

STATE UNIVERSITY OF MARINGÁ
CENTER OF TECHNOLOGY
DEPARTMENT OF INFORMATICS
POSTGRADUATE PROGRAM IN COMPUTER SCIENCE

MASTER'S DISSERTATION

MAX NAEGELER ROECKER

Vehicle detection and classification in traffic images using
convolutional neural networks

Maringá

2019

MAX NAEGELER ROECKER

Vehicle detection and classification in traffic images using
convolutional neural networks

Master's dissertation presented to the
Postgraduate Program in Computer Science
of the Department of Informatics, Center
of Technology of the State University of
Maringá as requisite for obtainment of
Master of Science in Computer Science.
Concentration area: Computer Science.

Advisor: Ph.D. Yandre Maldonado e Gomes
da Costa

Maringá
2019

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá - PR, Brasil)

R711v

Roecker, Max Naegeler

Vehicle detection and classification in traffic images using convolutional neural networks / Max Naegeler Roecker. -- Maringá, PR, 2019.
61 f.: il. color.

Orientador: Prof. Dr. Yandre Maldonado e Gomes da Costa.
Dissertação (Mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2019.

1. Redes neurais convolucionais. 2. Detecção de veículos. 3. Visão computacional. 4. Aprendizado de máquina. I. Costa, Yandre Maldonado e Gomes da, orient. II. Universidade Estadual de Maringá. Centro de Tecnologia. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 23.ed. 006.32

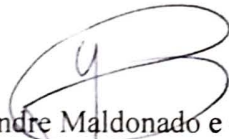
APPROVAL SHEET

MAX NAEGELER ROECKER

Vehicle detection and classification in traffic images using convolutional neural networks

Dissertation submitted to the Graduate Program in Computer Science of the Department of Informatics, Center of Technology at the State University of Maringá, as a partial requirement for obtaining the Master degree in Computer Science by the examination board composed of the following members:

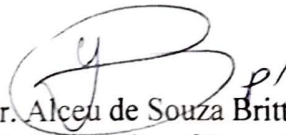
EXAMINATION BOARD



Prof. Dr. Yandre Maldonado e Gomes da Costa
State University of Maringá – DIN/UEM



Prof. Dr. Diego Bertolini Gonçalves
Federal Technological University of Paraná– DACOM/UTFPR-CM



Prof. Dr. Alceu de Souza Britto Júnior
Pontifical Catholic University of Paraná – PPGIa/PUCPR
participation by videoconference

Approved on: September 18, 2019.

Place of defense: Room 101, Block C56, *campus* of the State University of Maringá.

AGRADECIMENTOS

Agradeço a Deus por ter me abençoado nesta caminhada e as pessoas que contribuíram na realização deste trabalho. Em especial:

A minha família pelo carinho, apoio e incentivo.

Ao meu orientador professor Dr. Yandre Maldonado e Gomes da Costa pelo apoio, comentários e sugestões no desenvolvimento deste projeto.

Aos colegas e amigos que compartilharam essa jornada comigo e ajudaram de alguma forma no desenvolvimento do trabalho.

Aos demais professores das disciplinas que cursei e do conhecimento repassado.

E a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concedido a este trabalho.

ABSTRACT

Vehicle detection and classification is a fundamental component in intelligent traffic systems. In the last years, with the decreasing costs of digital images acquisition by surveillance cameras, the proposal of methods that uses digital images as the main source of information have been increased. This work proposes two convolutional neural networks backbones – α and ω – to be used with the YOLO method for detection and classification of vehicles present in digital images into six categories: Bus, Microbus, Minivan, Sedan, SUV, and Truck. Experimental results with the BIT-Vehicle Dataset (Dong et al., 2015) reports values 93.20% and 91.24% of the standard mean average precision for the models α and ω , respectively. The tests in three distinct environments exhibit the inference latency is always under one second in both models. We conclude that the model is discriminative and capable of generalizing the patterns of the vehicle type detection and classification task while not requiring expensive computational resources. These features suggest that the model can be useful in the development of embedded intelligent traffic systems, improving accuracy and decision latency.

Keywords: Vehicle detection; Convolutional Neural Networks; Machine learning; Computer vision.

RESUMO

A detecção e classificação de veículos é um componente fundamental para sistemas inteligentes de tráfego. Nos últimos anos, com o declínio do custo de aquisição de imagens por meio de câmeras de vigilância, a proposta de métodos que utilizam imagens digitais como principal fonte de informação aumentou consideravelmente. Este trabalho propõe duas configurações de redes neurais convolucionais – α e ω – para serem utilizadas junto ao método YOLO na detecção de veículos presentes em imagens digitais além da classificação em seis categorias: Ônibus, Microônibus, Minivan, Sedan, SUV e Caminhão. Resultados experimentais com o conjunto de dados BIT-Vehicle (Dong et al., 2015) reportam valores de 93.20% e 91.24% de precisão média entre as categorias para os modelos equipados com α e ω , respectivamente. Testes realizados em três ambientes distintos exibem que uma latência de inferência sempre abaixo de um segundo em ambos os modelos. Conclui-se que o modelo é discriminativo e capaz de generalizar padrões para a detecção e categorização de tipos de veículos ao mesmo tempo que não requer muitos recursos computacionais. Essa característica sugere que o modelo pode ser útil para o desenvolvimento de sistemas inteligentes de tráfego, melhorando a acurácia e a latência na tomada de decisão.

Palavras-chave: Detecção de veículos; Redes neurais convolucionais; Aprendizado de máquina; Visão computacional.

FIGURE LISTING

Figure - 1.1	Some vehicle categories, when viewed in a frontal perspective, can have high similarity as can be seen between the SUV (a) and Sedan (b) or in between Truck (c) and Minivan (d).	13
Figure - 2.1	Object detection methods can be classified under in three main groups: presence detection, boundaries detection and semantic detection. In this image, we depict examples of the method input and output of each group.	18
Figure - 2.2	The three cases of inferences when evaluating an object boundaries detection with IoU: a true positive (a), a false positive (b) and a false negative (c).	19
Figure - 2.3	An example of a 2-D convolution without kernel flipping. The output consists of the positions where the kernel lies entirely in the input. The arrows indicate the upper-left element of the output tensor. Source: Goodfellow et al. (2016, p. 330)	21
Figure - 2.4	The sparse connectivity as viewed by the highlighted input x_3 and the affected output units. (<i>Top</i>) When the output is formed by a convolution of a kernel of width 3, only three output units are connected with the input x_3 . (<i>Bottom</i>) When the output is formed by a matrix multiplication, all the output units are connected to x_3 . Source: Goodfellow et al. (2016, p. 331)	23
Figure - 2.5	The receptive field of deeper layers of a convolutional neural network is larger than the receptive field of shallow layers. The effect increases if the network includes features like strided convolution or pooling. Thus, even though direct connections are sparse, units in deeper layers can indirectly connect to all or most of the original input. Source: Goodfellow et al. (2016, p. 330)	23
Figure - 2.6	Comparison of the parameters usage between standard neural networks and convolutional neural networks. (<i>Top</i>) The highlighted arrows indicates the shared central parameter in a convolutional neural network with a kernel with width of 3. (<i>Bottom</i>) The single arrow indicates the parameter used only one time in the output of a standard neural network, with no parameter sharing. Source: Goodfellow et al. (2016, p. 333)	24

Figure - 2.7	Common terminology used to describe a convolutional neural network layer. Adapted from Goodfellow et al. (2016, p. 336) . . .	25
Figure - 2.8	The behavior of some activation functions near the origin.	27
Figure - 2.9	The application of the max-pooling function with a (2,2) neighborhood's size and a stride of (2,2) into a 4×4 input results in a 2×2 output with maximum values of each neighborhood.	27
Figure - 2.10	Residual layer building block. Source: He et al. (2016)	28
Figure - 3.1	The mapping of the output of the YOLO to euclidian coordinates.	33
Figure - 3.2	A simplified schema of the YOLO method. It divides an input image into an g^2 grid cells and for each grid it infers a boundaries with confidences along c categories probabilities. The output is given by the filtered detections multiplied by the categories. Adapted from Redmon et al. (2016)	34
Figure - 4.1	Some of the samples after the resizing and position into a canvas. There are positive (a-f) and negative (g-i) samples. The canvas is filled with an uniform distribution before the sample is positioned into it.	41
Figure - 5.1	Precision \times Recall Curves of the model equipped with the backbone α and with $t = 0.5$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f). . .	47
Figure - 5.2	Precision \times Recall Curves of the model equipped with the backbone α and with $t = 0.75$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f). . .	48
Figure - 5.3	Precision \times Recall Curves of the model equipped with the backbone ω and with $t = 0.5$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f). . .	49
Figure - 5.4	Precision \times Recall Curves of the model equipped with the backbone ω and with $t = 0.75$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f). . .	50

TABLE LISTING

Table - 3.1	Summary of the main results on the vehicle detection and classification task reported in the literature.	37
Table - 4.1	Architecture of Backbone α	39
Table - 4.2	Architecture of Backbone ω	39
Table - 4.3	Category distribution over the dataset partitions.	42
Table - 5.1	Average Precision of the YOLO models equipped with α and ω	46
Table - 5.2	Test Environments for Inference Latency.	51
Table - 5.3	Inference latency of the YOLO models equipped with α and ω	51
Table - 5.4	Average Precision of the SSD models equipped with α and ω	52
Table - 5.5	Inference latency of the SSD models equipped with α and ω	53
Table - 5.6	The yielding of SSD and YOLO models equipped with α	53
Table - 5.7	The yielding of SSD and YOLO models equipped with ω	53

SUMMARY

1	Introduction	11
1.1	Context and challenges	12
1.2	Approaches and limitations	13
1.3	Objectives, contributions and organization	14
1.4	Final considerations	15
2	Theoretical foundations	16
2.1	Object detection	16
2.2	Convolutional neural networks	20
2.3	Final considerations	29
3	Related works	30
3.1	Object detection with convolutional neural networks	30
3.2	Vehicle detection and classification	35
3.3	Final considerations	37
4	Proposed Model	38
4.1	Model's architecture	38
4.2	Dataset and preprocessing	40
4.3	Model's training	42
4.4	Final considerations	44
5	Results and discussion	45
5.1	Accuracy results	45
5.2	Inference latency results	51
5.3	Comparison against alternative methods	52
5.4	Final considerations	54
6	Final considerations and future works	55
6.1	Publications	56
	References	57

Introduction

The usage of digital cameras in traffic surveillance has been growing in the last years mostly because of the decreasing costs of this hardware device. Fixed traffic cameras acquire a large amount of vehicles' frontal view images every day and can provide a rich source of information for intelligent traffic systems at a relatively low cost. However, in order to properly take advantage of the data collected by the cameras, it is necessary to extract information from the images with reasonable latency. The recognition of vehicles presented in a scene captured by these cameras is useful information and can empower, for example, traffic density estimators, router's optimizers, and detectors of traffic's violation.

Object recognition is a core problem in computer vision and usually is separated into two main tasks: object detection and object classification. Humans glance at an image and almost instantly can localize, identify and delimit the objects presented on it by using key aspects such as trademarks, forms, and ornaments. For computer systems, however, this task can be challenging mostly because digital image inputs have high-dimensional features (Szeliski, 2010) and the acquisition of images in the traffic is also subject to surrounding conditions such as lighting, noising, partial occlusion, and weather. In this sense, fast and accurate algorithms for object detection are quite desirable, because they can unlock the potential for the development of responsive systems.

In current times, traffic play a vital role in people lives. The growth of the number of vehicles is one of the effects of quicken pace of urbanization, especially in developing countries. This amount of vehicles can cause severe issues such as traffic jams and congestion which leads to air contamination and increase in fuel burning. Along these lines, traffic management and control administration, especially in the field of speed measurement should be noticed as an important subject for authorities.

1.1 Context and challenges

The use of Intelligent Transportation Systems (ITS) has gained an extensive interest in current years. An ITS approach the traffic problem by using transportation guidelines united with the available technologies. Instruments installed in roadways, like inductive surveillance cameras, loop detectors and microwave detectors, are the first act to collect data to bolster an ITS. Between them, cameras are a common equipment due its low-cost installation and maintenance. These cameras are commonly placed in a high exposure location to provide suitable data for traffic monitoring. But the main point is that the data collected by the cameras can be stored as video and afford the advantage of using computer vision analysis and techniques. In this context, the video captured by the camera can be used as the input of the system that process the each image frame with algorithms to extract vehicles information. Such system is called a video-based system hence it extracts information from videos.

Even though video-based systems have advantages when compared with its alternatives, they face many different kinds of challenges. Video-based systems are susceptible to environment conditions like weather or illumination which may result in less accuracy and reliability. Furthermore, when regarding different orientation, vehicles come in different shapes and sizes since there are multiple vehicle categories. This can be seen as an advantage hence these feature discriminate each vehicle category. Although, the cameras are usually placed in locations where the most common perspective is the frontal view of the vehicle.

The frontal view of the vehicles imposes a challenge when comparing similar vehicle categories. Different vehicle categories, such as sedans and SUVs, when viewed from a frontal perspective, can be very similar and sometimes indistinguishable from each other. If the detection system is performs a classification of vehicles using their appearance features some vehicles may be misclassified in this context. Figure - 1.1 display some cases of high similarity between vehicles when viewed from a frontal perspective.

Another challenge concerning the vehicle detection task is the lack of standard evaluation methodology. There are a wide range of datasets reported by the research community claiming its usage on the vehicle detection task, but is common that the datasets are over adapted to specific circumstances and do not generalize enough. Furthermore, the inherent distribution of vehicle categories also stands as a challenge on the datasets and this unbalance can affect the accuracy of the methods considering some metrics. All of these issues make difficult to directly compare the results of works.

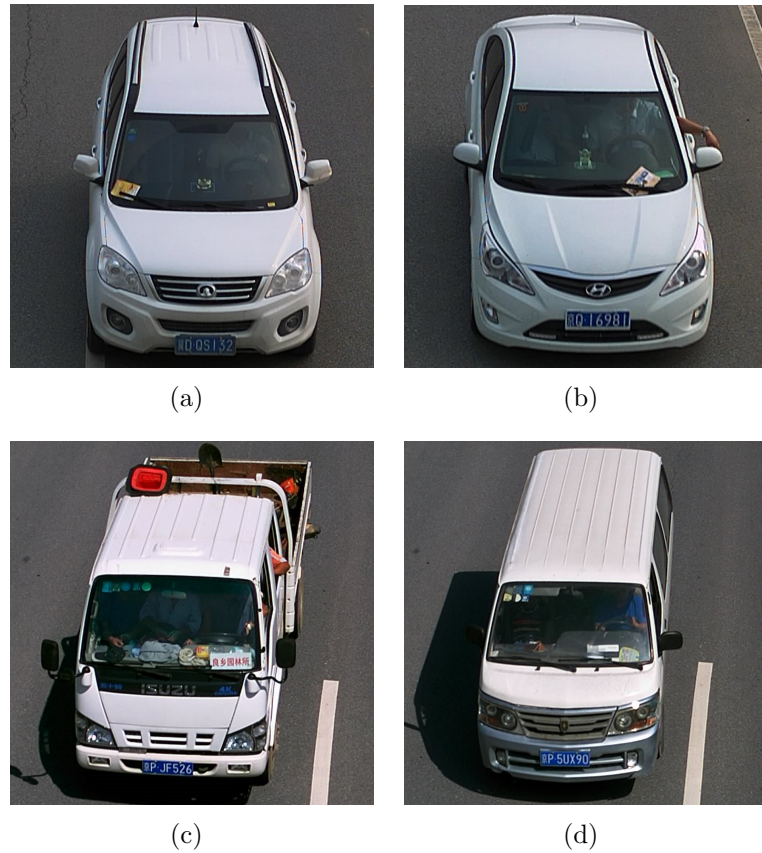


Figure 1.1: Some vehicle categories, when viewed in a frontal perspective, can have high similarity as can be seen between the SUV (a) and Sedan (b) or in between Truck (c) and Minivan (d).

