to separate each of the lanes.

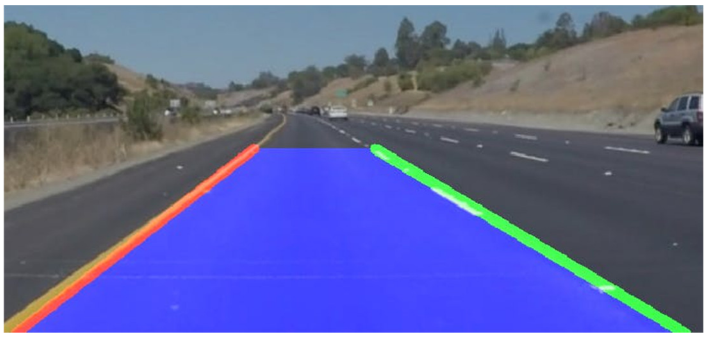

Figure 16. Line segmentation based on Mask R-CNN.

This paper also explores new possibilities demonstrating that the architectures mentioned in the previous part of this paper can be used in different ways to achieve similar objectives, which provides more information to the ego car, to be able to make better decisions by evaluating the state of the environment. On the other hand, this paper does not mention the time it takes for the algorithm to drop a given state's results.

3.2. *Driver Assistance and Predicting Driver Patterns*

One characteristic that differentiates a good driver from a bad one is the capability of choosing the best possible action in the proper time and execute it correctly, at all times, even in the rarest and most dangerous circumstances. In those cases, being a skilled driver can save lives. In the context of AVs, this is a real threat because, as mentioned before, one of the biggest challenges for self-driving cars is the unpredictability of the always-changing road conditions and human behaviors. At its beginnings, AVs were based on precarious algorithms, based on state conditionals, with the first driver-assisted systems, acceleration and deceleration, and conditional automation. The human intervenes only when needed. These old approaches were not able to counter the odds of real-time decisions (WIRED Brand Lab, *A brief history of Autonomous Vehicle Technology*, WIRED, 2019, A Brief History of Autonomous Vehicle Technology | WIRED).

Nowadays, state-of-the-art aims to evolve to fully automated systems, where non-human intervention is needed. But one barrier that challenges all the developments is how a human perceives the driving and, consequently, decides how a computer does. Many background context conditions are known for humans that can be subjective and for computers becomes complicated to understand; for example, the local cultural manners of driving often define what to expect from other cars, which differ in geographic location. Another condition is the passengers' feel. This concept means that it might be the same for a computer to avoid an object a second before the collision; therefore, this move can be made smoothly. Conversely, this decision for the passengers may seem uncomfortable, causing other issues, such as a human hitting the window. Therefore, the autonomous system should consider the passengers' inner motions, thus being dynamic and based on human behavior to improve it, without realizing actions that might confront what a human expects from another car.

The mentioned considerations are causes and circumstances that must be considered for a full automotive driving system. Thus, the decision-making part of an autonomous system is a critical part of the process. Driving patterns are ways in which a human driver behaves. These patterns are developed through experience and practice. Humans can predict how the environment can change within these driving patterns, and based on these predictions, a decision is made. Various methods and solutions make use of different NN architectures [60]. Some of them are explained in the following sections.

3.2.1. Behavioral Cloning (BC)

BC consists of a CNN that uses the real driving data obtained through the vehicle's camera and computer when driving during different situations. Then, the network learns the response of the driver to obstacles in other locations. According to [61], a whole model can be trained using only raw data and expect to have a steering output. This approach is called end-to-end learning, but electromobility and autonomous driving named BC are much more appealing. End-to-end learning is called that way because there are no subprocesses when expecting and output from the NN. These subprocesses include SLAM, Path planning, object recognition, etc. None of these techniques is used for the vehicle's motion planning; the only information used is data taken from feature extraction in images taken when the car was being driven. These images have the steering angle and gas pedal position as a label for the dataset. Thus, when seeing a similar scene to the ones in its training data set, then the gas and steering values are close to those trained on.

It could be very promising that there are no subprocesses required for achieving autonomous driving. Using raw images, then the only two values needed for driving are the ones generated. But the reality is much more discouraging. It would require enormous amounts of data, which is not at hand. Because of the model generalization, millions of different driving scenarios need to be recorded but have not yet been taken. BC works only during certain situations, but it works encouragingly. BC is a method that deserves to be discussed in the future once more data is available. For the time being, end-to-end learning is used and applied in natural language processing because of its capability of finding sense in sentences, compared to traditional methods. Data in this language is large enough for end-to-end learning to be applicable, not the case for autonomous driving in which research is being developed but years away from practical applications.

3.2.2. Reinforcement Learning (RL)

One of the most famous machine learning paradigms is RL; this type of algorithm has been used in different ways to process the information given by the sensors to output the best actions. This broad field of study aims to replicate with algorithms how living beings learn through the process of gaining experience by exploring their environment and realizing which activities are suitable and which are not, based on rewards, concerning a specific objective or task inherent to the environment. RL has been widely used as a control algorithm to determine the best policy or a simple word. This strategy will determine the best possible action for any presented state or circumstances that the environment presents [62]. Figure 17 shows the basic flow of this type of algorithms.

