OBJECT DETECTION FROM CAPTIVE BALLOON IMAGERY USING DEEP LEARNING

Victória Maria Gomes Velame

Master's Dissertation of the Graduate Course in Remote Sensing, guided by Drs. José Claudio Mura, and Leonardo Sant'Anna Bins, approved in April 02, 2020.

INPE
São José dos Campos
2020

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21c/2020/03.30.16.14-TDI

OBJECT DETECTION FROM CAPTIVE BALLOON IMAGERY USING DEEP LEARNING

Victória Maria Gomes Velame

Master's Dissertation of the Graduate Course in Remote Sensing, guided by Drs. José Claudio Mura, and Leonardo Sant'Anna Bins, approved in April 02, 2020.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/428J8UE>

INPE
São José dos Campos
2020

Cataloging in Publication Data

Velame, Victória Maria Gomes.

Ve54o Object detection from captive balloon imagery using deep learning / Victória Maria Gomes Velame. – São José dos Campos : INPE, 2020.

xxvi + 101 p. ; (sid.inpe.br/mtc-m21c/2020/03.30.16.14-TDI)

Dissertation (Master in Remote Sensing) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2020.

Guiding : Drs. José Claudio Mura, and Leonardo Sant'Anna Bins.

1. Object Detection. 2. Deep Learning. 3. Convolutional Neural Network. 4. Remote Sensing. 5. Captive Balloons. I.Title.

CDU 528.8:004.8

Aluno (a): ***Victória Maria Gomes Velame***

Título: "OBJECT DETECTION FROM CAPTIVE BALLOON IMAGERY USING DEEP LEARNING"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de ***Mestre*** **em** ***Sensoriamento Remoto***

Dr.	**Thales Sehn Körting**	[illegible]
		Presidente / INPE / São José dos Campos - SP
		(X) Participação por Video - Conferência
		(X) Aprovado ( ) Reprovado
Dr.	**José Claudio Mura**	[illegible]
		Orientador(a) / INPE / São José dos Campos - SP
		(X) Participação por Video - Conferência
		(X) Aprovado ( ) Reprovado
Dr.	**Leonardo Sant'Anna Bins**	[illegible]
		Orientador(a) / INPE / São José dos Campos - SP
		(X) Participação por Video - Conferência
		(x) Aprovado ( ) Reprovado
Dr.	**Ney Rafael Sêcco**	[illegible]
		Convidado(a) / ITA / São José dos Campos - SP
		(X) Participação por Video - Conferência
		(X) Aprovado ( ) Reprovado

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 02 de abril de 2020

"Intellectual growth should commence at birth and cease only at death".

Albert Einstein

Dedicated to God, my parents and my husband.

ACKNOWLEDGEMENTS

First of all, I thank God, who gave me the necessary conditions to develop this work.

I thank my advisors, Dr. José Claudio Mura and Dr. Leonardo Sant'Anna Bins, for the patience, guidance and teachings.

I thank my family for their endless love and support. João, my husband, for his love, friendship, understanding, encouragement and unconditional support. He always kept me motivated to face new challenges and helped me through the difficult moments. My parents and brother, for their incredible faith, backing and inspiration.

I would also like to thank my friend Ronaldo for the encouragement and friendship.

I thank ALTAVE for the images given and all the support. Notably, the Production Management team for all their friendship, motivation and support. Leonardo and Guilherme for cheering me up when needed, Michelle and Arthur for teaching me valuable lessons.

I thank Madalena, my English teacher, for all help and friendship.

I also thank everyone that contributed to this work, either directly or indirectly.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. I thank CAPES for the financial support.