1.2 Approaches and limitations

In the last years, several works proposed methods for vehicle type recognition in digital images. Dong et al. (2015) group these approaches in two categories: model-based methods and appearance-based methods. Model-based methods address the problem by using dimensional attributes of the vehicle, such as length, area, and height, to create a model (Gupte et al., 2002; Zhang et al., 2012). On the other hand, appearance-based methods address the problem by extracting visual features from the vehicle, such as edges, filters and visual descriptors (Ji et al., 2007; Jiang and Li, 2014).

In these circumstances, the use of an appearance-based method is a better alternative, since the model-based method may perform poorly due to the lack of variance of perspectives in viewpoints. However, most of the appearance-based methods use multiple handcrafted features, which cannot efficiently describe the complexity of the patterns for vehicle type classification in images.

In the last decades, methods inspired on the biological behavior of the mammal's visual cortex were proposed, including the NeoCognitron (Fukushima, 1980), the HMAX (Hierarchical Model and X (Serre et al., 2007)) and the CNN (Convolutional Neural Networks (Lecun, 1989)). Convolutional neural networks are a specialized kind of neural network for processing data that have spatial interactions and have gained prominence recently due to its high capacity to generalize patterns in images. This characteristic is useful for the vehicle type classification because it minimizes the problems previously mentioned.

The field of object recognition has seen a tremendous progress the increasing employment of CNN approaches (Girshick, 2015; Girshick et al., 2014; He et al., 2017; Huang et al., 2017; Lin et al., 2017; Ren et al., 2017), providing state-of-art results on well-known datasets such as Microsoft COCO (Lin et al., 2014b) or Pascal VOC (Everingham et al., 2015). These models are well suitable for usage in a resource plentiful environment such as computer servers equipped with powerful graphics processing units (GPU). When employed in a resource-constrained environment, these models can have a high latency and its usage can be prohibitively for a near real-time application.

Unfortunately, in the context of vehicle detection and classification for intelligent traffic systems, a local plentiful computational resource environment is usually difficult to maintain due to the climatic hazards and deploying in remote locations. Some systems propose to transmit the images captured by cameras to servers specially dedicated to processing the images, but there are downfalls in this approach as the system relies on the network infrastructure and there is inherent latency associated. Models that do not require expensive computational resources and can be locally deployed are desired since these costs and decision latency are minimized while the maintainability and the fault-tolerance are increased.

1.3 Objectives, contributions and organization

The present master's degree dissertation has the main objective to propose two models to detect and classify vehicles in images presented in the frontal view. As secondary objectives we want to test the models in different environments, to evaluate the accuracy and inference latency with constrained resources. We also aim to compare the models with alternatives proposed by the research community regarding its accuracy and the inference latency.

The development of the research has applied nature, with exploratory purposes and quantitative approach to measuring the models' performance and inference latency at

well-known datasets. The main contributions of the present work is: (i) the development of models that are able of detect and classify multiple vehicles present in digital images in a frontal-view perspective; (ii) an evaluation and discussion of the proposed model regarding its accuracy and inference time and; (iii) a comparison between the proposed model with similar methods presented by the research community.

Among the contributions, two papers were published. The first, entitled “Automatic vehicle type classification with convolutional neural networks”¹, proposed a convolutional neural network to classify vehicles in low resolution images in a frontal-view perspective. The second paper, entitled “Vehicle detection and classification in traffic images using ConvNets with constrained resources”², is the continuation of the first paper and proposed a convolutional neural model to detect and classify vehicles in images and evaluated the results concerning the accuracy and the inference time. Both papers were published on the International Conference on Systems, Signals and Image Processing (IWSSIP) in the years of 2018 and 2019, respectively.

This document is organized as follows: Chapter 2 presents the theoretic foundations of the research, detailing the object detection task and convolutional neural networks. Next, in Chapter 3 we detail related works of general object using convolutional neural networks and vehicle detection with multiple approaches. We present and describe the models proposed along with the methodology used by this work in Chapter 4. Chapter 5 contains the results of the experimental results regarding accuracy and inference latency. Final considerations and future works are presented in Chapter 6. The references of this work is listed at the end.

1.4 Final considerations

In this chapter, we introduced the computer vision’s problem of object detection and scrutinize the application of its methods in the context of Intelligent Transportation Systems. We addressed the challenges of the problem and also described some approaches along with its drawbacks. Next, we defined the primary and secondary objectives this work and its contributions to the research community. We also described the organization of the work, summarizing the next chapters of the document.

In Chapter 2, we will go deeper into the theoretical fundamental concepts to better comprehend this work, detailing the field of the object detection research and the convolutional neural networks.

¹Roecker et al. (2018)

²Roecker et al. (2019)

Theoretical foundations

In this chapter we'll approach the theoretical foundations of this work. Firstly, in Section 2.1 we describe the computer vision problem of object detection, defining some concepts and evaluation methodology. Next, in the Section 2.2, we will approach the convolutional neural networks more deeply, explaining its formal definition, properties and differences between standard neural networks. Lastly, in Section 2.3, we have some final considerations of the chapter.

2.1 Object detection

The main goal of the computer vision is to create and develop methods that take an image as input and produce a significant interpretation describing objects and actions along with the semantic information. Our visual system is able to carry out such task with little effort. Humans can detect and recognize objects from a library of thousands if not tens of thousands in very complex scenes. However, the goal of developing computational methods for these tasks is still a newly research area with relative success in the last few years. **Object detection** is, in general terms, the process of discovery of an object within an image (Szeliski, 2010) and can be grouped under three classes: presence detection, boundary detection and semantic detection.

Presence detection methods performs the simplest form of an object detection by receiving an image as input and outputting, normally along with a probability, if an set of objects is present in the image. In this type of method, the entire image can be classified in one or more categories. Presence detection is similar to image classification, but the

method must be prepared to receive as input multiple kinds and variations of objects and images that does not includes any desirable object at all. Figure - 2.1(a) illustrates an example of presence detection.

Methods in the second group, called **boundary detection**, not only outputs the probability of the presence of an set of objects in a image, but also outputs dimensional coordinates of the region of the image in which the object is located. These coordinates are called the boundaries of the object and can be used to segment the object from the rest of the image. In this type of method, regions of the images can be classified in one or more categories. Figure - 2.1(b) shows an method that performs a boundary detection in a sample image.

Methods in the last group, called **semantic detection**, perform the most exhaustive form of detection. In a semantic detection, each unit of the image (a pixel) can be classified in only one of the available categories. Methods in this group, formally, maps each pixel into a category and a object can be located in clusters of equally classified pixels. These methods usually take into consideration some semantic information – and hence its name – to correctly infers and detect objects that are partially occluded or composed by non contiguous pixel clusters. In Figure - 2.1(c) we depict an semantic detection method.

Each group of object detections has its own specific evaluation metrics and methodology. However, in the development of this work, we will describe a model grouped under the boundaries detections. Thus, in the next paragraphs, we will detail and formally describe the concepts and metrics used when evaluating these kind of methods.

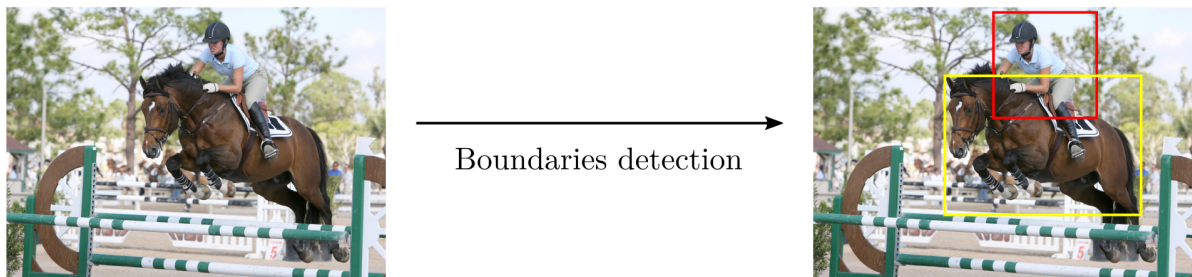
A fundamental concept when evaluating object’s boundaries detection is the function of **Intersection over Union** (IoU). The IoU, also formally known as the Jaccard similarity coefficient, is a function used to compare similarity between two finite sets. In the context of object’s boundaries detection, the IoU evaluates the overlap between two bounding boxes. Let \mathbf{b}_l be the ground truth bounding box and \mathbf{b}_p the predicted bounding box, the IoU of \mathbf{b}_l and \mathbf{b}_p is defined as in Equation 2.1.

$$\text{IoU}(\mathbf{b}_l, \mathbf{b}_p) = \frac{\text{area}(\mathbf{b}_l \cap \mathbf{b}_p)}{\text{area}(\mathbf{b}_l \cup \mathbf{b}_p)} \quad (2.1)$$

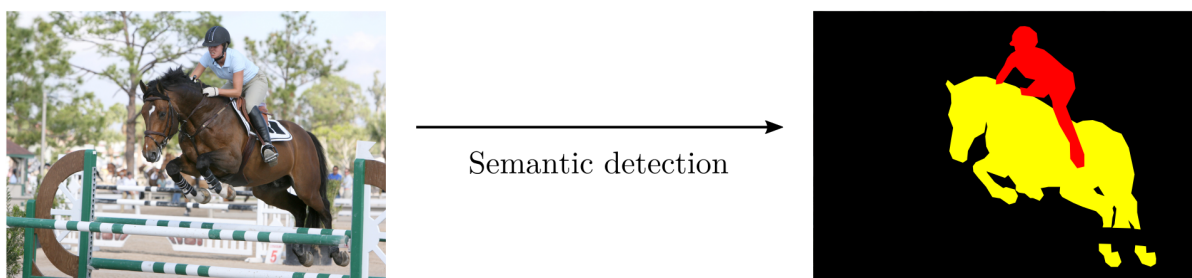
When applying the IoU, we can define if the detection is a true positive (TP), false positive (FP) or a false negative (FN). Let $t \in \mathbb{R}$ be a threshold value, a **true positive** is a correct detection of the model where $\text{IoU}(\mathbf{b}_l, \mathbf{b}_p) \geq t$. A **false positive** is a wrong detection of the model where $\text{IoU}(\mathbf{b}_l, \mathbf{b}_p) < t$. A **false negative** is a case where the ground truth \mathbf{b}_l is not predicted by the model. The true negative case does not apply to the object’s boundaries detection context because it would represent a correct misdetection.



(a) Presence detection is similar to image classification with multiple categories. In this example, we illustrate the input and the output of presence detection method. The output represents the probability of each category be present in the image.



(b) Boundaries detection encloses the region of image that contains the object and label it with a category. Usually, this boundaries are defined by two coordinates that represents a rectangular region where the object is located. In this example, we portray the input and the output of boundaries detection method. The output shows the object's rectangular boundaries detected: one label as a person (red) and other label as a horse (yellow).



(c) Semantic detection maps each pixel of the image into a category. As a result, we have an image with clusters of pixels that are equally labelled that can be inferred as objects. Along with some semantic information, objects partially occluded or composed of multiple clusters can be detected. In this example, the input and the output of semantic detection method are illustrated. The output shows the object's detected: one object composed of one cluster labelled as person (red) and another object composed of multiple clusters labelled as horse (yellow).

Figure 2.1: Object detection methods can be classified under in three main groups: presence detection, boundaries detection and semantic detection. In this image, we depict examples of the method input and output of each group.

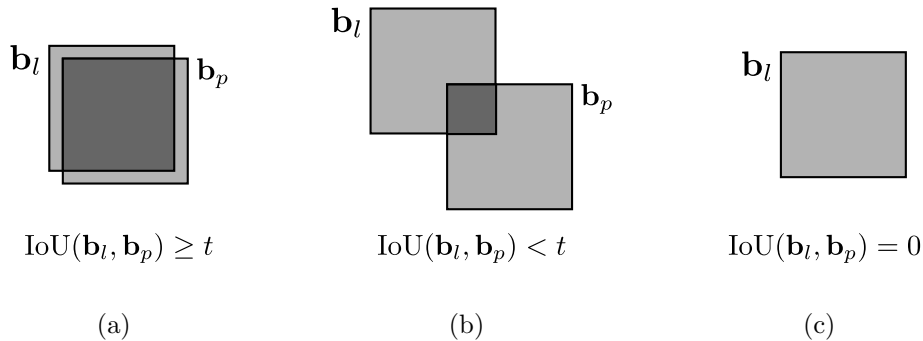


Figure 2.2: The three cases of inferences when evaluating an object boundaries detection with IoU: a true positive (a), a false positive (b) and a false negative (c).

To measure the information retrieved by an object detection method, we need to define metrics that can be extracted from the model and then prescribe an evaluation. **Precision** and **Recall** metrics can be applied in the context of object detection in images. In this context, the precision expresses the percent of how many objects selected by the model are relevant and is formally defined by the ratio between true positives and all detections. In addition, the recall expresses the percent of how many relevant objects were selected by the model and is formally defined as the ratio between true positives and all ground truth bounding boxes.

With the metrics defined we can analyze object detection models. **Precision** \times **Recall** **curve** (PR-curve) is a method that can portray the quality of an object detection model. An model – for some particular category – is considered good if its precision stays high as recall increases, i.e., if we increment the threshold t value, the precision and recall of a good model should not drastically drop. A poor model needs to increase the number of the detected objects – and thus increasing the number of false positives – in order to retrieve all ground truth objects. This justifies why PR-curves usually have decreasing precision values as the recall increases.

Since the PR-curve often has many fluctuations, use it to compare detectors is not an easy task. So, to compare models, we use the **average precision** (AP) across all recall values between 0 and 1. This provides an concise metric that ables comparison between the models. Accordingly to Everingham et al. (2015), the Pascal VOC Challenge defines the AP metric as interpolation of all data points as in Equation 2.2, where $p(\tilde{r})$ is the measured precision at recall \tilde{r} .

$$\sum_{r=0}^1 (r_{n+1} - r_n) p_i(r_{n+1}) \quad \text{with} \quad p_i(r_{n+1}) = \max_{\tilde{r}: \tilde{r} \geq r_{n+1}} p(\tilde{r}) \quad (2.2)$$

All the metrics previously described are relative to one single category of the method. When the method outputs multiple categories, a common single numeric value used to evaluate the model is the mean of all categories AP metrics (mAP). Most of the evaluation standards set $t = 0.5$, but in the last years, with recent increase in the accuracy of the methods, $t = 0.75$ is considered a standard value (Everingham et al., 2015; Lin et al., 2014b).

2.2 Convolutional neural networks

Introduced by Lecun (1989), a **convolutional neural network** (CNN) is a specialized kind of neural network that employs a convolution in place of ordinary matrix multiplication in at least one of its steps (Goodfellow et al., 2016). The **convolution** is an operation on two functions of a real-valued argument that produces a third function. Formally, a convolution of $f : \mathbb{R} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$, written as $f * g$, is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a particular kind of integral transform defined as Equation 2.3

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (2.3)$$

In the convolutional network terminology, the first argument, f , is named to as the **input**, and the second argument, g , as the **kernel** or **filter**. The output is defined as the **feature map**. Since in machine learning methods the input is usually a tensor — high dimensional generalizations of matrices represented as multidimensional arrays — of the features values; the kernel is also a tensor of parameters that are adjusted by the optimization algorithm. Thus, the convolution of an input $\mathbf{I} \in \mathbb{R}^{p \times q}$ and a kernel $\mathbf{K} \in \mathbb{R}^{r \times s}$ is defined as in Equation 2.4.

$$(\mathbf{I} * \mathbf{K})_{i,j} = \sum_{n=1}^p \sum_{m=1}^q \mathbf{I}_{m,n} \mathbf{K}_{i-m,j-n} \quad (2.4)$$

The convolution is commutative; i.e., it is equivalent to Equation 2.5; which is habitually more straightforward to implement because there is less variation in the range of values of m and n since the kernel is commonly much smaller than the input.

$$(\mathbf{I} * \mathbf{K})_{i,j} = (\mathbf{K} * \mathbf{I})_{i,j} = \sum_{n=1}^r \sum_{m=1}^s \mathbf{I}_{i-m,j-n} \mathbf{K}_{m,n} \quad (2.5)$$

The commutative property of the convolution arises because the kernel was flipped, which can be useful to write proofs but is usually not important in a neural network implementation. Thus, it is common for many implementations to use a related function called **cross-correlation**, which is the same as convolution but without kernel flipping. In the context of machine learning, it is rare for a convolution to be used alone, and normally is used with other functions and combinations, which does not commute regardless of whether the kernel is flipped or not. Figure - 2.3 illustrates an example of a convolution (without kernel flipping) applied to a two-dimensional tensor.