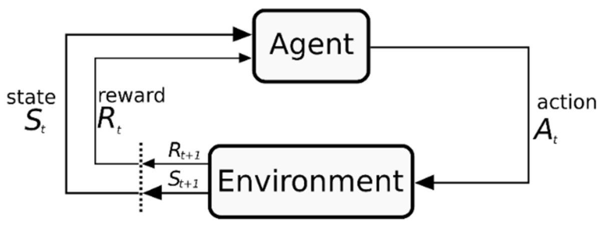

Figure 17. Basic Reinforcement Learning (RL) process, agent-environment relation (image under creative commons license).

The necessary process for an RL to work is as follows. An agent that is the learning object interacts with the environment, which can be virtual or real, the agent performs actions (a) given an environment state s, that will affect the environment, in exchange, the environment will feedback the agent with a new state s' and a reward r. If the action were right, then the reward would be positive. Contrarily if the move were wrong, the reward would be negative; in other words, a punishment. The purpose of the agent is to maximize rewards. The process aims to create a system that behaves in the best possible way to achieve a goal.

With a good overview of RL and without deepening the algorithm's details because of this work's scope, this paradigm's application to AVs can be discussed. This type of algorithm is used in the second part of the vehicle's decisions process. First, it has to acquire data of the environment, in this case, the road and the objects such as other cars interacting within it, via the already mentioned and studied approaches such as LiDAR and object detection algorithms via image processing. Based on these data, advanced RL versions based on NNs, such as deep RL, deep Q-learning, and deep deterministic policy gradient (amongst others), have been used for AVs' decision-making processes. These deep versions of RL present the advantage of a more robust, more powerful capability of learning behaviors, understanding patterns, and predicting situations in comparison with their non-deep versions.

An example is control of speed and lane change decisions [63]. As the authors mention, a known problem of decision taking on AVs is that the methods are designed for a specific task. An example would be the Intelligent Driver Model (IDM), which is applied to decide only when to change lines. Therefore, there's a need for a more global algorithm to take care of AVs' multiple decision needs. This work proposes that the car uses a 27-input vector providing information on the road and eight possible surrounding cars to make lane change decisions and speed control. However, in the same DQL algorithm, they trained two agents to address each task [64].

One perceived disadvantage of RL algorithms in autonomous cars is that they cannot predict other road users' intentions but evaluate all the objects' current state. These kinds of learning do not have long-term future recognition of the environment. This problem is intrinsic in the math that the algorithms are based on because Markov's decision process takes decisions based on the last observations state.

3.2.3. LSTM Based Models

Currently, there are many approaches to road prediction using LSTM architectures for these types of forecasts. Bai et al. [65] suggested spatiotemporal LSTM architecture for motion planning in AVs. It uses convolutional layers for feature extraction in an image, and it runs through an LSTM to find sequences in those images, something similar to a CRNN architecture. Once this data is acquired, a CNN architecture is applied to extract spatiotemporal information from multiple frames. Lastly, a fully connected layer is used to extract the trajectory taken by peripheral vehicles. This trained model can be of service for detecting multiple object trajectories. It is considered an end-to-end algorithm since all this raw data enters this architecture, and it outputs a solution. Additionally, it is quite demanding in terms of training and testing data.

Atlché et al. [66] propose using an LSTM architecture for lane changing detection on highways, using data such as velocity and lateral displacement from vehicles. It was trained using the NGSIM US-101 dataset and trained two different LSTM models, one for lateral displacement and another for linear velocity. It performed very rapidly with 70 cm of error in displacement and 3 m/s error for speed, with 10 s ahead of raw predictions. This is a less direct approach since it requires the labeling of data. It can be assumed that they are using a Frenét like topology since they are only considering linear velocity and lateral displacement in their model.

3.3. Human Sentiment Analysis

One of the most critical objectives for intelligent transportation applications is road safety, and unfortunately, one part of traffic accidents is human error. A driver's behavior could affect its way of driving. For that reason, a monitoring system inside an intelligent vehicle would provide relevant information and indicate if the driver's action is allowed. Otherwise, the car would be able to correct it. This monitoring system used cameras inside the cabin of a vehicle to detect face sentiment. Then, CNNs are applied for action recognition. To be more specific, R-CNNs are the ones that select the most informative regions from the drivers [67].

These NNs are trained and tested with an extensive data set to detect any change in gaze and facial emotions, thus discerning regular and aggressive driving [68] or distracted driving actions [69]. In some other cases, there want to predict drivers' behavior; to make this possible, CRNNs are used. Kim et al. [70] proposed a line-segment feature analysis-convolutional recurrent neural network (LFA-CRNN) to analyze drivers' facial expressions indicating pain to prevent automobile accidents and respond to emergency health-risk situations. The LFA-CRNN showed an accuracy of 97.4% in contrast with 98.21% and 97.4% of CRNN and AlexNet.