ABSTRACT

The combination of remote sensing and computer vision technologies have been used to monitor large areas. In order to ensure their local security. This monitoring requires high temporal and spatial resolution sensors. Captive balloons with infrared and visible sensors, like Altave system, can perform a long-term day-night surveillance with viable cost in comparison with other aerial vehicles. Altave captive balloon system provides security of large areas by continuously monitoring people and vehicles, which is exhaustive for humans due to the large amount of data. To provide a more efficient and less arduous monitoring, this work developed a technology based on DL (Deep Learning), more specifically Faster R-CNN (Region-based Convolutional Neural Network - R-CNN), capable of detecting people and vehicles in images from captive balloon's infrared and visible sensors. The advantage of CNN object detectors is their ability to generalize, which make them more efficient to deal with some captive balloon image features, such as objects on different points of view, positions and scales. This work used videos provided by Altave Company (from their captive balloon system) to manually build two databases containing about 700 images each, one for the infrared and the other for the visible data. Since training a large CNN from scratch requires a large database and high computational power, two networks were fine-tuned from a Faster R-CNN, pre-trained on RGB (red, green, blue) images. The accuracy, mAP and AR metrics reached on the test datasets indicates the network high performance. The accuracy was 87.1% for the infrared network and 86.1% for the visible. These high accuracies demonstrated that a Faster R-CNN pre-trained only in ordinary RGB images can be fine-tuned to work satisfactorily on 3-band RGB visible remote sensing images and even on 1-band infrared images, as long as they are properly converted for 3-band images by repeating the infrared band on the three channels. The networks satisfactorily detected people and vehicle on images from Altave captive balloon system. They could detect multiple objects in an image with a variety of angles, positions, types (for vehicles), scales, and even with some noise and overlap. They also presented some mistaken detections caused by splitting parts of one object into two objects or merging two objects from the same class in one large object. These types of mistakes are not a relevant problem for surveillance because it is much more important to detect the objects than to locate or count them.

Keywords: Object Detection. Deep Learning. Convolutional Neural Network. Remote Sensing. Captive Balloons.

DETECÇÃO DE OBJETOS EM IMAGENS DE BALÃO CATIVO UTILIZANDO DEEP LEARNING

RESUMO

A combinação de tecnologias de sensoriamento remoto com visão computacional tem sido utilizada para monitorar grandes áreas, de modo a garantir a segurança local. Esse monitoramento requer sensores de alta resolução temporal e espacial. Os balões cativos com sensores visível e infravermelhos, como os da Altave, são capazes de realizar vigilância diurna e noturna a longo prazo, com custo viável comparado com outros veículos aéreos. O sistema de balões cativos da Altave fornece segurança para grandes áreas por meio do monitorando contínuo de pessoas e veículos, função que é exaustiva para seres humanos devido à grande quantidade de dados. Com o objetivo de proporcionar um monitoramento mais eficiente e menos árduo, neste trabalho foi desenvolvido uma tecnologia baseada em Aprendizado Profundo, mais especificamente Faster R-CNN (Region-based Convolutional Neural Network - R-CNN), capaz de detectar pessoas e veículos em imagens de sensores infravermelho e visível de balões cativos. A vantagem dos detectores de objetos baseados em CNN é sua capacidade de generalização, tornando-os mais eficientes para algumas características de imagem de balões cativos, como objetos em diferentes visadas, posições e escalas. Este trabalho utilizou os vídeos fornecidos pela empresa Altave (do sistema de balão cativo) para criar, manualmente, dois bancos de dados com cerca de 700 imagens, um para o infravermelho e a outro para o visível. Como o treinamento de uma CNN de grande complexidade desde o início requer um banco de dados grande e alto poder computacional, duas redes foram ajustadas a partir de uma rede Faster R-CNN pré-treinada em imagens RGB (vermelha, verde, azul). A acurácia, métricas mAP e AR alcançadas nos conjuntos de dados de teste comprovam o alto desempenho das redes treinadas. A acurácia do sistema foi de 87,1% para a rede infravermelha e de 86,1% para a óptica. Essas altas acurácias demonstraram que uma Faster R-CNN pré-treinada apenas em imagens RGB comuns, pode ser ajustada para funcionar satisfatoriamente em imagens de sensoriamento remoto visível RGB de 3-bandas e até mesmo em imagens infravermelhas de 1-banda, desde que sejam adequadamente convertidas para imagens 3-bandas através da repetição desta banda nos três canais. As redes construídas foram capazes de detectar satisfatoriamente pessoas e veículos em imagens do sistema de balões cativos da Altave, sendo capaz de detectar múltiplos objetos em vários ângulos, posições, tipos (no caso de veículos), escalas e até mesmo com algum ruído e sobreposição. Eles também apresentaram algumas detecções erradas causadas pela divisão de partes de um objeto em dois objetos ou pela fusão de dois objetos da mesma classe em um objeto maior. Esse tipo de erro não é relevante para o monitoramento com vigilância devido ao fato de ser mais importante detectar objetos do que localizá-los ou contá-los.