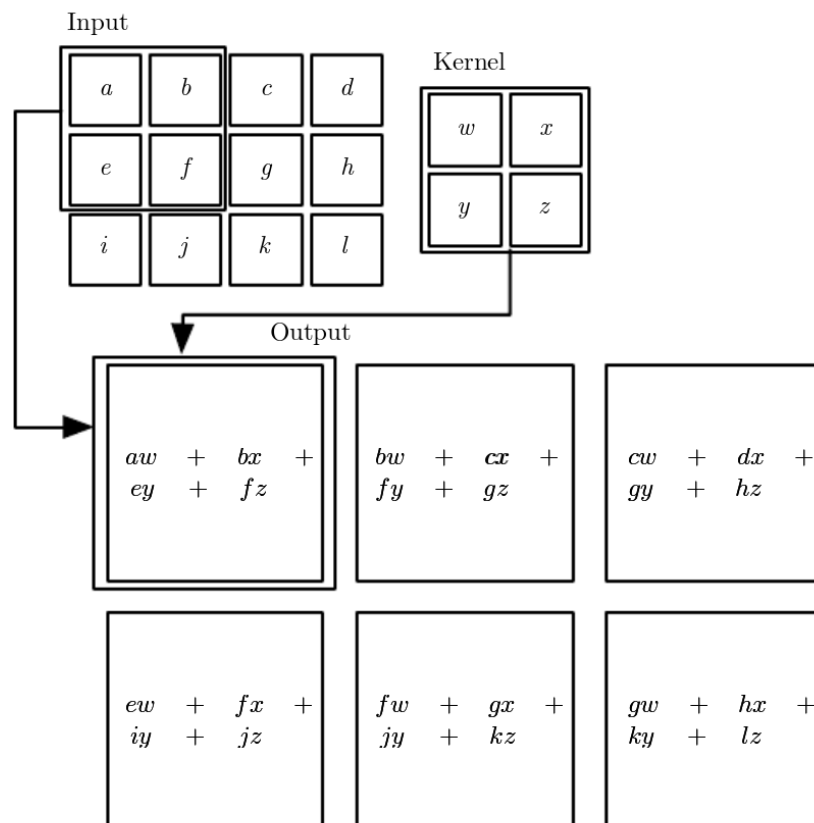


Figure 2.3: An example of a 2-D convolution without kernel flipping. The output consists of the positions where the kernel lies entirely in the input. The arrows indicate the upper-left element of the output tensor. Source: Goodfellow et al. (2016, p. 330)

According to Goodfellow et al. (2016), the use of convolution leverages three important concepts that can help to improve the induction algorithm when processing data with spatial interactions: **sparse interactions**, **parameter sharing**, and **equivariant representations**.

Traditionally, standard neural networks, also called fully-connected neural networks, use a matrix multiplication of an input and the parameters to describe the interaction of each input element and each output element. Thus, every output element interacts with every input unit, creating a dense interaction between them. Alternatively, convolutional networks have **sparse interactions** (or sparse connectivity). For example, when processing an image, the input might have millions of units (in this case, pixels), but the network can detect small and meaningful features such as edges with kernels that occupy only hundreds of pixels. This means that we need to store fewer parameters, which reduces memory requirements of the model and improves statistical efficiency. The improvements in efficiency are quite large. If there are m inputs and n outputs, a matrix multiplication requires $m \times n$ parameters, and the algorithms usually have an $O(m \times n)$ runtime. If the number of connections is limited as each output may have k units, the sparsely connected operation requires only $k \times n$ and $O(k \times n)$ runtime. In practical applications, it is possible to obtain a value for k that is several orders of magnitude smaller than m . Figure - 2.4 illustrates the sparse connectivity from the perspective of a input unit of a convolutional neural network when comparing with the dense connectivity of a standard neural network.

Even if the output unit is not directly connected to all the input units, when more layers of convolutions are employed, the deeper units may indirectly interact with a broader portion of the original input, as can be seen in Figure - 2.5. This allows the network to efficiently describe complicated interactions between the units by constructing it from simple ones.

Parameter sharing refers to the use of the same parameter for more than one function in a model. In traditional neural networks, each parameter element is used one time with the input to compute the output and never revisited. On the other hand, in the convolutional networks, each unit of the kernel is applied many times in all units of the input (except perhaps in the boundary of the image).

The parameter sharing used by the convolution operation means that rather learning the parameters of each unit for every location, the network learns only one set. This does not improve the runtime propagation – it is still $O(k \times n)$ – but reduces the requirement of storage of parameters to k , and k is very small when compared to $m \times n$. Figure - 2.6 illustrates a comparison of the sharing of the parameters between standard neural networks and a convolutional neural network.

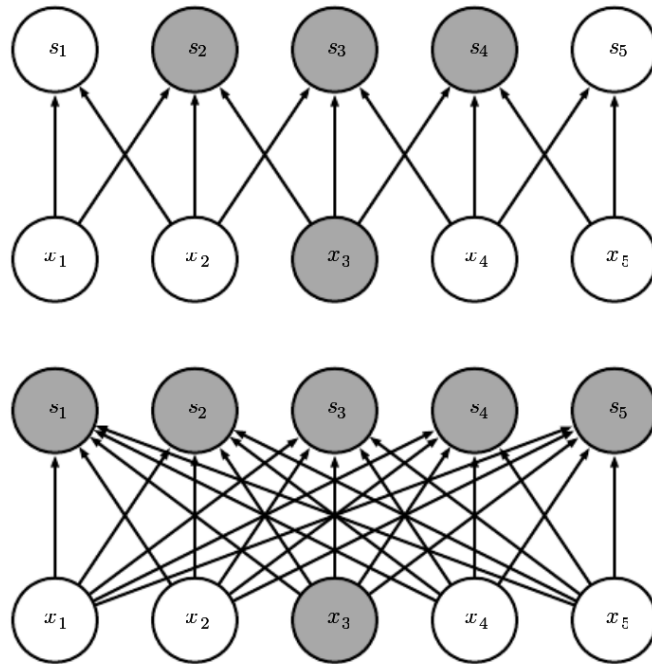


Figure 2.4: The sparse connectivity as viewed by the highlighted input x_3 and the affected output units. (*Top*) When the output is formed by a convolution of a kernel of width 3, only three output units are connected with the input x_3 . (*Bottom*) When the output is formed by a matrix multiplication, all the output units are connected to x_3 . Source: Goodfellow et al. (2016, p. 331)

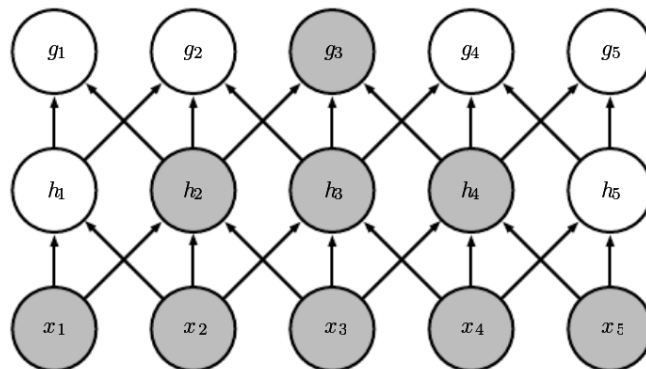


Figure 2.5: The receptive field of deeper layers of a convolutional neural network is larger than the receptive field of shallow layers. The effect increases if the network includes features like strided convolution or pooling. Thus, even though direct connections are sparse, units in deeper layers can indirectly connect to all or most of the original input. Source: Goodfellow et al. (2016, p. 330)

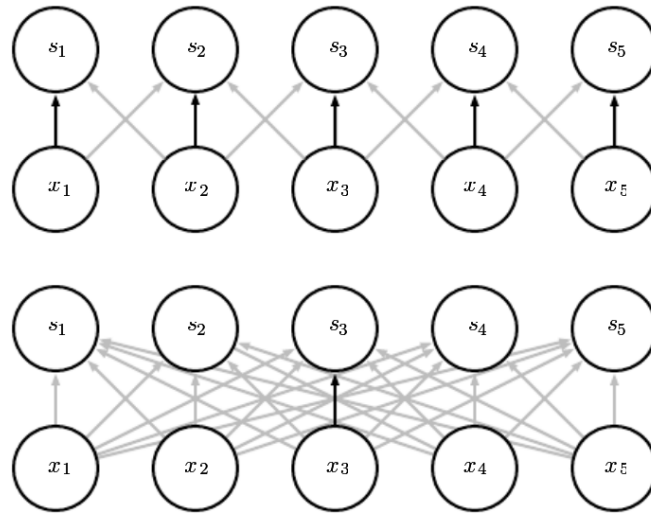


Figure 2.6: Comparison of the parameters usage between standard neural networks and convolutional neural networks. (*Top*) The highlighted arrows indicates the shared central parameter in a convolutional neural network with a kernel with width of 3. (*Bottom*) The single arrow indicates the parameter used only one time in the output of a standard neural network, with no parameter sharing. Source: Goodfellow et al. (2016, p. 333)

Due to the parameter sharing, the convolution has the property of being **equivariant to translation**. A function $f : X \rightarrow Y$ is said to be equivariant to a function $g : X \rightarrow Y$ if $f(g(x)) = g(f(x))$. Let c be a convolution of an input i and a kernel k , i.e., $c(i, k) = i * k$. Also, let t be a function that shifts the input i by some constant value a , i.e., $t(x) = x + a$. In this case, c is equivariant to t , because $c(t(i), k) = t(c(i, k))$. When processing time-series data, this means that the convolutions can detect a feature that is placed later or earlier than the original without penalty. In a two dimensional space, this means that the convolution can detect a trait even when it is translated in the image. The convolution is not naturally equivariant to other transformations, such changes in the scale or rotation.

Typically, a convolutional neural network layer consists of three stages main stages – the **convolutional stage**, the **detector stage** and the **pooling stage** – and optional **shortcut connections**. The Figure - 2.7 illustrates an feed-forward diagram with the interactions of each stage. The first stage, called **convolutional stage**, performs convolutions of the input units and one or more kernels to produce a set of one or more feature maps, each feature map corresponding to a kernel. This is a affine transformation does not adds non-linearity to the representation.

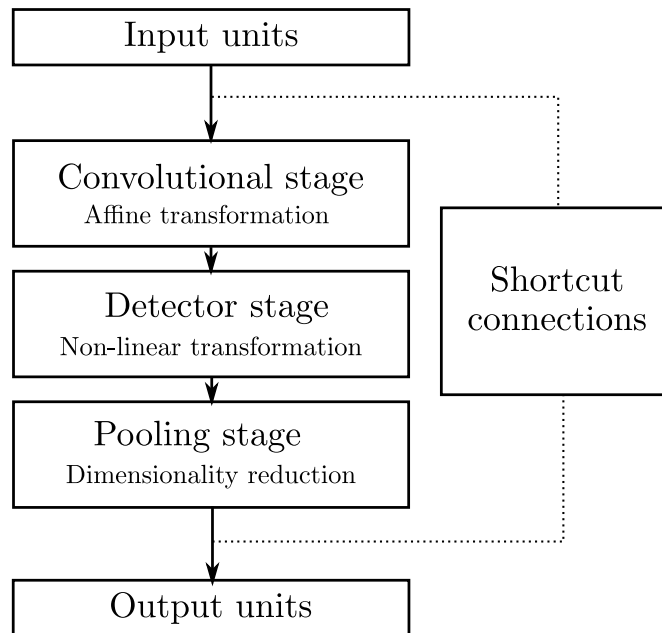


Figure 2.7: Common terminology used to describe a convolutional neural network layer. Adapted from Goodfellow et al. (2016, p. 336)

In the second stage, called **detector stage**, each unit of the feature maps is transformed by a **activation function**. Accordingly to Goodfellow et al. (2016), a desirable activation function is non-linear, continuously differentiable, and monotonic. A two-layer network can be proven to be a universal function approximator if it is equipped with a non-linear activation function (Cybenko, 1989). A continuously differentiable function enables gradient-based optimization methods. A monotonic function guarantees that the error surface associated with a single model is convex (Wu, 2009).

Regarding activation function, specific properties are also desired in its range. An activation function with a finite range tends to stabilize the gradient on training because the pattern's presentations significantly affect only limited weights of the parameters. In contrast, with an infinite range, the training is usually more efficient because pattern's presentations significantly affect most of the weights but requires smaller learning rates. An approximate identity behavior near the origin is also desirable, because the neural network can learn efficiently when it is initialized with small random values. There are several activation functions proposed by the research community with distinct results in each use case, the choice of an activation function is usually an hyperparameter (Goodfellow et al., 2016).

The **sigmoid** function $\sigma : \mathbb{R} \rightarrow [0, 1]$, defined as $\sigma(x) = 1/(1+e^{-x})$, has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: zero (0) when the neuron is not firing and one (1) when the neuron is fully-saturated. As can be seen in Figure - 2.8(a), the sigmoid function is non-linear, continuously differentiable and monotonic, but has fallen out of favor because it super-saturates the gradients when training with back-propagation and does not approximate identity behavior near the origin.

The **hyperbolic tangent** function $\tanh(x) : \mathbb{R} \rightarrow [-1, 1]$, defined as $\tanh(x) = (e^{2x}-1)/(e^{2x}+1)$, like the sigmoid function, has a similar interpretation when the neuron is firing and is also non-linear, continuously differentiable and monotonic. Although, as can be seen in Figure - 2.8(b) the range of the tanh function is centered on zero and approximates identity behavior near the origin. Ergo, in practice, the tanh is always preferred to the sigmoid.

The **rectified linear unit** $\text{ReLU} : \mathbb{R} \rightarrow \mathbb{R}_0^+$ is a non-linear monotonic function that has become very popular in the last years due its good results concerning the accuracy results and the convergence time during training. Firstly introduced by Hahnloser et al. (2000), it is defined as $\text{ReLU}(x) = \max(0, x)$ and has multiple desired properties like having a better gradient propagation and an efficient computation (Glorot et al., 2011; Krizhevsky et al., 2012). Figure - 2.8(c) illustrates the behavior of the ReLU function near the origin.

As demonstrated by Maas et al. (2013), neural networks employing ReLU functions often suffers from the “dying ReLU” problem where a gradient flowing through a ReLU-equipped neuron could cause its weights to update in such a way that the neuron will never activate on any input again. The proportion of “dead” neurons can be up to 40% of the entire model. The “**leaky**” **variation** of the rectified linear unit, $\text{LReLU} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, defined as $\text{LReLU}(\iota, x) = \max(\iota x, x)$, attempts to minimize this problem by adding a hyperparameter ι to create a small negative slope, as can be seen in Figure - 2.8(d).

The third stage of the convolutional neural network layer, called **pooling stage**, applies a **pooling function** that can replace each unit of the feature map as a summary statistic of nearby units. This behavior helps on preventing over-fitting by providing an abstract form of the representation whilst optionally reducing the dimensionality of the output. There are a wide range of pooling functions available, but in general we can categorize each pooling function as a spatial pooling or a cross-filter pooling.