Other systems also analyze people's voice tones inside the cabin, such as Affectiva Automotive[2], which understands what is happening inside a vehicle. This technology works as both a driver and monitoring tool, ensuring that the safety drivers keep their eyes on the road even as the self-driving software drives the car. An emotional tracker ensures robot taxi passengers feel safe during the trip. The system monitors levels of driver fatigue or drowsiness, driver distraction, understanding driver mood and reactions, and could enhance fleet management. To solve these tasks, the software uses some deep learning architectures that have been mentioned in the previous sections. For instance, for face detection, tracking and classification are used CNNs, in specific RPNs; and for voice-activity detection and classification RNNs, in specific LSTM.

3.4. Using the Cloud with Autonomous Driving Systems

To solve the limitations posed by cloud systems, there are different solutions in the literature. In this section, some of the solutions proposed are discussed. According to Liu et al. [71], the cloud has to provide distributed computing, distributed storage, and heterogeneous computing. It is expected that these systems are tailored together. However, this creates problems with resource sharing because it has to be copied between each application, reducing available space and speed. In this paper, the authors propose a unified infrastructure that complies with the services mentioned. The system developed was reliable, had low latency, and a high-throughput autonomous driving cloud. These practices are useful if anyone wants to build a system in which several AVs share the compiled information.

Kumar et al. [72] developed a cloud-assisted system that involved several vehicles communicating with each other and with several sensors located in the environment. To avoid bandwidth limitations, they used a request-based system, which allowed the car to have information about different locations at various resolutions. They used the Octree representation, which is a 3D-point cloud that can be divided recursively into eight. For more information, the reader is referred to [72].

4. Future Trends

Research on AVs has shown great promise, and there have been so many breakthroughs in recent years. However, there are still many challenges that the community needs to overcome to provide the safest and robust vehicles. All the sensors involved in ADS produce a lot of data from their surroundings, and they are required to be processed in real-time so that the AV can make a correct decision; this is very critical because a small delay could make a considerable change. In this section, some future ADS trends, which can improve performance and reduce response times, will be presented.

4.1. Cloud Computing

Nowadays, cloud computing is becoming a trend for many aspects of our everyday life. More and more businesses are migrating several of their systems to the cloud because of its availability, security, scalability, among others. When discussing businesses' status, payments, or stock inventory, the amount of time required to get a result may be enough if a human perceives it as instantaneous or even tolerated if a small delay occurs. However, in application domains such as autonomous driving, real-time results are a must if accidents or life-threatening scenarios have to be avoided. Thus, the need to explore concepts such as edge and fog within the cloud computing domain. Edge and fog computing consists of moving the location where the information is processed closer to where it is generated [73]. Any device with an internet connection that can store and process data can be used as a fog node (Cisco, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are", *Cisco*, computing-overview.pdf (cisco.com)). This way, the cloud server might rely on these nodes for most analysis, reducing latency and increasing privacy. The authors considered three characteristics of an application that needs fog computing:

1. Data is gathered on an edge. In this case, an AV or a group of AVs can be considered an extreme edge.
2. There is a lot of data being generated in a large geographic area.
3. The information gathered needs to be analyzed, and it has to provoke a response in less than a second.

An average cable provider in 2008 allowed 16 Mbps (Jakob Nielsen, Nielsen's Law of Internet Bandwidth, *Nielsen Norman Group*, Nielsen's Law of Internet Bandwidth (nngroup.com)). Two years later, bandwidth increased to 31 Mbps. The last measurement was from 2019 with 325 Mbps. This behavior can be described by Nielsen's law, which says that each year, bandwidth increases by approximately 50%. Some authors even compare Nielsen's law with Moore's law, which describes computer power. Despite recent advances in bandwidth capacity, edge computing is still the most critical step for real-time cloud analysis. As shown in [45], a DDNN that involves the edge brings several benefits, such as fault tolerance, privacy, and reduced communication costs.

All and all, it is concluded that edge and fog computing are the best ways to implement a NN in the cloud for applications that require fast analysis like AVs. Despite requiring more powerful end devices, or edge servers, these techniques reduce network traffic, allowing faster analysis of information, more privacy, and fewer transmission errors.

4.2. Parallelization of Neural Networks

The review of deep learning NNs presented in Section 2 showed that the accuracy of models could be improved by increasing the number of parameters and the scale (i.e., nodes, layers, etc.). However, these actions also derive slower training times, which is inefficient for adaptive autonomous driving scenarios. Therefore, it has been acknowledged that deep learning models need to be parallelized to accelerate training and deployment. Indeed, deep learning models can be trained and executed in multiple GPUs. The key consideration to take advantage of their parallelization is to know how to divide the GPUs' tasks. Hence, three approaches should be considered if anyone wants to train parallelized models: data parallelism, model parallelism, and data-model parallelism [73].