Palavras-chave: Detecção de Objetos. Aprendizado Profundo. Rede Neural Convolucional. Sensoriamento Remoto. Balões Cativos.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	–	Artificial Neural Network
AP	–	Average Precision
AR	–	Average Recall
CNN	–	Convolutional Neural Network
CS	–	Confidence Score
DL	–	Deep Learning
FN	–	False Negative
FOV	–	Field of View
FP	–	False Positive
FVs	–	Fisher Vectors
GPU	–	Graphic Processing Unit
ILSVRC	–	ImageNet Large Scale Visual Recognition Challenge
IoU	–	Intersection over Union
LTA	–	Lighter Than Air
mAP	–	Mean Average Precision
ML	–	Machine Learning
NMS	–	Non-Maximum Suppression
PNG	–	Portable Network Graphics
R-CNN	–	Region-based CNN
ReLU	–	Rectified Linear Unit
RGB	–	Red, green, blue
RoI	–	Region of Interest
RPN	–	Region Proposal Network
TN	–	True Negative
TP	–	True Positive
SIFT	–	Scale Invariant Feature Transform
SVM	–	Support Vector Machine
UAV	–	Unmanned Aerial Vehicle
XML	–	Extensible Markup Language

LIST OF SYMBOLS

α	–	Learning rate
$\mathcal{L}$	–	Loss function
$\mathcal{L}_{s_n}$	–	Loss function of a training sample
θ_i	–	Network current parameters of the i-th iteration
$\sigma(x)$	–	Sigmoid function
A^T	–	Transpose of matrix A
b	–	Neuron bias
$b^{(l)}$	–	Bias vector of the l-th network layer
$b_j^{(l)}$	–	Bias value for the j-th neuron of layer l
c_i	–	Network output desired probability for i-th class
$\bar{c}_i$	–	Network output prediction for i-th class
C	–	Total number of classes
$cosh(x)$	–	Hyperbolic cosine
e	–	Euler constant
$f^{(l)}$	–	Activation functions of layer l
$f'^{(l)}$	–	Activation function derivative of layer l
$h^{(l)}$	–	Output vector of layer l
$h_j^{(l)}$	–	Output of j-th neuron of layer l
$H(p,q)$	–	Cross-entropy function for a prediction q and expectation p
K_h	–	Height CNN filter size
K_w	–	Width CNN filter size
$\ln(x)$	–	Natural logarithm of x
$\log(x)$	–	Base 10 logarithm of x
L	–	Last layer of a network
$max(a,b)$	–	Function that returns the maximum value between a and b
p_i	–	Predicted probabilities
p_i^*	–	Ground-truth label
P_h	–	Height CNN padding
P_w	–	Width CNN padding
$ReLU(x)$	–	Rectified Linear Unit activation function
s_n	–	Training sample
$sinh(x)$	–	Hyperbolic sine
S_h	–	Height CNN stride
S_w	–	Width CNN stride
$tanh(x)$	–	Hyperbolic tangent
t_i	–	Coordinates vector of a predicted bounding box
t_i^*	–	Coordinates vector of a ground-truth box
T	–	Threshold
T_{CS}	–	Confidence score threshold
T_{IoU}	–	Intersection over Union threshold

$\vec{w}$	–	Neuron weights vector
$w_{jk}^{(l)}$	–	Weight between the j-th neuron of layer l and the k-th neuron of layer $l-1$
$W^{(l)}$	–	Weight matrix of layer l
$\vec{x}$	–	Neuron input vector
X	–	Network input vector
y	–	Neuron output
$z_j^{(l)}$	–	Network input on the j-th neuron of layer l
(l)	–	Index of the l-th network layer from left to right
∂	–	Partial derivative operator
∇	–	Gradient operator
$\nabla_\theta \mathcal{L}$	–	Loss function gradient with respect to θ
$\mathcal{L}_{cls}$	–	Classification loss
$\mathcal{L}_{reg}$	–	Regression loss
$\tilde{\mathcal{L}}$	–	Loss function with regularization

CONTENTS

Page

1 INTRODUCTION

Remote sensing technologies and techniques combined with computer vision and image processing algorithms have been used to monitor large areas. In oceans, for example, these tools have been applied to monitor, through periodic surveillance, vessels (PAVLAKIS et al., 2001) and oil spills (JHA et al., 2008). To prevent and mitigate environmental damages.