A **spatial pooling** summarizes an dimensional neighborhood area of the input and subsamples it based on a prior. Formally, let (x, y) be the size of the neighborhood, the

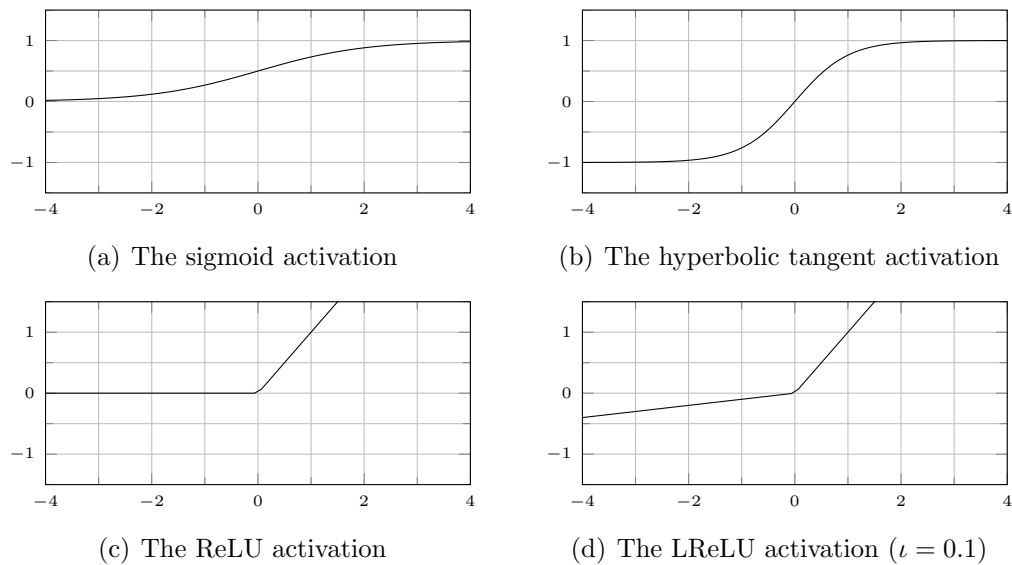


Figure 2.8: The behavior of some activation functions near the origin.

pooling operation reports one unit for every (x, y) units thus improving the statistical efficiency of the model because the next layer has roughly (x, y) times fewer inputs to process. For example, let the input $\mathbf{I} \in \mathbb{R}^{4 \times 4}$ be applied by a **max-pooling** (Zhou and Chellappa, 1988) with a $(2, 2)$ sized neighborhood. As illustrated in Figure - 2.9, this pooling results in a output with all the maximum values of each non-overlapping neighborhood of the image.

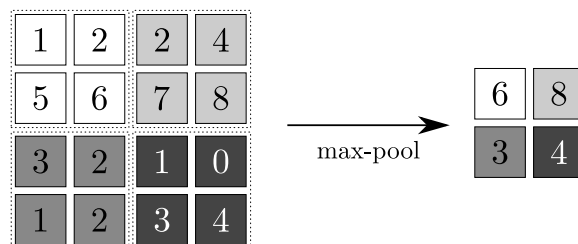


Figure 2.9: The application of the max-pooling function with a $(2, 2)$ neighborhood's size and a stride of $(2, 2)$ into a 4×4 input results in a 2×2 output with maximum values of each neighborhood.

Other popular spatial pooling functions includes the average of a rectangular neighborhood and the weighted average based on the distance from the neighborhood center. Regardless its popularity, the use of spatial pooling is not required to create a convolutional neural layer. Springenberg et al. (2015), for example, demonstrated that there is no accuracy penalty on using a $(2, 2)$ stridden convolution instead a $(2, 2)$ sized max-pooling.

On the other hand, a **cross-filter pooling** summarizes units in the same coordinates of various distinct inputs into small number of outputs and is usually obtained by using “one-by-one” convolutions. Firstly introduced by Lin et al. (2014a), a “one-by-one” convolution acts like a coordinate-dependent transformation in the filter space that is strictly affine, but is usually equipped with a non-linear activation. For example, let $\mathbf{I} \in \mathbb{R}^{x \times y \times z}$ be a feature map where x and y are spatial dimensions and z are number of previously applied convolutional filters. The convolution and application of an activation function of \mathbf{I} with $\mathbf{K} \in \mathbb{R}^{1 \times 1 \times z'}$ results in new feature map $\mathbf{I}' \in \mathbb{R}^{x \times y \times z'}$. Since we choose $z' < z$, this type of pooling adds non-linearity to the representation whilst reducing dimensions only of the filter space of previous layers and helping a faster computation.

Aside the main stages, convolutional layers can employ **shortcut connections** that transforms the output with independent operations. **Residual layers** are popular instances of shortcut connections that demonstrated good results when training very deep convolutional neural networks. As demonstrated by He et al. (2016), residual layers relieves the saturation of the feature maps and thus decreasing the over-fitting and increasing the accuracy of the network. Formally, consider the underlying mapping as $\mathcal{H}(\mathbf{I})$, we let the stacked nonlinear layers fit another mapping of $\mathcal{F}(\mathbf{I}) = \mathcal{H}(\mathbf{I}) - \mathbf{I}$, so the original mapping is recast to $\mathcal{F}(\mathbf{I}) + \mathbf{I}$. As can be seen in Figure - 2.10, the formulation of $\mathcal{F}(\mathbf{I}) + \mathbf{I}$ is be implemented as shortcut connection that that skip one or more layers to transfer the residual parameters by adding an input identity with the output of the main stages.

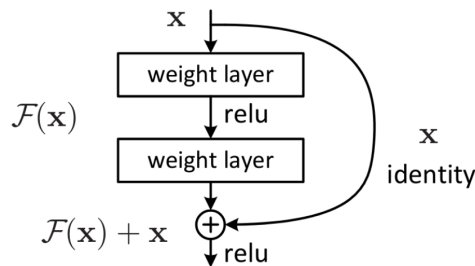


Figure 2.10: Residual layer building block. Source: He et al. (2016)

In general terms, convolutional neural networks can be viewed as standard fully connected neural nets, but with an strong prior over the weights. This prior states that the weights of one hidden unit must be identical to the weights of its neighbor but shifted in space. The prior also states that the weights must be zero for all except an small and spatially contiguous receptive fields assigned to the hidden unit.

As any other neural network, CNNs can be reused into different tasks after its training. This process, called **transfer learning**, usually involves saving the core parameters, often called **backbone**, and apply it with repurposed inputs and outputs. This leverages the sharing of the network, contributing to its longevity and decreasing the training time in the new task.

2.3 Final considerations

In this chapter, we described the theoretical fundamental concepts to comprehend this ongoing work. In the first section, we wrote about the research field of general object detection in the computer vision's area. We reported its types and categories along with evaluation metrics. Then, in the next section, we detailed and formalized the elements of the convolutional neural networks, describing its properties and advantages when compared with standard neural networks.

Coming up, in Chapter 3, we will summarize the related works proposed by the scientific research community to approach the object detection and its application in the detection and classification of vehicles presented in digital images.

Related works

In this chapter, we review works of the research community approaching the object detection problem. In Section 3.1, we describe an historic overview of recent methods approaching the object detection problem using convolutional neural networks, highlighting the YOLO method due its focus on the development of this work. Next, in Section 3.2, we introduce related works approaching the vehicle detection and classification in digital images, highlighting the methodology, the dataset and results achieved by the models. Lastly, with Section 3.3, we have some final considerations about the chapter.

3.1 Object detection with convolutional neural networks

Traditional object recognition algorithms usually perform detection by using a classifier. In these systems, the classifier is evaluated with image descriptors in various regions of the image with different scales (Felzenszwalb et al., 2010; Viola and Jones, 2001). Still, in the last years, the computer vision research community developed other approaches employing convolutional neural networks with state-of-art results.

Girshick et al. (2014) introduced the **Region-based Convolutional Neural NetworkFast** (R-CNN) as a method to perform object detection consisting of three steps: (i) the method scans the input image for possible objects with a selective search (Felzenszwalb and Huttenlocher, 2004), generating near 2000 regions of interest (RoI); (ii) each RoI is transformed by a pre-trained convolutional neural network that outputs a feature map; (iii) each RoI's feature map is inputted into: (a) an SVM to classify the RoI and; (b) a linear regressor that tightens the bounding box of the RoI, if such RoI is classifiable. In

simple words, the method search for RoIs, extract features via a CNN and classify those regions based on the extracted features. In essence, the object detection is turned into a object classification. The R-CNN method achieved good results – 53.3% mAP on the Pascal VOC 2010 when is equipped with the AlexNet backbone (Krizhevsky et al., 2012) – although the training is expensive and slow and also presents a high inference latency (40-50 seconds).

To improve the results of the R-CNN method, Girshick (2015) proposed a unification of three different models into a one trained network, increasing the shared computation. This method was called **Fast R-CNN**. Instead of extracting a feature map with a CNN from each RoI, the proposed model extracts a feature map of the entire image with a CNN and then performs the selective search on the feature map. In addition, the Fast R-CNN method also replaces the SVM classifier with a standard neural network, thus extending the neural network capabilities direct into the predictions. The Fast R-CNN model achieved significant accuracy results by simplifying its architecture – 65.7% mAP on the Pascal VOC 2012 when equipped with the VGGNet-16 backbone (Simonyan and Zisserman, 2015) – while also improved significantly the training and inference latency (cutting to 2-3 seconds).

In the following years, Ren et al. (2017) proposed the **Faster R-CNN** method with an intuitive speedup solution: use a Region Proposal Network (RPN) instead the slow selective search algorithm with the principal CNN of the method. The RPN is a dedicated neural network that shares some of its convolutional layers with the primary CNN. The last layer of the primary CNN is forked in the process, and then a small sliding window generates multiple possible regions based on a $k \in \mathbb{Z}$ fixed-ratio anchor boxes – default bounding boxes previously defined – that are scored with a probability of having a object within and the coordinates of the boundaries. If an anchor box has an probability of having an object greater than a threshold, the region is outputted. Although the RPN outputs the boundaries coordinates, it does not try do classify any potential objects: it is only responsible for proposing the object regions. The final step of the method is identical to the Fast R-CNN method: classify each region using the a standard neural network. The Faster R-CNN achieved better results than the previous methods — 73.2% mAP on the Pascal VOC 2012 when equipped with the VGGNet-16 backbone (Simonyan and Zisserman, 2015) – and limiting the inference latency to near 200 milliseconds.

The R-CNN, Fast R-CNN and Faster R-CNN are object detection methods based on the same principle: hypothesize object regions and then classify them. Alongside with these methods, the research community also introduced another approaches not

fundamented in region proposal, like the Single Shot Detector and the You Only Look Once.

The **Single Shot Detector** (SSD), proposed by Liu et al. (2016), is a method that uses only a single deep CNN to perform the detection and classification of the objects in the image. Instead of proposing regions and classifying them, this method produces both the coordinates and classes with a single convolutional neural network, following: (i) pass the input image through a series of convolutional layers, yielding multiple feature maps in different scales. (ii) For each feature map, like the Faster R-CNN, uses an anchor boxes to evaluate the boundaries and; (iii) For each anchor box, simultaneously predicts (a) the bounding box offset and (b) the class probabilities. The SSD is more straightforward but its training requires additional efforts to remove the multiple negative detections considered by the model. To mitigate this behavior, the SSD first remove multiple overlapping detections using a non-maximum suppression and maintain an 1:3 ratio between positive and negative detections. The SSD has a good performance in terms of accuracy – 72.4% mAP on the Pascal VOC 2012 when equipped with the VGGNet-16 backbone (Simonyan and Zisserman, 2015) – and inference latency under 20 milliseconds per image.

The **You Only Look Once** (YOLO) method was the first attempt to develop an real-time object detector using convolutional neural networks. It was firstly proposed by Redmon et al. (2016), with following updates in Redmon and Farhadi (2017) and Redmon and Farhadi (2018). The method does not employ a region proposal and only detects a limited number of objects per image, straight yielding the coordinates of the boundaries and the categories of each object.

To understand the YOLO method, we need firstly understand how it encodes the detections. Firstly, the input image pass through a series of convolutional layers and outputs a feature map. Typically, in a object detector, the feature map outputted by the convolutional stage serves as input to a classifier/regressor which infers the coordinates of the detected object’s boundary and its category. In YOLO, this step is also made by an convolutional layer which only employs 1×1 convolutions. The complete feature map is a tensor $\mathbf{t} \in \mathbb{R}^{g^2 \times a \times (5+q)}$, where g^2 is the number of grid cells, a is the number of anchor boxes and q is the number of categories that can be inferred by the model.

The YOLO method partitions the input image into g^2 **grid cells** to reduce the search space. Each one of the g^2 grid cells has a boundaries that represents log-spaced offsets relative to a predefined boxes, called **anchors**. Therefore, for each boundary j of each grid cell i , the feature map $\mathbf{t}_{ij} = \langle t_x, t_y, t_w, t_h, t_o, t_{p1}, \dots, t_{pq} \rangle$ encodes **four spatial coordinates, one confidence score** and q **probabilities** for each category.

Suppose the input image with dimensions $l \times l$, partitioned into nine grid cells ($g = 3$) and with one anchor box prior with dimensions (a_w, a_h) , Figure - 3.1 presents an example to decode the feature map \mathbf{t} in the fifth grid cell ($i = 5$). The euclidean centroid coordinates of the detection (b_x, b_y) are encoded as normalized values relative to the grid size (l/g) padded by the grid coordinates $(\mathbf{c}_{ix}, \mathbf{c}_{iy})$. The euclidian dimensions of the detection (b_w, b_h) are encoded as coordinates in a natural logarithm space relative to the corresponding anchor box. The confidence b_o and the probabilities p_n for each category are all decoded by a sigmoid mapping. The usage of coordinates in a log-space along sigmoid applications ensures all units of the feature map \mathbf{t} falls between zero and one, increasing the gradient stability during training.

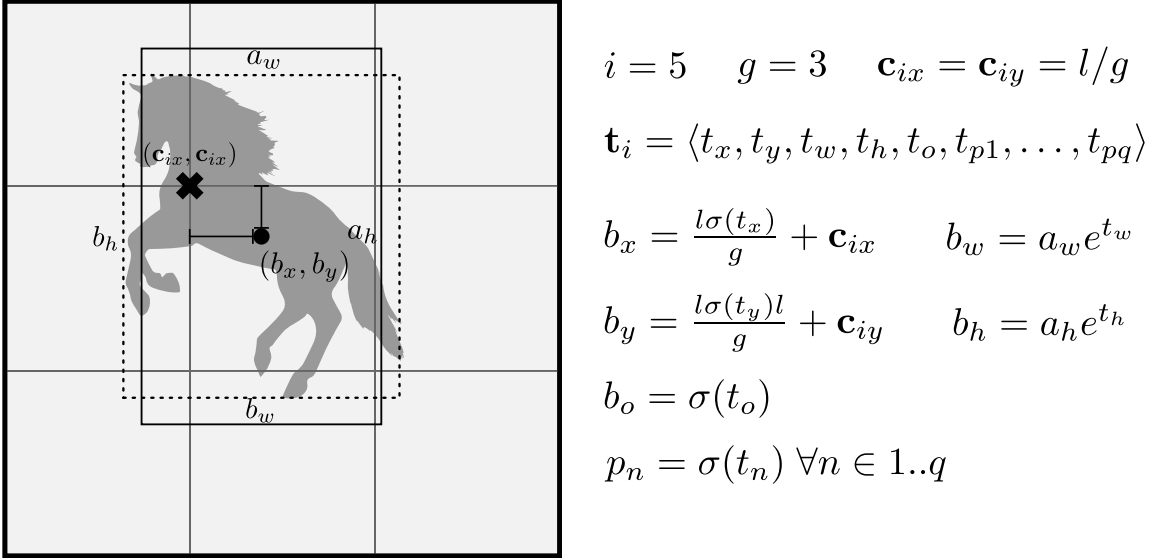


Figure 3.1: The mapping of the output of the YOLO to euclidian coordinates.