- Data parallelism: Data parallelism is a different kind of parallelism that, instead of relying on the process or task concurrency, is related to both the flow and the information structure. Each GPU uses the same model to trains on different data subsets. If there are too many computational nodes, it is necessary to reduce the learning rate to keep a smooth training process.
- Model parallelism: In model parallelism, each computational node is responsible for parts of the model by training the same data samples. The computational nodes communicate between them when a neuron's input is from another computational node's output. However, the performance is worse than data parallelism, and the performance of the network will be decreased if it has too many nodes.
- Data-model parallelism: Both previous models have disadvantages, but they also have positive characteristics. Model parallelism could get good performance with many neuron activities, and data parallelism is efficient with many weights.

In summary, data-model parallelism combines the best of the first two approaches and is more suited for AV tasks as long as there is a clear idea of where to apply each step. For instance, the convolutional layer of a CNN could use data parallelism (since it contains about 90% of the computation and 10% of the parameters). In contrast, the fully connected layer could only rely on model parallelism because it contains 90% of the parameters and 10% of the computation.

4.3. Parallelization of the Whole System

One thing that is rarely discussed in research is the likelihood of the proposed solution being applicable in real-life settings. Factors such as the time required for the whole data to be processed and if that time is good enough for real-time applications are commonly discussed for a system to be deployed in the real world. However, hardware requirements for the processing are acceptable for these applications. Code parallelization that accelerates the clocks in the processor at just the right rate speed (but not too much so that hardware operates at dangerous temperatures) needs to be acknowledged well. In this regard, thermal throttling can be considered a viable solution. Thermal throttling is a safety protocol that the processor starts once the system reaches a determined temperature. It starts to diverge power from processing the data towards the cooling system techniques to avoid any hardware malfunctions.

Although this technique is beneficial for the hardware, it derives from slower data processing (almost half of its capabilities). For example, using thermal throttling in image processing might drop the frame rate from 30 FPS to 15 FPS (or less); this quality drop in autonomous driving could even be dangerous as it could result in lower recognition rates and maximum risk of accidents. This issue can be addressed using proper parallelization techniques in CPU and GPU architectures. It is implemented for data to be transferred smoothly throughout the system without frame drops and appropriate speed for real-time applications without compromising the hardware's integrity. For instance, CUDA libraries from NVIDIA (NVIDIA, "CUDA Toolkit 10.1 original Archive", *NVIDIA developer*, CUDA Toolkit 10.1 original Archive | NVIDIA Developer) have excellent parallelization frameworks to segment complex instructions into smaller ones; all of them are run concurrently in different GPU CUDA cores, giving the system the robustness and speed it requires. From Google, TensorFlow API also has adequate parallelization protocols for NN training and deployment in GPU architectures. It can be used in object detection, instance segmentation, and RNN prediction tasks. The robotics operating system provides parallelization options in CPU for the SLAM, path planning, communication protocols, and wireless data transfer for multiple path planning operations in the Frenét coordinate system.

4.4. First Thoughts towards an Approach for LSTM-Based Path Planning Prediction

After considering the conditions mentioned above, we propose a novel LSTM approach to predict both lateral displacement and linear velocity in a Frenét coordinate system.

Many techniques have been proposed to achieve this (as described in Sections 2.5 and 3). Still, they are considered isolated ways to solve the problem as they do not consider integrating the overall pipeline. The reader must recall that vast amounts of data are required to perform this task and acknowledge that a critical concept is often ignored in this phase, i.e., causality. It is essential to consider a high chance of a reckless driver making a poor judgment that could affect the AV, or that a single car's predictions would not change because another care changes the whole driving scenario to poor or unexpected choices. Even deployed systems, such as the Ego Car (described in Section 3), would be part of this causality scenario. Therefore, using the output from the Frenét coordinate system that was taken from object classification via YOLO and boundary classification with instance segmentation, a bidirectional LSTM can be fed with the Frenét data, both for lateral displacements and linear velocities.

These are four different LSTM models that feed their weights and share weights between lateral displacements and linear velocities. As seen in Figure 18, the LSTM architecture is represented graphically.

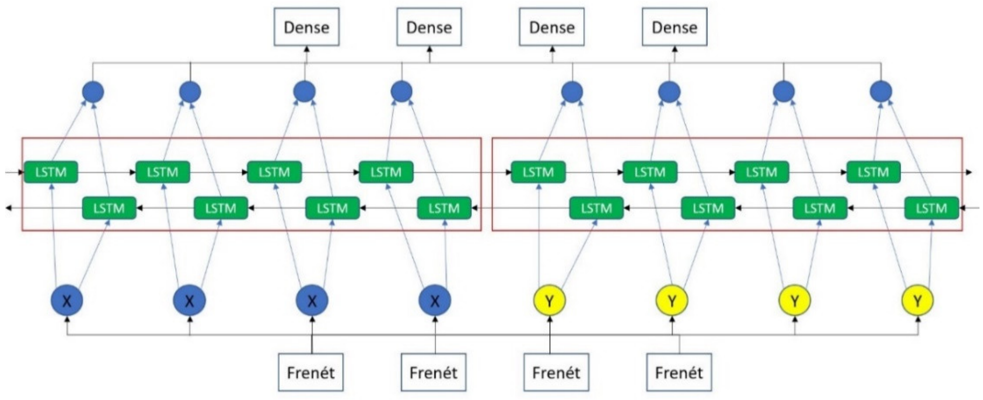

Figure 18. LSTM representation for the prediction task.