Orbital sensor images have been used to monitor large areas, such as deforestation (GUILD et al., 2004), land use and land cover (LU et al., 2005), and spread of diseases in agricultural crops (BENDINI et al., 2014). Applications such as crop stress detection require high temporal and spatial resolution images. To obtain this type of data, a possible solution is the use of remote sensors in Unmanned Aerial Vehicles (UAVs).

The monitoring of large areas like courtyards, industries, major events, and pipelines for security purposes requires high temporal and spatial resolution images, since constant monitoring is required with a great detail of the objects, allowing only short time between new observations from the same location (HAUSAMANN et al., 2005). The spatial resolution must be high enough to allow detection of objects such as people and vehicles. To obtain these resolutions, in general, the monitoring of these areas is performed by imaging with remote sensors arranged in UAVs (HAUSAMANN et al., 2005). Such as Lighter Than Air (LTA) platforms, drones, and helicopters.

An aerostat is a LTA aircraft that gains its lift through the use of a buoyant gas. The main types of aerostat are airships, balloons, tethered balloons and hybrid airships (SANTOS, 2018; AZEVEDO, 2016). Tethered aerostats (also knows as captives aerostats), in comparison to other flying solutions have the advantage of lower operational costs and the ability to stay aloft for larger periods without interruption (AZEVEDO, 2016). The Brazilian company Altave develops unmanned captive balloons capable of reaching up to 200 meters in height, resulting in a privileged viewpoint for monitoring, as shown in Figure 1.1. Nevertheless, there are some peculiarities in the image features, mainly due to the angle on the line of sight and displacements in all axes (AZEVEDO et al., 2013), specially in roll axis.

Figure 1.1 - Altave Captive balloon monitoring a forest.

SOURCE: Altave (2019).

Monitoring people and vehicles in large areas is extremely important to ensure local security, but the analysis of the large amount of data generated in the process is exhaustive for a human being. Therefore, it is important to develop algorithms capable of assisting humans in this task. To make it less arduous and more efficient. Captive balloons are well suited for this work because of their ability of long-term operation (several days) carrying high temporal and spacial resolution sensors at low cost. These features make this aircraft an attractive solution to monitor people and vehicles in a large area.

The sensors commonly used for people and vehicles monitoring are cameras with infrared and visible sensors (HAUSAMANN et al., 2005), since they allow detecting and monitoring the flow of people and vehicles in the presence or absence of sunlight. The reflective and emissive properties of materials vary greatly along the spectrum, so features detected by the infrared and visible sensor image may be different and show the same object in distinct perspectives.

While humans easily detect many objects on real life images, this is still a difficult task for a machine. The same object can be in many distinct orientations, viewpoints or even partially occluded, causing further variations in the resulting image (BIEDERMAN, 1987).

In computer vision, change and object detections are the most used techniques to perform automatic monitoring. Change detection requires, at least, two chronologically successive, geometrically registered images to be able to calculate the differences. Therefore, change detection is only suited in monitoring with static cameras or already registered images. One advantage of change detection is that the algorithms used do not require training, for most applications. On the other hand, object detection techniques don't need successive images, therefore, the geometric registry isn't needed either. But they need to be properly trained to recognize the sampled objects.

For non registered images with constant movement due to the natural movement of the sensors, as those generated by imaging sensors in captive balloons, object detection is better suited. The state-of-the-art object detection techniques are based on Deep Learning (DL), more specifically Convolution Neural Network (CNN), as pointed by Krizhevsky et al. (2012), because CNN has better generalization capabilities than manually-engineered object detectors. Making them much more selective about the detected changes on the monitored scene.

1.1 Motivation

Large areas such as courtyards, industries, and major events need a technology to ensure their security. A surveillance performed by continuous aerial monitoring can be a feasible solution. Captive balloons with infrared and visible sensors can perform long-term day-night surveillance with viable cost in comparison with other aerial vehicles. An Altave balloon was successfully used to monitor people and vehicles during the 2016 Olympic Games in Rio de Janeiro (AZEVEDO, 2016).

Continuous monitoring is an arduous task for humans, which motivated the development of technologies to facilitate this task. Although there already are several monitoring algorithms, there is none specifically tailored to the particularities of images sensed by visible and infrared sensors arranged on captive balloons. This problem motivated the development of a CNN able to detect people and vehicles (objects of primary interest in this type of surveillance) in this type of image for the purpose of assisting humans during monitoring tasks.