Since the YOLO method infers multiple detections per grid cell, the methods define the presence of an object in the cell as \mathbb{O}_i , which evaluates to one if an object is present in the i -th cell or zero otherwise. The method also assign only one anchor box to be responsible for detect some object contained in the cell using the IoU. Thus, the detection related to the anchor with maximum value of IoU with an object in the cell is responsible for detect that object. This strategy leads to specialization among the inferred detections as the model gets better at predicting certain sizes and aspect ratios. This responsibility is formally defined as \mathbb{O}_{ij} , which evaluates as one if the j -th detection of the i -th grid cell is responsible for detection the object or zero otherwise.

The detection's confidence (b_o) is used to indicate the probability of the boundary containing an object. The confidence conditions the final probabilities of the detection, thus the final probability for each $n \in 1..q$ categories is given by $P(p_n | b_o)$. In practice, this conditional probability means that if no object is present on the grid cell, the error function will not penalize the model for an wrong class prediction.

To output the final results, detections with confidence below a threshold are removed. The actual classification of the detection is given by the maximum $P(p_n | b_o)$. Then, overlapping detections with same category are addressed with a non-maximum suppression. Figure - 3.2 presents a simplified schema of the YOLO method as used by Redmon et al. (2016) when evaluated on the Pascal VOC 2012 dataset.

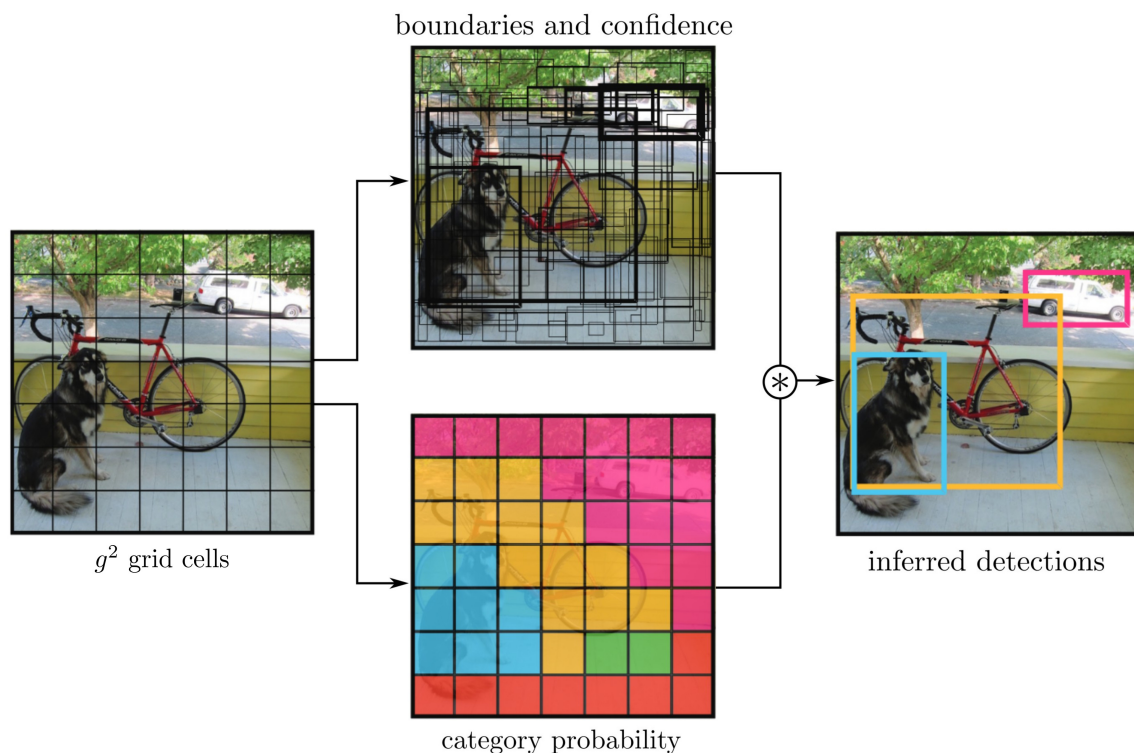


Figure 3.2: A simplified schema of the YOLO method. It divides an input image into an g^2 grid cells and for each grid it infers a boundaries with confidences along c categories probabilities. The output is given by the filtered detections multiplied by the categories. Adapted from Redmon et al. (2016)

The number of anchors and grid cells used by the model are hyperparameters. In Redmon and Farhadi (2017) and Redmon and Farhadi (2018), the authors recommended using 5 anchors and splitting the input image into 7×7 grid cells. The number of convolutional layers in the backbone of the model is very adaptable and, as a rule of thumb, it can be decreased to achieve a faster training and inference latency or increased

to achieve a better inference accuracy. In its first version, Redmon et al. (2016) used a backbone influenced on the AlexNet (Krizhevsky et al., 2012) with 24 convolutional layers followed by 2 fully-connected layers. In its second version, Redmon and Farhadi (2017) used a backbone motivated by the VGG-16 (Simonyan and Zisserman, 2015) with 19 convolutional layers with heavy usage of a cross-filter pooling. In the last version, Redmon and Farhadi (2018) used novel architecture with 53 convolutional layers that makes use of cross-filter pooling and shortcut connections.

3.2 Vehicle detection and classification

In the last years, the task of recognition of vehicles in digital images has received noticeable attention by computer vision research community. Although, because of the absence of a standard framework for the evaluation of the proposed models, there are significant differences in the works and makes it difficult to perform a fair comparison of the obtained results.

Han et al. (2009) presents a vehicle detection approach using Haar-like features of vehicle edges. The method is effective to detect vehicles in the image scene, removing environmental noise and detecting the boundary of the vehicle in the image. As there is only one vehicle class, the model does not performs classification. The model achieved an inference for detection of 66 milliseconds in the testing embedded system.

Dong et al. (2015) propose a vehicle type classification method using a semi-supervised convolutional network from vehicle frontal-view image of the BIT-Vehicle Dataset. The architecture of the model consists of two convolutional stages, and each stage contains a convolution, a non-linearity absolute rectification, a local contrast normalization, and average pooling. The input of the first stage is the image, and the output of the first stage is the input of the second stage. The fully connected stage takes as input the fusion of the outputs of the first and the second stages. In the end, the model outputs the probability of each of the six vehicle types: Bus, Microbus, Sedan, SUV, and Truck. To achieve an accuracy of 88.11%, Dong et al. (2015) also employ a Laplacian Filter to obtain the initial value for the kernels of the network with large amounts of unlabeled data.

Wang et al. (2017) employed a CNN influenced by the Faster R-CNN architecture (Ren et al., 2017) to perform vehicle detection and classification into four categories: Car, Bus, Minivan, and Trucks. The dataset employed to train and test the model was acquired by the authors with traffic surveillance cameras fixed at crossroads. The authors provided a new Region Proposal Network that aims to reduce the inference time of the model, and

they achieved 81.06% mAP. When using an NVIDIA Jetson TK1 with 192 CUDA cores, the authors reported that the model inference latency is approximately 354 milliseconds.

Selbes and Sert (2017) address the vehicle type classification using a multimodal method from videos of traffic scenarios, extracting both image’s features and audio’s features and fusing the features as input to a Support Vector Machine (SVM) multiclassifier. To extract the image-based features, the authors use the trained versions of the renowned convolutional network’s architectures AlexNet (Krizhevsky et al., 2012) and GoogleNet (Szegedy et al., 2015). Mel-frequency Cepstral Coefficients (MFCCs) are used to extract audio-based features from the video. The SVM then classify each video snippet as an armored vehicle, a construction vehicle, a crane vehicle, an emergency vehicle, a military vehicle, a motorcycle, and a rescue vehicle. This multimodal method achieves 72.1% accuracy.

Kim and Lim (2017) propose a new scheme of vehicle type classification for multi-view images of surveillance cameras using convolutional networks with data augmentation, bootstrap aggregating (bagging) and, a post-processing voting between the models of the bagging method. The model consisted of seven independently trained convolutional networks with the same characteristics that output a prediction by voting. Inspired by the works of Simonyan and Zisserman (2015), the authors modeled all the convolutional networks with fifteen convolutional layers. The model was evaluated in a subset of the ImageNet Dataset with eleven classes: articulated truck, background (negative examples), bicycle, bus, car, motorcycle, non-motorized vehicle, pedestrian, pickup truck, single-unit truck and work van; achieving an accuracy of 97.84%.

Şentaş et al. (2018) describe a performance evaluation for the recognition of vehicles in images between two distinct models: HOG+SVM and TinyYOLO. The HOG+SVM model is composed of an Histogram of Gradients (HOG) feature descriptor as the detector and a Support Vector Machine (SVM) as the classifier. The TinyYOLO is a model composed of a CNN constructed as a shallow version of the YOLO architecture (Redmon and Farhadi, 2017) that acts both as detector and classifier. For the task of object detection, the authors evaluated both models on a dataset created by the authors called TPSdataset. In the TPSdataset, each vehicle can be classified on one of the five following categories: Bus, Minivan, Minibus, Truck, and Auto. For the HOG+SVM model, Şentaş et al. (2018) reported the the HOG+SVM model achieved 97.14% of average precision and 87.81% of recall on the TPSdataset. For the TinyYOLO model, the authors reported achieving 62.83% of precision, 86.22% of recall and 69.74% of IoU.

In Table Table - 3.1 we summarize the related works reported by the research community in a historic order. The tables features, in column “Task”, the task that

the proposed work aimed to approach; in column “Method”, the technique employed by the authors and; in column “Performance” the reported result. It’s important to observe that the listed works does not uses the same metrics and are not tested using the same conditions and datasets. Therefore, this table does not aim to direct compare the works, but offer a concise view about the contemporary state of the research field of vehicle’s detection in digital images.

Table 3.1: Summary of the main results on the vehicle detection and classification task reported in the literature.

Work	Task	Method	Performance¹
Dong et al. (2015)	Classification	CNN	88.11%
Wang et al. (2017)	Detection/Classification	Faster CNN	81.06%
Selbes and Sert (2017)	Detection/Classification	CNN + MFCC + SVM	72.10%
Kim and Lim (2017)	Classification	CNN + AdaBoost	97.84%
Şentaş et al. (2018)	Detection/Classification	HOG + SVM	97.14%
Şentaş et al. (2018)	Detection/Classification	TinyYOLO	62.83%

3.3 Final considerations

In Chapter, we reviewed related works in the research field of object detection. In Section 3.1 is summarized the recent approaches to detection objects in digital images using convolutional neural networks, highlighting the accuracy scores in standard evaluation datasets. We also highlighted the YOLO method, detailing its characteristics, encoding and properties. Next, in Section 3.2, we reviewed a historic summary the approaches to detect and classify vehicles present in digital images, highlighting the methodology of the work and the reported results regarding accuracy and inference time.

Coming up, in Chapter 4, we will detail the proposed model describing its architecture, the dataset used in training and evaluation and, lastly, the methodology of the research.

¹Results not necessarily obtained with the same metrics and not even on the same dataset

Proposed Model

In this chapter we'll present the proposed model. Firstly, in Section 4.1, we describe the architecture of the model. In the Section 4.2, we present the dataset and the preprocessing employed in the work. Next, in the Section 4.3, we will detail the training process and methodology used in the model. At Section 4.4 we present some final considerations about the chapter.

4.1 Model's architecture

Aiming to get a good trade-off between accuracy and inference latency when addressing the vehicle detection, we chose to build our own model inspired by the YOLO architecture. To evaluate the accuracy of the model in different conditions, we propose two convolutional backbones for the model: α and ω . The backbone α is a heavy and deep convolutional neural network, it has 53 convolutional layers and its usage is designed to be implanted in a resource plentiful environment. The backbone ω , otherwise, is relatively shallow – with 10 convolutional layers – and can be used in an environment with resource restrictions.

The structure of the backbones α and ω for each image channel is summarized in Table - 4.1 and Table - 4.2, respectively. The column “Input units” displays the number of units that the layer receives and the “Parameters units” presents the number of parameters (weights) in the layer. Both columns are in the “width \times height \times depth” format. The “Stride” column shows the stride of the convolution and pooling operations in the (x, y) axis. As there are multiple layers with equivalent configurations, the “ \times ” column indicates how many times the layers are sequentially repeated.

Table 4.1: Architecture of Backbone α .

\times	Layer type	Input units	Parameters units	Stride
1	Convolutional	$512 \times 512 \times 1$	$32 \times 3 \times 3$	(1,1)
	Convolutional	$512 \times 512 \times 32$	$64 \times 3 \times 3$	(2,2)
	Convolutional	$256 \times 256 \times 64$	$32 \times 1 \times 1$	(1,1)
	Convolutional	$256 \times 256 \times 32$	$64 \times 3 \times 3$	(1,1)
	Residual	$256 \times 256 \times 64$	$256 \times 256 \times 64$	–
	Convolutional	$256 \times 256 \times 64$	$128 \times 3 \times 3$	(2,2)
2	Convolutional	$128 \times 128 \times 128$	$64 \times 1 \times 1$	(1,1)
	Convolutional	$128 \times 128 \times 64$	$128 \times 3 \times 3$	(1,1)
	Residual	$128 \times 128 \times 128$	$128 \times 128 \times 128$	–
1	Convolutional	$128 \times 128 \times 128$	$256 \times 3 \times 3$	(2,2)
8	Convolutional	$64 \times 64 \times 256$	$128 \times 1 \times 1$	(1,1)
	Convolutional	$64 \times 64 \times 128$	$256 \times 3 \times 3$	(1,1)
	Residual	$64 \times 64 \times 256$	$64 \times 64 \times 256$	–
1	Convolutional	$64 \times 64 \times 256$	$512 \times 3 \times 3$	(2,2)
8	Convolutional	$32 \times 32 \times 512$	$256 \times 1 \times 1$	(1,1)
	Convolutional	$32 \times 32 \times 256$	$512 \times 3 \times 3$	(1,1)
	Residual	$32 \times 32 \times 512$	$32 \times 32 \times 512$	–
1	Convolutional	$32 \times 32 \times 512$	$1024 \times 3 \times 3$	(2,2)
4	Convolutional	$16 \times 16 \times 1024$	$512 \times 1 \times 1$	(1,1)
	Convolutional	$16 \times 16 \times 512$	$1024 \times 3 \times 3$	(1,1)
	Residual	$16 \times 16 \times 1024$	$16 \times 16 \times 1024$	–
1	Global Average Pooling			

Table 4.2: Architecture of Backbone ω .

\times	Layer type	Input units	Parameters units	Stride
1	Convolutional	$512 \times 512 \times 1$	$16 \times 3 \times 3$	(2,2)
	Convolutional	$256 \times 256 \times 16$	$32 \times 1 \times 1$	(2,2)
	Convolutional	$128 \times 128 \times 32$	$64 \times 3 \times 3$	(2,2)
	Convolutional	$64 \times 64 \times 64$	$128 \times 3 \times 3$	(2,2)
	Convolutional	$32 \times 32 \times 128$	$256 \times 3 \times 3$	(2,2)
	Convolutional	$16 \times 16 \times 256$	$512 \times 3 \times 3$	(2,2)
1	Convolutional	$8 \times 8 \times 512$	$1024 \times 3 \times 3$	(1,1)
	Convolutional	$8 \times 8 \times 1024$	$256 \times 1 \times 1$	(1,1)
	Convolutional	$8 \times 8 \times 256$	$512 \times 3 \times 3$	(1,1)
	Convolutional	$8 \times 8 \times 256$	$512 \times 3 \times 3$	(1,1)
1	Global Average Pooling			

Motivated by the works of Ciresan et al. (Ciresan et al., 2011), Krizhevsky et al. (Krizhevsky et al., 2012) and Simonyan and Zisserman (Simonyan and Zisserman, 2015), we designed all the stages of the model with the same principles to simplify the model and minimize setup of hyperparameters. Both backbones receives as input a 512 pixel wide and high RGB image. The input pass through a stack of convolutional layers with an increasing number of filters with size 3×3 . We fixed the convolution stride at one point and preserved the spatial padding. The output of each convolutional layer is equipped with the leaky rectifier activation function (LReLU). Both detectors do not employ spatial pooling. Although, they employ a cross feature map pooling by using 1×1 sized convolutions in some layers. α employs residual layers but ω don't use the same technique.