Every Frenét point of each car will be fed to the model, including the Ego Car points. Thus, the prediction will consider every variable in the scenario and change its predictions if the causality demands it. This will derive in a new Frenét prediction point for every neuron in the LSTM architecture, and so the position, orientation, and velocity of every point can be predicted. As a result, the car's volume can be included in every future time instance. This output data can be used for decision-making in an AV, reducing the number of accidents enormously since causality is part now of the overall algorithm deployment and execution.

5. Conclusions

This paper presented a comprehensive and thorough review towards achieving autonomous driving by considering concepts such as image processing, NNs, and other deep learning models. Moreover, it is acknowledged that two key concepts must be addressed to deploy these systems in real life, i.e., DS and parallelization. The review was structured towards establishing the proper ways to evaluate these systems and structure their use in pipeline systems that attempt to perform path planning (using Frénét coordinates) and identify multiple objects on the road analyze the driver's behavior. Although the review's scope is extensive and demanded a considerable amount of time to sort and select the proper references, we believe that the result is a comprehensive guide for novel and experienced researchers who intend to develop systems autonomous systems in the near future.

Our efforts and research in this domain do not conclude here; there is still a considerable amount of research to be made regarding the latest trends in this area and incorporating them into our proposed system's design. Additionally, we are considering how the latest developments, such as the COVID-19 pandemic, will affect these systems' development. The team comprises members working in different parts of the world. Therefore, our investigation in DS and parallelization techniques is not only intended for the sole purpose of improving the performance of the system but also, to better sort the training and deployment tasks in different machines and servers. Finally, it is worth noting that this study may have some further developments and applications for other areas which demand enhanced ADS, such as underwater vehicles (as described in Section 1).

Author Contributions: E.C.G.M., V.M.V.E., D.C.S., S.F.R.d.l.C., M.P.H., A.Z.C. and J.Á.G.G. investigated, compiled papers and wrote the manuscript. R.B.-B. and C.F.M.-G. supervised all developments and revised the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: Authors thank to Centro de Investigación en Microsistemas y Biodiseño (cimb.com.mx) at Tecnológico de Monterrey Campus Ciudad de México for all the support in the research process.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

ADS	Autonomous Driving System
AV	Autonomous Vehicle
AP	Average Precision
mAP	Mean Average Precision
BC	Behavioral Cloning
BEV	Bird's Eye View
CDBN	Convolutional Deep Belief Network
CLDA	Constrained Linear Discriminant Analysis
CNN	Convolutional Neural Network
COCO-ODC	Common Objects in Context Object Detection Challenge
CRNN	Convolutional Recurrent Neural Network
CUDA	Compute Unified Device Architecture
DAG-SVM	Directed Acyclic Graph-Support Vector Machine
DARPA	Defense Advanced Research Projects Agency
DBN	Deep Belief Network
DDNN	Distributed Deep Neural Network
DS	Distributed System
DOG	Difference of Gaussian
FCN	Fully Convolutional Network
FP	False Positive
FN	False Negative
GAN	Generative Adversarial Network
IARA	Intelligent Autonomous Robotics Automobile
IDM	Intelligent Driver Model
IoU	Intersection over Union
LiDAR	Light Detection and Ranging
LSTM	Long Short-Term Memory
MOT	Multi-Object Tracking
M^3OT	3D Multi-Object Tracking
MV3D	Multi-View 3D Object Detection Network for Autonomous Driving
MVX-Net	Multimodal Voxel for 3D Object Detection Network
NHTSA	National Highway Traffic Safety Administration
NN	Neural Network
R-CNN	Region-Based Convolutional Neural Network
RL	Reinforcement Learning

RNN	Recurrent Neural Network
RPN	Region Proposal Network
RoI	Region of Interest
SAE	Society of Autonomous Engineers
SIFT	Scale Invariant Feature TransformScale Invariant Feature Transform
SLAM	Simulation Localization And Mapping
SURF	Speeded-Up Robust Features
TN	True Negative
TP	True Positive
VOC	Visual Object Classes
YOLO	You Only Look Once