1.2 Objective

This work aims to develop a system capable of detecting people and vehicles in images from visible and infrared sensors arranged in captive balloons. This system should be able to help humans on the continuous monitoring of large areas.

The object detection system developed in this work must cope with the following situations:

- object viewpoint changes;
- radiometric changes; and
- the presence of some fog or dust.

This work is limited to make a proof of concept on the capability of CNN object detectors to deal with images from visible and infrared sensors arranged in captive balloons. Although the captive balloons records data in video format, only images were used in this work. Thus, the frames (images) were captured from the videos.

1.3 Contributions

The main contributions of this work are listed below:

- outline a technique to perform object detection, more specifically people and vehicles, in captive balloons imagery;
- the usage of state-of-the-art object detectors based on CNN methods on remote sensing images;
- verify the viability of using an object detector based on CNN pre-trained for a 3-bands ordinary image on a red, green and blue (RGB) visible and 1-band infrared remote sensing image; and
- verify the viability of using the CNN fine tuning algorithm on a remote sensing image, even with spectrum band changes.

1.4 Organization

This is organized as follows:

- **Chapter 2 - Theoretical Background** introduces the concepts of artificial neural network, DL, CNN and the network used to perform the object detection, including the process of training and some regularization metrics;

- **Chapter 3 - Materials and Methods** describes the sensors, computer and API used in this work, the process of building the database, the evaluation metrics, and, finally, the methodology and parameters used to train the network;
- **Chapter 4 - Results and Discussions** presents the results achieved for the infrared and visible data with some examples, and their discussions; and
- **Chapter 5 - Conclusions** presents the conclusions and suggestions for future work.

2 THEORETICAL BACKGROUND

This chapter presents an overview of the fundamental principles relevant to this master dissertation. As mentioned in Chapter 1, this work is motivated by the usage of a CNN to perform people and vehicle detection in remote sensing images. Therefore, this chapter will progress from basic networks to complex neural structure capable to perform object detection. Furthermore, it will present the basic principles involved in the training of CNNs and some techniques to improve it.

2.1 Artificial Neural Network

An Artificial Neural Network (ANN) is a branch in the field of artificial intelligence designed to learn statistical models for pattern recognition tasks (VELTE, 2015). It is a computer model structure capable to learn and generalize the input data pattern. It can solve real non-linear tasks which are difficult to model using algorithms based on static rules (VELTE, 2015). In short, ANNs learn to perform tasks by considering examples, without being programmed with any specific rule. They were inspired by biological neural networks that gain knowledge through experience, like the human brain network which has about 86 billion neurons (AZEVEDO et al., 2009).

The patterns recognized by an ANN are numerical, therefore real world data, such as images, sounds, texts and time series, must be converted into numbers or numerical vectors. In classification, an ANN predicts a class based on the pattern learned by the examples in its training stage. In recent years neural networks have achieved good results in many image analysis areas, being able to learn features which would be hard to design by hand (MALMROS, 2018).

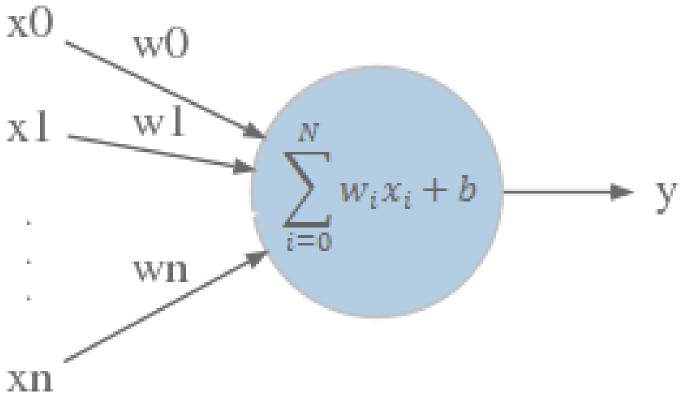
In remote sensing, it's not different, ANNs have obtained good results in many tasks such as in the prediction of wood volume and tree vegetation biomass (MIGUEL et al., 2015) and in the classification of pasture degradation levels (CHAGAS et al., 2009). Despite the good results achieved, there still are some difficulties with training time and the large number of samples required.