Before the fully-connected stage, the model performs a **Global Average Pooling** (GAP) (Lin et al., 2014a) to minimize the overfitting of the fully-connected stage. In conventional convolutional neural networks, the bridge between the convolutional stage and the fully-connected stage is made by vectorizing the last feature map of the convolutional stage and fed it into the fully-connected stage. The GAP proposes a different approach. First, it calculates the average of each of the last feature maps of the convolutional stage and then the resulting vector is fed into the fully-connected stage. As an example, last feature maps in the the backbone α have $16 \times 16 \times 1024$ units, thus there are 1024 feature maps with a size of 16×16 from which the average is calculated. Therefore, the resulting GAP vector has 1024 units.

4.2 Dataset and preprocessing

The **BIT-Vehicle Dataset** (Dong et al., 2015) is a dataset comprised of 9,850 vehicle images with high resolution (1600×1200 px and 1920×1080 px). The images are in a wide range of changes in illumination, scale, surface color and position of the vehicles. Each image may contain more than one vehicle, and so the dataset also contains the annotation of each bounding box of each vehicle in the image. All the vehicles in the dataset are labeled as one of the following categories: **Bus**, **Microbus**, **Minivan**, **Sedan**, **SUV**, and **Truck**.

To improve the generalization of the model, we also included in the dataset some negative samples, i.e., samples that do not have a vehicle on it. The negative examples were selected from the **ImageNet Dataset** (Russakovsky et al., 2015) and contain images with roads, highways, passages and some landscapes. Each image was resized to fit into 512×512 canvas. To provide more generalization to the model, each canvas is generated by a random uniform distribution and the image is randomly positioned on the canvas.

The resizing was performed using bilinear interpolation. Figure - 4.1 displays the positive and negative samples after the preprocessing step.

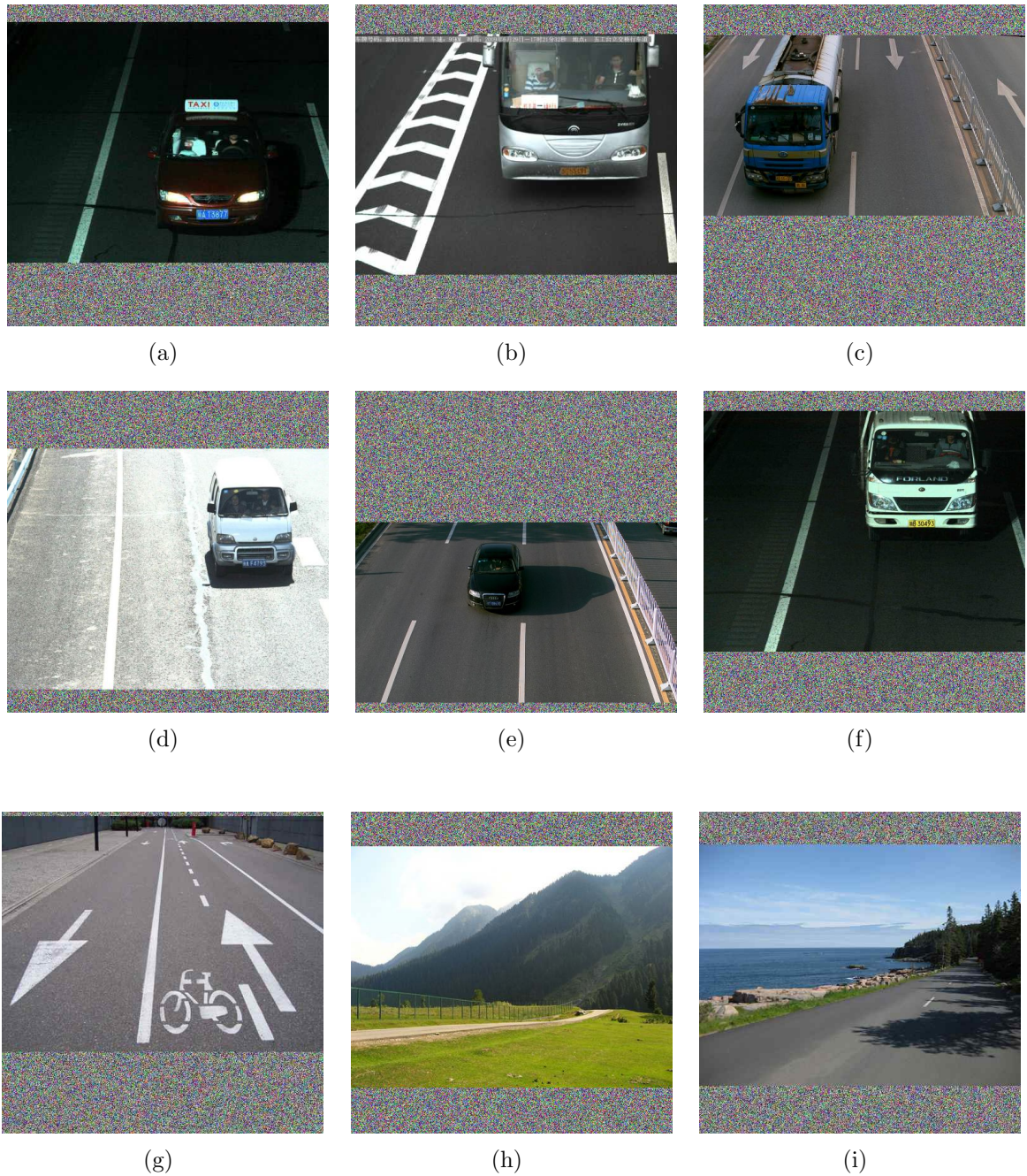


Figure 4.1: Some of the samples after the resizing and position into a canvas. There are positive (a–f) and negative (g–i) samples. The canvas is filled with a uniform distribution before the sample is positioned into it.

The BIT-Vehicle Dataset has an probability distribution of the categories far from uniform, with a coefficient of variation almost equal to 1.27. We have opted for using the partition of the dataset with 70% for training and 30% for testing. Since the samples of the dataset are not uniformly distributed into the categories, different amounts of samples were randomly selected for the training and testing partitions. Table - 4.3 summarizes the distribution of each category into the partitions.

Table 4.3: Category distribution over the dataset partitions.

Category	Training	Testing	%	Total
Bus	390	168	$\approx 5.6 \%$	558
Microbus	618	265	$\approx 8.8 \%$	883
Minivan	333	143	$\approx 4.7 \%$	476
Sedan	4,144	1,777	$\approx 58.9 \%$	5,921
SUV	974	418	$\approx 13.8 \%$	1,392
Truck	576	247	$\approx 8.2 \%$	823
Total	7,035	3,018	100%	10,053

Instead of choosing anchor boxes by hand, we employed the k-means clustering on the training set bounding boxes to find good priors. The k-means clustering algorithm can be adapted to find anchor boxes as we choose $k \in \mathbb{Z}$ random boxes as our initial means \mathbf{b}_i . Then, we assign each ground truth bounding box \mathbf{b}_l to a cluster C_i defined as in Equation 4.1, where $i \in 1 \dots k$.

$$C_i = \{\mathbf{b}_l \mid d(\mathbf{b}_l, \mathbf{b}_i) \leq d(\mathbf{b}_l, \mathbf{b}_j), \forall j \in I, i \neq j\} \quad (4.1)$$

The mean is calculated for all boxes belonging to C_i and this yields a new \mathbf{b}_i and the process is repeated until it converges. If we define d as the Euclidian distance, larger boxes generate more error than smaller boxes (Redmon and Farhadi, 2017). Thus, to make the method independent of the size of the box, we defined d as the Jaccard distance, so $d(\mathbf{b}_l, \mathbf{b}_i) = 1 - \text{IoU}(\mathbf{b}_l, \mathbf{b}_i)$. In the development of this work, we choose $k = 5$ and thus both models have five anchors.

4.3 Model's training

The general method for training both models follows Redmon and Farhadi (2018). The error function, written as ε , is composed as the sum of four terms: ε_c , ε_d , ε_o and ε_q representing the error for detection's coordinates, dimensions, confidence and

classification, respectively. Let \mathbf{b} be the an detected bounding box and $\hat{\mathbf{b}}$ the ground-truth bounding box. The complete definition of ε is denoted in Equation 4.2.

$$\varepsilon(\mathbf{b}, \hat{\mathbf{b}}) = \varepsilon_c(\mathbf{b}, \hat{\mathbf{b}}) + \varepsilon_d(\mathbf{b}, \hat{\mathbf{b}}) + \varepsilon_o(\mathbf{b}, \hat{\mathbf{b}}) + \varepsilon_q(\mathbf{b}, \hat{\mathbf{b}}) \quad (4.2)$$

The error of coordinates ε_c is defined as the mean-squared error of the detection's centroid coordinates, written in Equation 4.3.

$$\varepsilon_c(\mathbf{b}, \hat{\mathbf{b}}) = \frac{\lambda_c}{ag^2} \sum_{i=1}^{g^2} \sum_{j=1}^a \mathbb{O}_{ij} \left[\left(\mathbf{b}_{ijx} - \hat{\mathbf{b}}_{ijx} \right)^2 + \left(\mathbf{b}_{ijy} - \hat{\mathbf{b}}_{ijy} \right)^2 \right] \quad (4.3)$$

The error of dimensions ε_d is defined as the mean-squared error of squared root of the detection's width and height as written in Equation 4.4. The dimension error uses the squared root to reflect that small deviations in large boundaries is less than in small boundaries.

$$\varepsilon_d(\mathbf{b}, \hat{\mathbf{b}}) = \frac{\lambda_c}{ag^2} \sum_{i=1}^{g^2} \sum_{j=1}^a \mathbb{O}_{ij} \left[\left(\sqrt{\mathbf{b}_{ijw}} - \sqrt{\hat{\mathbf{b}}_{ijw}} \right)^2 + \left(\sqrt{\mathbf{b}_{ijh}} - \sqrt{\hat{\mathbf{b}}_{ijh}} \right)^2 \right] \quad (4.4)$$

The error of the confidence score ε_o is a mean-squared error of the inferred confidence with the IoU of the detection with the ground truth and is written in Equation 4.5, where $\overline{\mathbb{O}_{ij}}$ is the complementary of \mathbb{O}_{ij} .

$$\varepsilon_o(\mathbf{b}, \hat{\mathbf{b}}) = \frac{1}{ag^2} \sum_{i=1}^{g^2} \sum_{j=1}^a \left[\mathbb{O}_{ij} \left(\mathbf{b}_{ijo} - \text{IoU}(\mathbf{b}, \hat{\mathbf{b}}) \right)^2 + \lambda_o \overline{\mathbb{O}_{ij}} \left(\mathbf{b}_{ijo} - \text{IoU}(\mathbf{b}, \hat{\mathbf{b}}) \right)^2 \right] \quad (4.5)$$

Lastly, the error for classification ε_q is given by the mean-squared error of the the probabilities of c classes inferred by the model and the ground truth also conditioned by the presence of an object \mathbb{O}_i . The error is formally is written in Equation 4.6, where \mathbf{p}_i and $\hat{\mathbf{p}}_i$ contains, respectively, the probabilities of the predicted and ground truth objects centered in the i -th grid cell.

$$\varepsilon_q(\mathbf{b}, \hat{\mathbf{b}}) = \frac{1}{cg^2} \sum_{i=1}^{g^2} \mathbb{O}_i \left[\sum_{j=1}^c (\mathbf{p}_{ij} - \hat{\mathbf{p}}_{ij})^2 \right] \quad (4.6)$$

The constants $\lambda_c = 5$ and $\lambda_o = 0.5$ are used to stabilize the gradients during training and penalize more the localization error than the classification error. For both models,

the parameter initialization follows the pre-trained parameters based on the ImageNet 1000-class dataset (Russakovsky et al., 2015). We trained both models for 6,500 epochs with a starting learning rate of 10^{-3} . We use a weight decay of $5 \cdot 10^{-4}$ and momentum of $9 \cdot 10^{-1}$. We also use data augmentation during training with random crops, color shifting, angle rotations, and etc.

4.4 Final considerations

In this chapter we presented the proposed model. Firstly, in Section 4.1, we described the architecture of the model and its structure, discussing the choices made. In the Section 4.2, we present the dataset used in the work, highlighting the challenges found and the preprocessing employed. Lastly, in the Section 4.3, we detailed the training process and methodology used in the model.

Next, in Chapter 5, we will present the accuracy and inference time results of the proposed backbones in the distinct environments.

Results and discussion

In this chapter, we present the results of the development and tests of the work. In Section 5.1 we present the accuracy results of the models equipped with the backbones α and ω previously proposed. Next, in Section 5.2 we summarize the inference latency results. In Section 5.3 we compare the proposed YOLO method with alternative models using the SSD method. In the last section, Section 5.4, will describe the final considerations about the chapter.

5.1 Accuracy results

The average precision of the models equipped with Backbone α and Backbone ω are detailed on Table - 5.1. Backbone α 's model achieves a mean average precision of 93.20% for a threshold value $t = 0.50$. The Precision \times Recall Curves for each category are plotted in Figure - 5.1. With an increase in $t = 0.75$, the model with Backbone α achieves a 91.64% mAP and the Precision \times Recall Curves are plotted on Figure - 5.2. Next, the model with Backbone ω achieves a mean average precision of 93.20% for a threshold value $t = 0.50$. The Precision \times Recall Curves for each category are plotted in Figure - 5.3. With an increase in $t = 0.75$, the model with Backbone ω achieves a 91.64% mAP and the Precision \times Recall Curves are plotted on Figure - 5.4.

In the results, we can see that, for both models, the Minivan and Microbus categories are the least accurate. These vehicle categories also have a notable fall of precision in low values of recall; firstly falling before a recall reaches 0.2. This behavior does not happen in the other vehicle categories and suggests that the model struggles to generalize the

Table 5.1: Average Precision of the YOLO models equipped with α and ω

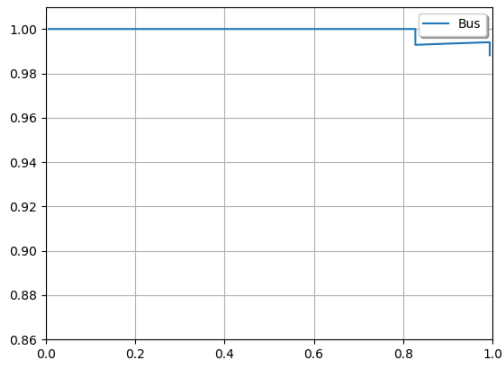
\times	t	Bus	Microbus	Minivan	SUV	Sedan	Truck	<i>Mean</i>
α	0.50	99.31%	90.08%	85.46%	92.21%	97.77%	94.36%	93.20%
	0.75	96.94%	90.08%	83.96%	91.81%	97.77%	89.29%	91.64%
ω	0.50	95.44%	81.19%	86.82%	93.64%	95.72%	94.64%	91.24%
	0.75	95.44%	80.65%	83.77%	93.61%	95.72%	91.39%	90.10%

parameters to learn distinctive features for the Minivan and Microbus categories. We conjecture this particular poor precision is the association between an inherent difficulty and the lack of samples for the Minivan and Microbus categories – represented in only 4.73% and 8.78% of the samples of the dataset, respectively – and can be improved with the supply of new samples.