References

1. Singh, S. *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*; National Highway Traffic Safety Administration; U.S. Department of Transportation, National Highway Traffic Safety Administration: Washington, DC, USA, 2015.
2. Rangesh, A.; Trivedi, M.M. No Blind Spots: Full-Surround Multi-Object Tracking for Autonomous Vehicles Using Cameras and LiDARs. *IEEE Trans. Intell. Veh.* **2019**, *4*, 588–599. [CrossRef]
3. Huang, Y.; Chen, Y. Autonomous Driving with Deep Learning: A Survey of State-of-art Technologies. *arXiv* **2020**, arXiv:2006.06091.
4. Yurtsever, E.; Lambert, J.; Carballo, A.; Takeda, K. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* **2020**, *8*, 58443–58469. [CrossRef]
5. Badue, C.; Guidolini, R.; Vivacqua, R.; Azevedo, P.; Brito, V.; Forechi, A.; Ferreira, A. Self-Driving Cars: A Survey. *arXiv* **2019**, arXiv:1901.04407. [CrossRef]
6. Szymak, P.; Piskur, P.; Naus, K. The Effectiveness of Using a Pretrained Deep Learning Neural Networks for Object Classification in Underwater Video. *Remote. Sens.* **2020**, *12*, 3020. [CrossRef]
7. Lowe, D.G. Object recognition from local scale-invariant features. In Proceedings of the International Conference on Computer Vision, Corfu, Greece, 20–25 September 1999; pp. 1150–1157.
8. Bay, H.; Ess, A.; Tuytelaars, T.; Van Gool, L. Speeded Up Robust Features. *Comput. Vis. Image Underst.* **2008**, *110*, 346–359. [CrossRef]
9. Cortes Gallardo-Medina, E.; Moreno-Garcia, C.F.; Zhu, A.; Chípuli-Silva, D.; González-González, J.A.; Morales-Ortiz, D.; Fernández, S.; Urriza, B.; Valverde-López, J.; Marín, A.; et al. A Comparison of Feature Extractors for Panorama Stitching in an Autonomous Car Architecture. In Proceedings of the IEEE International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE), Cuernavaca, Mexico, 26–29 November 2019.
10. Varghese, J.Z.; Boone, R.G. Overview of Autonomous Vehicle Sensors and Systems. In Proceedings of the International Conference on Operations Excellence and Service Engineering, Orlando, FL, USA, 10–11 September 2015.
11. Beltran, J.; Guindel, C.; Moreno, F.M.; Cruzado, D.; Garcia, F.; De La Escalera, A. BirdNet: A 3D Object Detection Framework from LiDAR Information. In Proceedings of the IEEE International Conference on Intelligent Transportation Systems, Maui, HI, USA, 4–7 November 2018.
12. Chen, X.; Ma, H.; Wan, J.; Li, B.; Xia, T. Multi-View 3D Object Detection Network for Autonomous Driving. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
13. Geiger, A.; Moosmann, F.; Car, O.; Schuster, B. A Toolbox for Automatic Calibration of Range and Camera Sensors Using a Single Shot. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Saint Paul, MN, USA, 14–18 May 2012; pp. 3936–3943.
14. Vora, S.; Lang, A.H.; Helou, B.; Beijbom, O. PointPainting: Sequential Fusion for 3D Object Detection. *arXiv* **2019**, arXiv:1911.10150.
15. Starr, W.; Lattimer, B.Y. A Comparison of IR Stereo Vision and LiDAR for Use in Fire Environments. In Proceedings of the Sensors, 2012 IEEE, Taipei, Taiwan, 28–31 October 2012.
16. Viitaniemi, V.; Laaksonen, J. Techniques for Image Classification, Object Detection and Object Segmentation. Visual Information Systems. Web-Based Visual Information Search and Management, Lecture Notes in Computer Science. In Proceedings of the VISUAL 2008, Salerno, Italy, 11–12 September 2008; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5188.
17. Leonard, J.K. Image Classification and Object Detection Algorithm Based on Convolutional Neural Network. *Sci. Insights* **2019**, *31*, 85–100. [CrossRef]
18. Chen, C.; Qin, C.; Qiu, H.; Tarroni, G.; Duan, J.; Bai, W.; Rueckert, D. Deep Learning for Cardiac Image Segmentation: A Review. *Front. Cardiovasc. Med.* **2020**, *7*, 25. [CrossRef]
19. Tabian, I.; Fu, H.; Khodaei, Z.S. A Convolutional Neural Network for Impact Detection and Characterization of Complex Composite Structures. *Sensors* **2019**, *19*, 4933. [CrossRef] [PubMed]
20. Sabzekar, M.; Ghasemigol, M.; Naghibzadeh, M.; Yazdi, H.S. *Improved DAG-SVM: A New Method for Multiclass SVM Classification*; ICAI: Las Vegas, NV, USA, 2012.
21. Du, Q. Unsupervised Real-Time Constrained Linear Discriminate Analysis to Hyper Spectral Image Classification. *Pattern Recognit.* **2007**, *40*, 1510–1519. [CrossRef]