2.1.1 Network propagation

The ANN consists of a large number of simple computational elements denominated (artificial) neurons or nodes, which are densely interconnected and operating in parallel (SECCO; MATTOS, 2017). A neuron is a single element of an ANN. That converts a group of n inputs into one output. These inputs can derive from external variables or from other neurons. In a neuron, each input (x_i) is multiplied by a corresponding weight (w_i), and then summed altogether with a threshold value, called bias (b). The result of this sum is then applied into a function, named activation function, which determines the neuron output. Figure 2.1 and Equation 2.1 show the output (y) of a single neuron when its parameters: inputs, weights, bias and activation function, respectively, are $\vec{x} = (x_0, x_1, ..., x_N)^T$, $\vec{w} = (w_0, w_1, ..., w_N)$, b and $f(x)$. The bias value allows the activation function to be shifted left or right to obtain a better fitting.

$$y = f(\vec{w} \cdot \vec{x} + b) = f(\sum_{i=0}^{N} \omega_i x_i + b) \tag{2.1}$$

Figure 2.1 - Diagram of a single neuron.

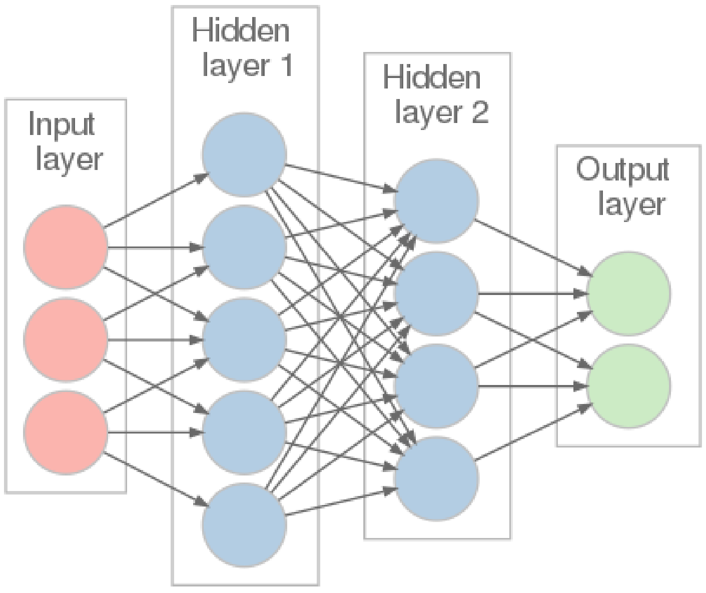


SOURCE: Author's production.

A single neuron can make an ANN, but it will be a very poor one. Many neurons are connected to scale up and make more complex networks, enabling it to learn more complex patterns. There are several ANN architectures, the multi-layer feed-forward ANN is the most commonly used for object detection applications. In this architecture, the nodes are connected in structures called *layers* to expand into large ANNs. A layer consists of a set of several neurons, each one receiving inputs from the previous layer by weighted connections.

Figure 2.2 shows an example of a small generic multi-layer feed-forward ANN with two hidden layers. The first layer is the input layer, which receives the input data as numeric attributes; then there are some intermediary layers called hidden layers, where the main propagation processing happens to extract feature from the inputs; and the output layer, which gives the network output, such as classes predictions. The number of hidden layers determine the network depth.

Figure 2.2 - Example of a small multi-layer feed-forward ANN with two hidden layers.



SOURCE: Author's production.

2.2 Network architecture

An ANN is specified by its topology, also called architecture, which is the structure composed by its neurons and their connections. The word architecture refers to the overall structure of the network, how many layers it has, how many neurons each layer has, and how these units are connected to each other. Most ANN arrange these layers in a chain structure, called chain-based architectures (GOODFELLOW et al., 2017). In these networks the main architectural considerations are choosing the depth (number of layers) of the network and the width (number of neuron) of each layer. Expanding the network depth or width increases the generalization capabilities of the model.

In a chain-based architecture, the output of a layer is a function of the output of the layer that preceded it. The first layer is given by Equation 2.2, the second layer by Equation 2.3 and so on by Equation 2.4. In these equations, X is the input vector, $f^{(l)}$ the activation functions, $h^{(l)}$ the layer vector output, $W^{(l)}$ the weight matrix, $b^{(l)}$ the bias vector and the top index (l) represents the l-th layer index from left to right.