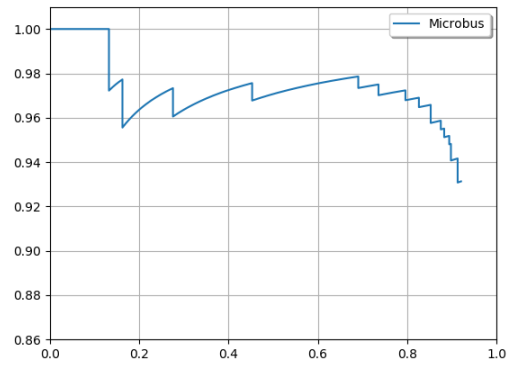
Otherwise, the Sedan and Bus vehicle categories have good precision. They all have high AP values ($>95\%$) and PR-Curves that remains high even when t increases. This nature suggests that the model can generalize well the parameters to learn distinctive features. The Sedan is the most common category in the dataset, counting with 58.89% of the samples. This abundance of samples can justify the particular good results. The Bus category does not have plenty of samples like the Sedan, but its features are very particular and we conjecture this helps the model’s learning.

In addition, the Truck and SUV vehicle categories have a regular performance, with good AP ($>90\%$) values but with quickly falling PR-Curves. This behavior indicates that the model does not excel when generalizing its parameters to learn the features from these categories. We conjecture that these categories share multiple features with others, like the Sedan and Bus, and can induce the model into confusion. Similar demeanour is noted in the works of Dong et al. (2015) and Roecker et al. (2018).

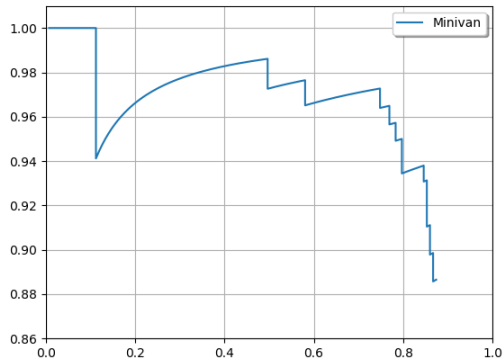
As expected, the model equipped with backbone α performs better than ω , but interesting characteristics come up when compared. Both models struggles to detect the Minivan and Microbus categories and have the best accuracy results for the Bus and Sedan categories. But, for the Truck and SUV categories, the shallow model with ω actually performs slightly better than the deep model α . We speculate that the deeper model cannot adjust its parameters to learn distinctly features like the shallow model, and the larger number of layers actually hurts the training by increasing the instability of the gradient.



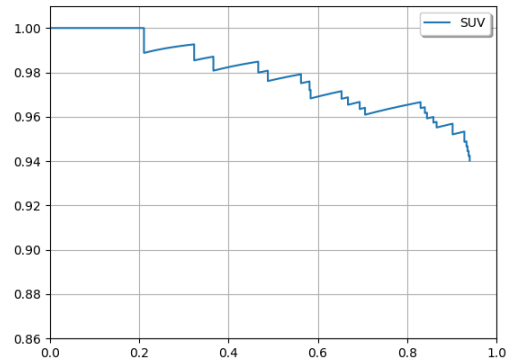
(a)



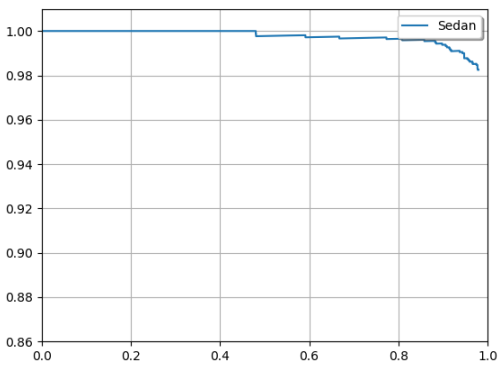
(b)



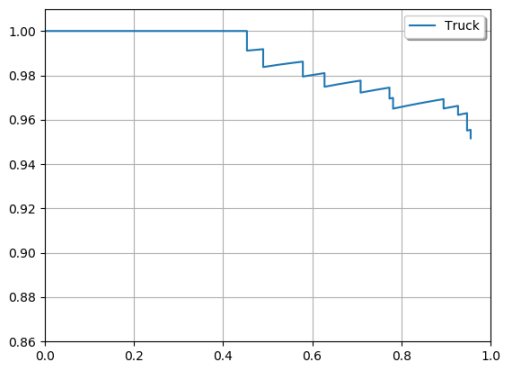
(c)



(d)

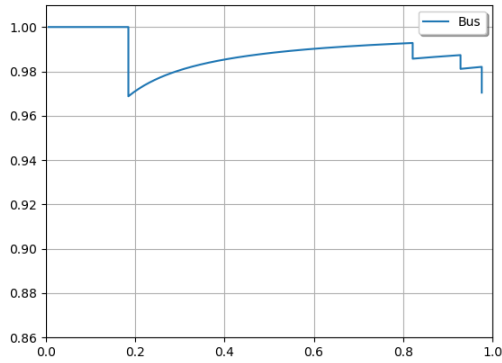


(e)

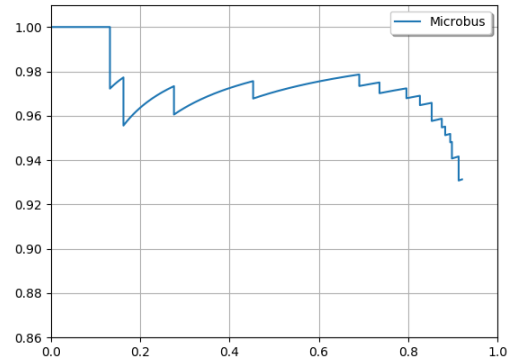


(f)

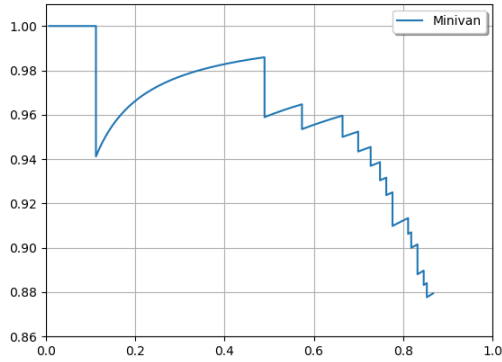
Figure 5.1: Precision×Recall Curves of the model equipped with the backbone α and with $t = 0.5$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f).



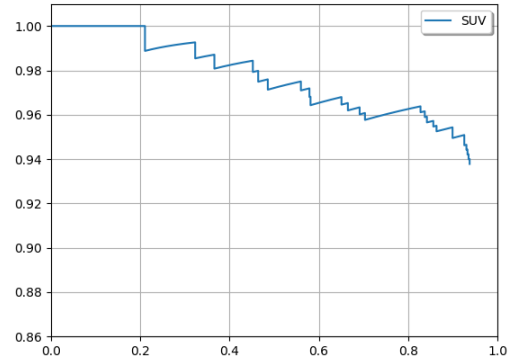
(a)



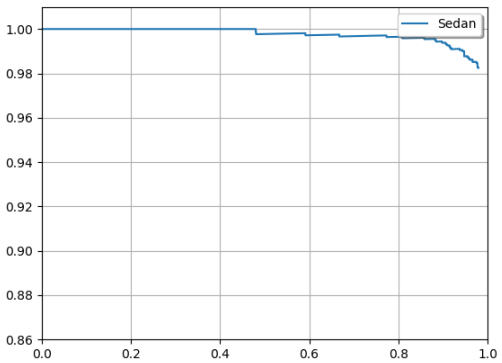
(b)



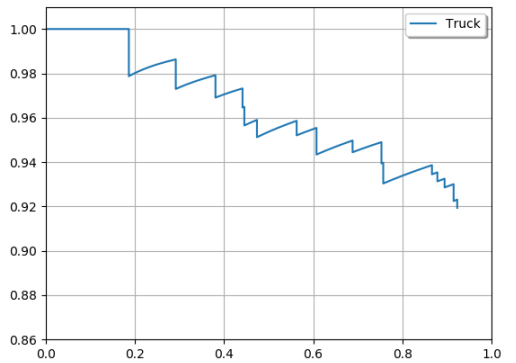
(c)



(d)

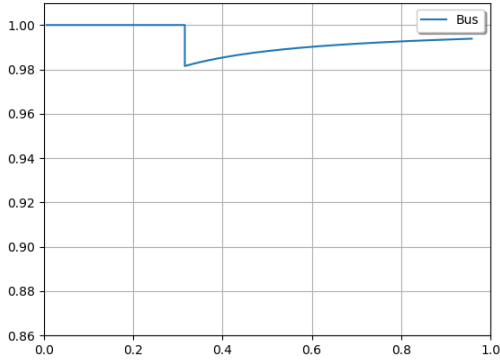


(e)

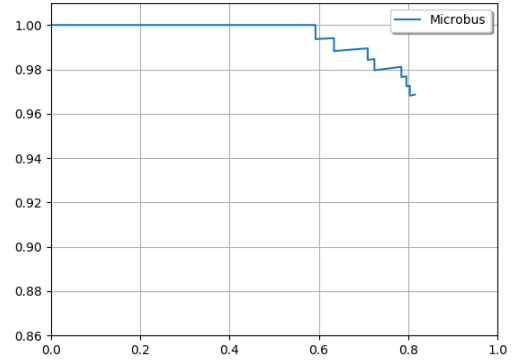


(f)

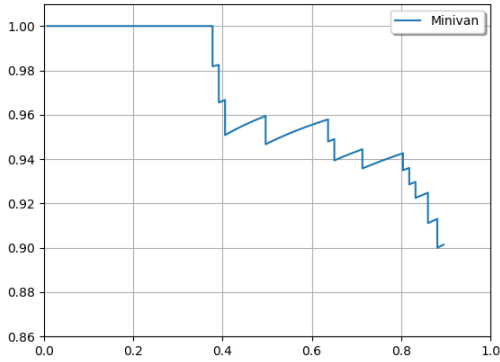
Figure 5.2: Precision×Recall Curves of the model equipped with the backbone α and with $t = 0.75$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f).



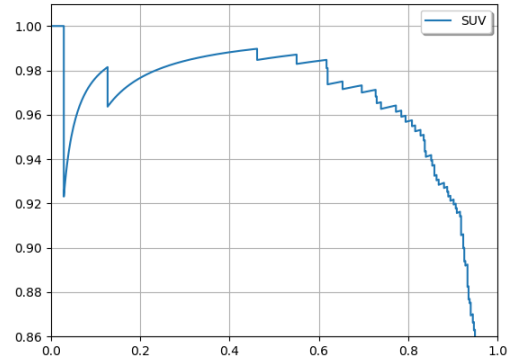
(a)



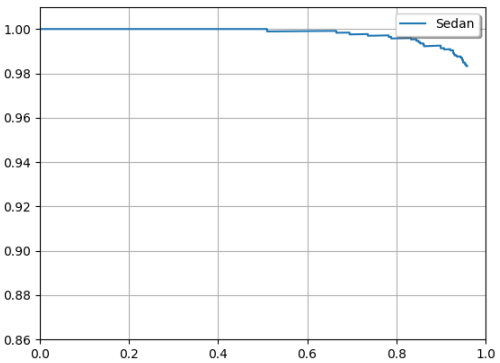
(b)



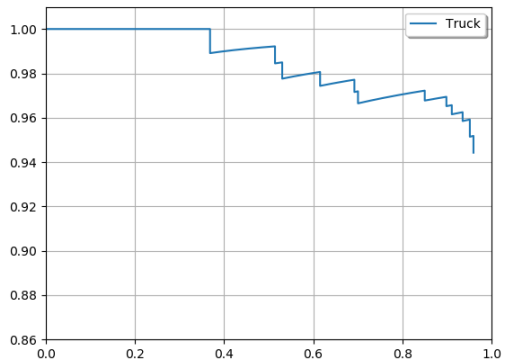
(c)



(d)

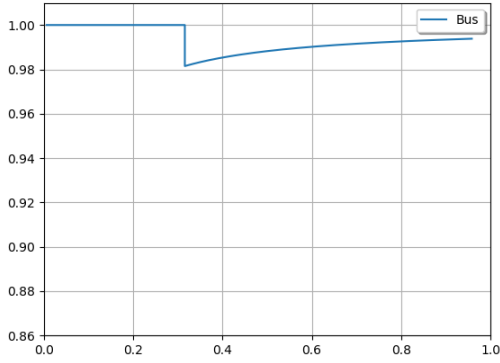


(e)

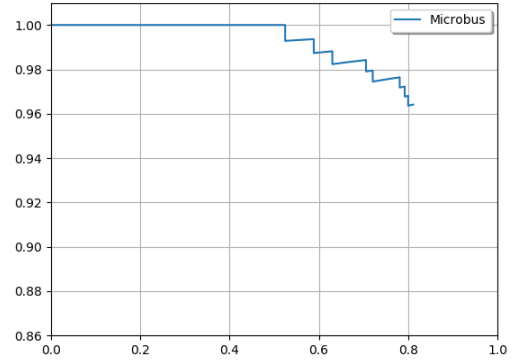


(f)

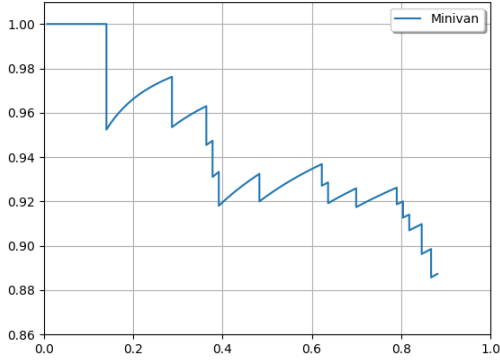
Figure 5.3: Precision×Recall Curves of the model equipped with the backbone ω and with $t = 0.5$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f).



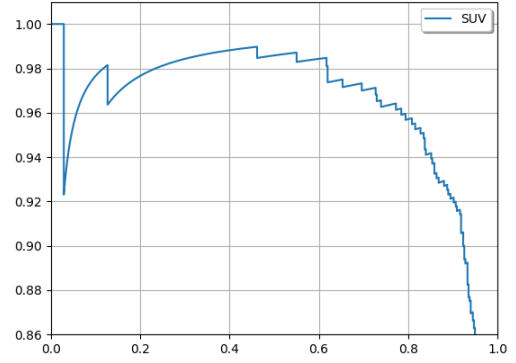
(a)



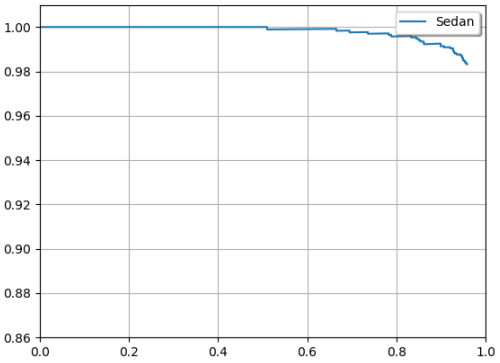
(b)



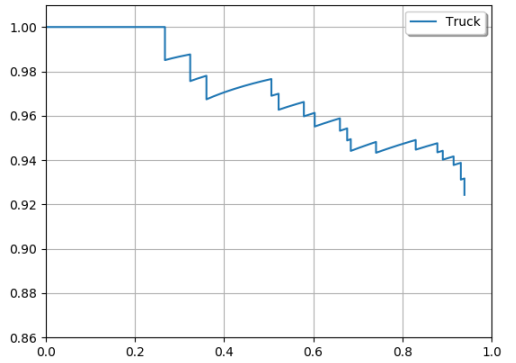
(c)



(d)



(e)



(f)

Figure 5.4: Precision×Recall Curves of the model equipped with the backbone ω and with $t = 0.75$ for each vehicle category: Bus (a), Microbus (b), Minivan (c), SUV (d), Sedan (e) and Truck (f).