22. Lecun, Y.; Bottou, L.; Bengio, Y. Gradient-based Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
23. Lee, H.; Grosse, R.; Ranganath, R.; Ng, A.Y. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009; pp. 609–616.
24. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
25. He, K.; Gkioxari, G.; Dollar, P.; Girshick, R. Mask R-CNN. *Facebook AI Res.* **2018**, 1–12.
26. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *arXiv* **2015**, arXiv:1506.02640.
27. Simonyan, J.K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
28. Pascanu, R.; Mikolov, T.; Bengio, Y. On the Difficulty of Training Recurrent Neural Networks. *arXiv* **2012**, arXiv:1211.5063.
29. Choi, K.; Fazekas, G.; Sandler, M.; Cho, K. Convolutional Recurrent Neural Networks for Music Classification. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5–9 March 2017; pp. 2392–2396.
30. Çakır, E.; Parascandolo, G.; Heittola, T.; Huttunen, H.; Virtanen, T. Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2017**, *25*, 1291–1303. [CrossRef]
31. Maddula, R.; Stivers, J.; Mousavi, M.; Ravindran, S.; Sa, V.D. Deep Recurrent Convolutional Neural Networks for Classifying P300 BCI signals. In Proceedings of the 7th Graz Brain-Computer Interface Conference, Graz, Austria, 18–22 September 2017.
32. Zuo, Z.; Shuai, B.; Wang, G.; Liu, X.; Wang, X.; Wang, B.; Chen, Y. Convolutional Recurrent Neural Networks: Learning Spatial Dependencies for Image Representation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 18–26.
33. Hu, Z.; Hu, Y.; Liu, J.; Wu, B.; Han, D.; Kurfess, T. A CRNN module for hand pose estimation. *Neurocomputing* **2019**, *333*, 157–168. [CrossRef]
34. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
35. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014.
36. Mirza, M.; Osindero, S. Conditional Generative Adversarial Nets. *arXiv* **2014**, arXiv:1411.1784.
37. Odena, A. Semi-supervised Learning with Generative Adversarial Networks. *arXiv* **2016**, arXiv:1606.01583.
38. Odena, A.; Olah, C.; Shlens, J. Conditional Image Synthesis with Auxiliary Classifier GANs. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.
39. Ali-Gombe, A.; Elyan, E.; Savoye, Y.; Jayne, C. Few-shot classifier GAN. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018.
40. Yong, F.; Tanfeng, S.; Xinghao, J.; Ke, X.; Paisong, H. Robust GAN-Face Detection Based on Dual-Channel CNN Network. In Proceedings of the 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Suzhou, China, 19–21 October 2018; pp. 1–5.
41. Ali-Gombe, A.; Elyan, E.; Jayne, C. Fish Classification in Context of Noisy Images. *Eng. Appl. Neural Netw.* **2017**, *CCIS 744*, 216–226.
42. Elyan, E.; Jamieson, L.; Ali-Gombe, A. Deep learning for symbols detection and classification in engineering drawings. *Neural Networks* **2020**, *129*, 91–102. [CrossRef]
43. Coulouris, G.; Dollimore, J.; Kindberg, T.; Blair, G. *Distributed Systems Concepts and Design*; Addison-Wesley: Boston, MA, USA, 2012.
44. Usman, M.; Anjum, A.; Farid, M.; Antonpoulos, N. Cloud-Based Video Analytics Using Convolutional Neural Networks. *Softw. Pract. Exp.* **2019**, *49*, 565–583.
45. Teerapittayanon, S.; McDanel, B.; Kung, H.T. Distributed Deep Neural Networks over the Cloud, the Edge and End Devices. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017.
46. Khandelwa, R. COCO and Pascal VOC Data Format for Object Detection. Towards Data Science. Available online: https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5 (accessed on 17 October 2020).
47. Zeng, N. An Introduction to Evaluation Metrics for Object Detection. Available online: https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/ (accessed on 17 October 2020).
48. El Aidouni, M. Evaluating Object Detection Models: Guide to Performance Metrics. Available online: https://manalelaidouni.github.io/manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html (accessed on 17 October 2020).
49. Werling, M.; Ziegler, J.; Kammel, S.; Thrun, S. Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenét Frame. In Proceedings of the 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 3–7 May 2010.
50. Zhou, Y.; Tuzel, O. VoxelNet End-to-End Learning for Point Cloud Based 3D Object Detection. *arXiv* **2017**, arXiv:1711.06396.
51. Zhou, Y.; Tuzel, O. MVX-Net: Multimodal VoxelNet for 3D Object Detection. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.