5.2 Inference latency results

To evaluate the inference time, we performed the tests in three distinct environments to analyze the behavior of the models. The Test Environment I is a desktop computer equipped with an Intel CPU with 3.50GHz, 8 GB of RAM and an integrated GPU Intel HD Graphics 630. The Test Environment II is also a desktop computer equipped with the same CPU and RAM as Environment I but has a dedicated GPU NVIDIA GeForce GTX 1050-Ti with 4GB RAM and 768 CUDA cores. The Test Environment III is a single board computer ODROID-XU4 equipped with an ARM-based processor Exynos with 2 GHz, 2 GB of RAM and an integrated GPU Mali-T628. The features of each environment are summarized in Table - 5.2

Table 5.2: Test Environments for Inference Latency.

Environment	CPU	RAM	GPU
I	Core i5-7600 3.50GHz	8 GB	Intel HD Graphics 630
II	Core i5-7600 3.50GHz	8 GB	NVIDIA GeForce GTX 1050-Ti
III	ARM Exynos 2GHz	2 GB	Mali-T628 MP6

The tests were made with no other application except the core functionality of the operating system of the environment. Eight test rounds were performed for each model in each environment. The results of inference latency for the models equipped with backbones α and ω are presented in Table - 5.3.

Table 5.3: Inference latency of the YOLO models equipped with α and ω

Backbone	Environment	Average (ms)	Std. Deviation (ms)	Median (ms)
α	I	315.152	18.672	310.100
	II	69.426	5.866	69.198
	III	874.365	51.874	845.125
ω	I	72.846	6.874	71.478
	II	22.972	1.984	22.877
	III	121.365	23.458	114.189

As expected, the model equipped with ω performs significantly better than the model equipped with α regarding the inference latency. This difference increases the less computational resources are available to the model. The difference between both models on Environment III is far greater than the difference between both models in Environment

II. When comparing the accuracy with the inference latency, it can be seen that the models have a fair trade-off between the two metrics. There is no prohibitively drop of precision in the shallow model and the inference latency does not explodes in the deeper model.

Although the average precision drops significantly in the shallow model for some categories – like the Microbus and Minivan – the same metric stays high even with the threshold value is elevated for other categories – like the Bus and Car. This reinforce our assumption that major cause for the poor results of some categories is the lack of samples in the dataset.

In general terms, the YOLO method proved very efficient and notably adaptable for task of vehicle detection in digital images. These technical features of the method are desired and can be employed in a wide range of intelligent traffic systems; from embedded and reliable deployed in constrained and economical environments to dedicated and rigorous deployed in powerful and rich environments.

5.3 Comparison against alternative methods

To better compare the backbones proposed, we trained two more models with the same characteristics: an 512×512 input trained in the same dataset. We have opted to use the Single-Shot Detector (SSD) method Liu et al. (2016) in both models because it is in the same object detection method category as YOLO: methods that does not employ a region-proposal algorithm. Both models were equipped with the same backbones α and ω . The results for accuracy are summarized on Table - 5.4.

Table 5.4: Average Precision of the SSD models equipped with α and ω

\times	t	Bus	Microbus	Minivan	SUV	Sedan	Truck	Mean
α	0.50	99.55%	91.47%	86.074%	93.23%	97.87%	93.69%	93.65%
	0.75	95.37%	89.35%	83.69%	93.96%	97.97%	90.69%	91.84%
ω	0.50	97.86%	82.72%	87.50%	94.39%	94.57%	94.91%	91.99%
	0.75	96.65%	81.35%	84.84%	93.07%	96.50%	91.29%	90.62%

As can be seen, the SSD methods performs slightly better than the model YOLO regarding the accuracy. The same behaviors present in the YOLO method are present in the SSD methods regardless the backbone employed: the more accurate vehicle category is Sedan followed by Bus, and the least accurate are the Microbus and Minivan.

We also measured the inference time for the models using SSD using the same methodology described in Section 5.2. The results are summarized on Table - 5.5.

Table 5.5: Inference latency of the SSD models equipped with α and ω

Backbone	Environment	Average (ms)	Std. Deviation (ms)	Median (ms)
α	I	408.394	22.974	395.540
	II	77.662	8.426	74.150
	III	1028.619	74.223	1000.003
ω	I	86.236	8.440	84.741
	II	28.471	3.005	27.023
	III	152.563	33.741	150.100

As can be seen, the SSD models performs considerably worse than the YOLO models. The SSD models have a high number of floating-points operations than the YOLO methods and this assist the inference time to increase. The same behavior regarding the inference time of the YOLO and SSD methods are reported by Redmon and Farhadi (2018).

To compare the models in a cohesive measure, we defined the ratio between the mAP accuracy and the inference time latency as the **yielding** of the model. This metric offers a notion of the relation the “value” (accuracy) by the “cost” (inference time). Higher values are desired, as the accuracy is increased when compared to the inference time. Table - 5.6 and Table - 5.7 summarizes the yielding of each method using and both t defined for each environment for the models equipped with backbones α and ω .

Table 5.6: The yielding of SSD and YOLO models equipped with α

t	Method	I	II	III
0.50	YOLO	0.296	1.342	0.107
	SSD	0.244	1.282	0.097
0.75	YOLO	0.291	1.320	0.105
	SSD	0.234	1.228	0.093

Table 5.7: The yielding of SSD and YOLO models equipped with ω

Method	t	I	II	III
YOLO	0.50	0.290	1.314	0.104
	0.75	0.286	1.298	0.103
SSD	0.50	0.225	1.184	0.089
	0.75	0.222	1.167	0.088

As seen, the YOLO methods has a better yielding than the SSD methods for all the cases described in this work. The experimental results endorses our conjecture that the YOLO is descriptive and adaptable enough to be employed to detect vehicles present in digital images on the in a wide range of environments.

5.4 Final considerations

In this charppter, we presented the results of the development and tests of the work. In Section 5.1 we presented the accuracy results of the models equipped with the backbones α and ω previously proposed in Chapter 4. Next, in Section 5.2 we summarize the inference latency results. In Section 5.3 we performed a comparison between the proposed YOLO method and an alternative using the SDD method.

In Chapter 6, we will present the final considerations of the work, outlining its contributions and limitations. We also will describe future works and expectations about the impact of the work in the research community.

Final considerations and future works

This work presented a Master's Degree dissertation that proposes two models to detect, bound and classify vehicles presented in digital images into categories employing convolutional networks. It also presented theoretical foundations for the object detection research field and convolutional networks, some related works, the proposal of the models including the structure, the dataset and the training and test methodology; and lastly, the test results with some discussion.

As future works, we intend to improve the model and compare it with other well-know models using transfer learning, considering the results of accuracy and inference running time. The main limitations of the research concentrate in the lack of hardware resources for the implementation and evaluation of the model since deep convolutional networks need a powerful hardware resources such as GPUs or TPUs to perform training in a non-prohibitively time. Another limitation is the lack of standard datasets in the image-based problem of vehicle type classification which difficult the comparison between models of another works.

As demonstrated by the experimental results, we conjecture that a convolutional network can optimize its parameters to learn discriminative and reliable features for the vehicle type classification even images with challenging conditions like the samples presented in the dataset used to evaluate this work. In addition, the heavy usage of data-augmentation, the negative samples and other regularization techniques prevent the model to overfit. As a conjecture, we demonstrated that the accuracy of the models can be impacted by the quality of the dataset and a uniform distribution between the categories is desired to perform an fair comparison between them.

The object detection method YOLO is not the state-of-the-art regarding the accuracy, but proved to have a good trade-off with the inference latency and to be reasonably adaptable. Shallow models have lower average precision and shorter inference latency. In opposition, deeper models can have more accurate results accompanied by longer inference latency. This characteristic allows its usability to reinforce embedded intelligent traffic system, where the latency is usually a bottleneck and demand near real-time decisions; but also can be used to enhance expensive and precise analysis systems, where top accuracy is required to improve critical decisions.

6.1 Publications

The publications related to this work are chronologically listed bellow. The items marked with [★] are publications directly resulted of the work in the present document. Although some publications are not directly related to the present work, similar theoretical and analysis concepts were employed that contributed to the development of this work.

- [★] ROECKER, MAX N.; COSTA, YANDRE M.G.; ALMEIDA, JOAO L.R.; MATSUSHITA, GUSTAVO H. G.. **Automatic Vehicle type Classification with Convolutional Neural Networks**. In: 2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP), 2018, Maribor. 2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP), 2018. p. 1-5.
- [★] ROECKER, MAX N.; COSTA, YANDRE M. G.; BRITTO, ALCEU S.; OLIVEIRA, LUIZ E. S.; BERTOLINI, DIEGO. **Vehicle Detection and Classification in Traffic Images Using ConvNets With Constrained Resources**. In: 2019 International Conference on Systems, Signals and Image Processing (IWSSIP), 2019, Osijek. 2019 International Conference on Systems, Signals and Image Processing (IWSSIP), 2019. p. 83-88.
- ALMEIDA, JOÃO; FLORES, FRANKLIN; ROECKER, MAX; BRAGA, MARCO; COSTA, YANDRE. **An Indoor Sign Dataset (ISD): An Overview and Baseline Evaluation**. In: 14th International Conference on Computer Vision Theory and Applications, 2019, Prague. Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2019. v. 4. p. 505-512.

References

CIREŞAN, D. C.; UELI, M.; MASCI, J.; GAMBARDILLA, L. M.; SCHMIDHUBER, J. Flexible, high performance convolutional neural networks for image classification. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, Barcelona, Catalonia, Spain: AAAI Press, 2011, p. 1237–1242.

ŞENTAŞ, A.; İSABEK TASHIEV; KÜÇÜKAYVAZ, F.; KUL, S.; EKEN, S.; SAYAR, A.; BECERIKLI, Y. Performance evaluation of support vector machine and convolutional neural network algorithms in real-time vehicle type and color classification. *Evolutionary Intelligence*, 2018.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, v. 2, n. 4, p. 303–314, 1989.

Disponível em <https://doi.org/10.1007/BF02551274>

DONG, Z.; WU, Y.; PEI, M.; JIA, Y. Vehicle type classification using a semisupervised convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, v. 16, n. 4, p. 2247–2256, 2015.

EVERINGHAM, M.; ESLAMI, S. M. A.; GOOL, L. V.; WILLIAMS, C. K. I.; WINN, J.; ZISSERMAN, A. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, v. 111, n. 1, p. 98–136, 2015.

FELZENSZWALB, P. F.; GIRSHICK, R. B.; MCALLESTER, D.; RAMANAN, D. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 32, n. 9, p. 1627–1645, 2010.

FELZENSZWALB, P. F.; HUTTENLOCHER, D. P. Efficient graph-based image segmentation. *International Journal of Computer Vision*, v. 59, n. 2, p. 167–181, 2004.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, v. 36, n. 4, p. 193–202, 1980.

- GIRSHICK, R. Fast R-CNN. In: *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, p. 1440–1448.
- GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, p. 580–587.
- GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011*, v. 15, p. 315–323, 2011.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. MIT Press, <http://www.deeplearningbook.org>, 2016.
- GUPTE, S.; MASOUD, O.; MARTIN, R. F. K.; PAPANIKOLOPOULOS, N. P. Detection and classification of vehicles. *IEEE Transactions on Intelligent Transportation Systems*, v. 3, n. 1, p. 37–47, 2002.
- HAHNLOSER, R.; SARPESHKAR, R.; MAHOWALD, M. A.; DOUGLAS, R.; SEUNG, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, v. 405, p. 947–951, 2000.
- HAN, S.; HAN, Y.; HAHN, H. Vehicle detection method using haar-like feature on real time system. *World Academy of Science, Engineering and Technology*, v. 59, p. 455–459, 2009.
- HE, K.; GKIOXARI, G.; DOLLAR, P.; GIRSHICK, R. Mask R-CNN. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, p. 2980–2988.
- HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, p. 770–778.
- HUANG, J.; RATHOD, V.; SUN, C.; ZHU, M.; KORATTIKARA, A.; FATHI, A.; FISCHER, I.; WOJNA, Z.; SONG, Y.; GUADARRAMA, S.; MURPHY, K. Speed/accuracy trade-offs for modern convolutional object detectors. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, p. 3296–3297.
- JI, P.; JIN, L.; LI, X. Vision-based vehicle type classification using partial gabor filter bank. In: *IEEE International Conference on Automation and Logistics*, 2007, p. 1037–1040.

- JIANG, M.; LI, H. *Vehicle classification based on hierarchical support vector machine* Springer International Publishing, p. 593–600, 2014.
- KIM, P. K.; LIM, K. T. Vehicle type classification using bagging and convolutional neural network on multi view surveillance image. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Honolulu, HI, USA, 2017, p. 914–919.
- KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGESS, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q., eds. *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., p. 1097–1105, 2012.
- LECUN, Y. *Generalization and network design strategies* Zürich, Switzerland: Elsevier, 1989.
- LIN, M.; CHEN, Q.; YAN, S. Network in network. In: *Proceedings of the International Conference on Learning Representations 2014*, Banff, arXiv:1312.4400, 2014a.
- LIN, T.-Y.; DOLLÁR, P.; GIRSHICK, R.; HE, K.; HARIHARAN, B.; BELONGIE, S. Feature pyramid networks for object detection. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, p. 936–944.
- LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft COCO: Common Objects in Context. In: *European Conference on Computer Vision*, Zürich: Springer International Publishing, 2014b, p. 740–755.
- LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y.; BERG, A. C. SSD: Single Shot MultiBox Detector. In: *Computer Vision – ECCV 2016*, Cham: Springer International Publishing, 2016, p. 21–37.
- MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.
- REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, p. 779–788.

REDMON, J.; FARHADI, A. YOLO9000: Better, Faster, Stronger. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, p. 6517–6525.

REDMON, J.; FARHADI, A. *YOLOv3: An Incremental Improvement*. Technical report, University of Washington, Seattle, USA, 2018.

Disponível em <https://pjreddie.com/darknet/yolo/>

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, n. 6, p. 1137–1149, 2017.

ROECKER, M. N.; COSTA, Y. M. G.; ALMEIDA, J. L. R.; MATSUSHITA, G. H. G. Automatic vehicle type classification with convolutional neural networks. In: *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2018, p. 1–5.

ROECKER, M. N.; COSTA, Y. M. G.; BRITTO, A. S.; OLIVEIRA, L. E. S.; BERTOLINI, D. Vehicle detection and classification in traffic images using convnets with constrained resources. In: *2019 26th International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2019, p. 83–88.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015.

SELBES, B.; SERT, M. Multimodal vehicle type classification using convolutional neural network and statistical representations of MFCC. In: *14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, Lecce, Italy, 2017, p. 1–6.

SERRE, T.; WOLF, L.; BILESCHI, S.; RIESENHUBER, M.; POGGIO, T. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 29, n. 3, p. 411–426, 2007.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

SPRINGENBERG, J. T.; DOSOVITSKIY, A.; BROX, T.; RIEDMILLER, M. Striving for simplicity: The all convolutional net. In: *Workshop proceedings of the International Conference on Learning Representations 2015*, 2015.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCHE, V.; RABINOVICH, A. Going deeper with convolutions. In: *Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015.

SZELISKI, R. *Computer vision: Algorithms and applications*. 1 d. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, p. I-I.

WANG, X.; ZHANG, W.; WU, X.; XIAO, L.; QIAN, Y.; FANG, Z. Real-time vehicle type classification with deep convolutional neural networks. *Journal of Real-Time Image Processing*, 2017.

WU, H. Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions. *Information Sciences*, v. 179, n. 19, p. 3432–3441, 2009.

ZHANG, Z.; TAN, T.; HUANG, K.; WANG, Y. Three-dimensional deformable-model-based localization and recognition of road vehicles. *IEEE Transactions on Image Processing*, v. 21, n. 1, p. 1–13, 2012.

ZHOU, Y. T.; CHELLAPPA, R. Computation of optical flow using a neural network. In: *IEEE 1988 International Conference on Neural Networks*, 1988, p. 71–78 vol.2.