52. Liu, T.; Liao, Q.; Gan, L.; Ma, F.; Cheng, J.; Xie, X.; Wang, Z.; Chen, Y.; Zhu, Y.; Zhang, S.; et al. Hercules: An Autonomous Logistic Vehicle for Contact-less Goods Transportation During the Covid-19 Outbreak. *arXiv* **2020**, arXiv:2004.07480.
53. Moraes, G.; Mozart, A.; Azevedo, P.; Piumbini, M.; Cardoso, V.B.; Oliveira-Santos, T.; De Souza, A.F.; Badue, C. Image-Based Real-Time Path Generation Using Deep Neural Networks. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020.
54. Cardoso, V.B.; Oliveira, A.S.; Forechi, A.; Azevedo, P.; Mutz, F.; Oliveira-Santos, T.; Badue, C.; De Souza, A.F. A Large-Scale Mapping Method Based on Deep Neural Networks Applied to Self-Driving Car Localization. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020.
55. Weaver, C. Self-Driving Cars Learn to Read the Body Language of People on the Street, IEEE SPECTRUM. Available online: https://spectrum.ieee.org/transportation/self-driving/selfdriving-cars-learn-to-read-the-body-language-of-people-on-the-street (accessed on 17 October 2020).
56. Aranzeta-Ojeda, L.; Moreno-García, C.F.; Granados-Reyes, A.; Bustamante-Bello, R. Design, Development and Testing of a Low-Cost, High Sensitivity System for Neurodegenerative Disease Detection and Characterization. In Proceedings of the International Conference on Microtechnologies and Medical Biology (MMB), Lucerne, Switzerland, 4–6 May 2011; pp. 64–65.
57. Bustamante-Bello, R.; Aranzeta-Ojeda, L.; Moreno-Garcia, C.F. Design and Development of a Low-Cost, High Sensitivity Device for Neurodegenerative Disease Detection. In Proceedings of the 24th IEEE International Conference Micro Electro Mechanical Systems (MEMS), Cancun, Mexico, 23–27 January 2011.
58. Barea, R.; Bergasa, L.M.; Romera, E.; López-Guillén, E.; Perez, O.; Tradacete, M.; López, J. Integrating State-of-the-Art CNNs for Multi-Sensor 3D Vehicle Detection in Real Autonomous Driving Environments. In Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; Volume 1.
59. Shah, N.; Shankar, A.; Park, J.-h. Detecting Drivable Area for Autonomous Vehicles. *arXiv* **2020**, arXiv:1911.02740.
60. Meng, X.; Lee, K.K.; Xu, Y. Human Driving Behavior Recognition Based on Hidden Markov Models. In Proceedings of the 2006 IEEE International Conference on Robotics and Biomimetics, Kunming, China, 17–20 December 2006.
61. Curiel-Ramirez, L.A.; Ramirez-Mendoza, R.A.; Bautista-Montesano, R.; Bustamante-Bello, M.R.; Gonzalez-Hernandez, H.G.; Reyes-Avedaño, J.A.; Gallardo-Medina, E.C. End-to-End Automated Guided Modular Vehicle. *Appl. Sci.* **2020**, *10*, 4400. [CrossRef]
62. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2018.
63. Hoel, C.J.; Wolff, K.; Laine, L. Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018.
64. El Sallab, A.; Abdou, M.; Perot, E.; Yogamani, S. Deep Reinforcement Learning framework for Autonomous Driving. *Electron. Imaging* **2017**, *2017*, 70–76. [CrossRef]
65. Bai, Z.; Cai, B. Deep Learning-Based Motion Planning for Autonomous Vehicle Using Spatiotemporal LSTM Network. *arXiv* **2019**, arXiv:1903.01712.
66. Atlché, F.; de la Fortelle, A. An LSTM Network for Highway Trajectory Prediction. *arXiv* **2018**, arXiv:1801.07962.
67. Yan, S.; Teng, Y.; Smith, J.; Zhang, B. Driver behavior recognition based on deep convolutional neural networks. In Proceedings of the 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Changsha, China, 13–15 August 2016; pp. 636–641.
68. Naqvi, R.A.; Arsalan, M.; Rehman, A.; Rehman, A.U.; Loh, W.-K.; Paul, A. Deep Learning-Based Drivers Emotion Classification System in Time Series Data for Remote Applications. *Remote. Sens.* **2020**, *12*, 587. [CrossRef]
69. Hu, Y.; Lu, M.; Lu, X. Feature refinement for image-based driver action recognition via multi-scale attention convolutional neural network. *Signal Process. Image Commun.* **2020**, *81*, 115697. [CrossRef]
70. Kim, C.-M.; Hong, E.J.; Chung, K.; Park, R.C. Driver Facial Expression Analysis Using LFA-CRNN-Based Feature Extraction for Health-Risk Decisions. *Appl. Sci.* **2020**, *10*, 2956. [CrossRef]
71. Liu, S.; Tang, J.; Wang, C.; Wang, Q.; Gaudiot, J.L. Implementing a Cloud Platform for Autonomous Driving. *arXiv* **2017**, arXiv:1704.02696.
72. Kumar, S.; Gollakota, S.; Katabi, D. A Cloud-Assisted Design for Autonomous Driving. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC)*; ACM: Helsinki, Finland, 2012.
73. Li, X.; Zhang, G.; Li, K.; Wang, Z. Chapter 4: Deep Learning and Its Parallelization. In *Big Data: Principles and Paradigms*; Morgan Kauffman: Burlington, MA, USA, 2016.