$$h^{(1)} = f^{(1)}({}^{(1)}X + b^{(1)}) \tag{2.2}$$

$$h^{(2)} = f^{(2)}(W^{(2)}h^{(1)} + b^{(2)}) \tag{2.3}$$

$$h^{(l)} = f^{(l)}(W^{(l)}h^{(l-1)} + b^{(l)}) \tag{2.4}$$

2.2.1 Activation function

Typically, the activation functions used in ANN are non-linear such as: hyperbolic tangent (Equation 2.5) and sigmoid (Equation 2.6), also named standard logistic function (ISAKSSON, 2016). Recently, Rectified Linear Unit (ReLU), given by Equation 2.7, became a popular activation function, since its relatively low computational load allows the expansion in the number of layers (ISAKSSON, 2016; LECUN et al., 2015). Thus, in modern neural networks, the default recommendation is to use the ReLU function (GOODFELLOW et al., 2017). Other commonly used functions are the identity and the binary step (Equation 2.8) functions.

$$f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.5}$$

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

$$f(x) = ReLU(x) = max(0, x) \tag{2.7}$$

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{2.8}$$

An ANN with at least one internal hidden layer with enough neurons and equipped with non-linear activation functions has the property of being capable of giving an uniform approximation of any continuous function. As a consequence, any network with multiple layers using the identity as activation function is equivalent to a single layer network with a non-linear activation function (CYBENKO, 1989).

A differentiable activation function is important for gradient-based optimization methods. The ReLU is not a differentiable function and it implies in some issues with optimization methods. Activation functions with output contained in the interval $[0, 1]$ are suitable for classification problems where prediction values represent probabilities.

2.2.2 Training

An important feature on a neural network is the stage of training. During this process, weights and biases are iteratively adjusted according to the training data samples with the goal of learning their patterns.

In classification, one of the most popular use of ANNs, the training data is composed of a set of inputs and their corresponding classes. The training process compares the output given by the network with the desired class and adjusts the weights and biases to produce a better prediction on the next iteration.

In short, the processes of training a classification ANN goes through the following steps:

- the inputs are propagated through the network with the current weights and biases;
- after finishing the propagation, the outputs can be seen as a vector whose values are probabilities for each class;
- the predictions are then compared with the training reference class using a loss function (detailed on section 2.2.3); and
- the weights and biases are modified by an optimization method in a process

called back-propagation (detailed on section 2.2.5) aiming to minimize the loss function.

2.2.3 Loss function

In training process, a classifying neural network learns its parameters (weights and biases) by using training data and its associated classes. The error between the output pattern generated by the ANN for the training data (predictions) and the desired output pattern is used to compute the loss function. The loss function ($\mathcal{L}$) is used by an optimization method such as gradient descent (detailed on section 2.2.4) in back-propagation process to minimize the network error. This scalar function computes the average classification error (also called loss) over the training data sample. An important aspect of the design of a neural network is the choice of the loss function. There are many functions that could be used to estimate the error of a set of weights in a neural network, but the training is more efficient when loss functions are chosen based on the expected output data of the network.

In classification with multi-classes, it is very common to use the softmax function as activation function in the last layer, because it normalizes the output vector. They represent the probability of a network input to belong to each class in a multi-class problem. The layer with softmax function is called Softmax layer, it must have the same number of nodes as the output layer, that is the number of classes designed for the network. Equation 2.9 gives the softmax function, also known as normalized exponential function, for the i-th neuron, where $\vec{z}$ is the input vector, z_i is the i-th vector element and C is the vector length, which should be the total number of classes.

$$softmax(\vec{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} \tag{2.9}$$

The suitable loss function for outputs with probability distribution is the cross-entropy (GOODFELLOW et al., 2017). Thus, it is used as a loss function in neural networks for multi-class classification which uses softmax to output a probability distribution. The cross-entropy measures the distance between the model predictions and the desired class. Equation 2.10 is a cross-entropy function of a probability prediction $q(x)$ for input x, when the expected probability was $p(x)$.

$$H(p,q) = -\sum_{x} p(x)\ln(q(x)) \tag{2.10}$$

Equation 2.11 is the cross-entropy of the n-th training sample (s_n) on a multi-class network, where the output prediction for i-th class was $\bar{c}_i$, but the desired probability for this class was c_i, and C is the total number of classes.

$$\mathcal{L}_{s_n} = -\sum_{i=1}^{C} c_i \ln(\bar{c}_i) \tag{2.11}$$

Equation 2.12 is the loss function ($\mathcal{L}$) of a multi-class network, which is computed by the cross-entropy average ($\mathcal{L}_{t_n}$) of all the N samples of the training data.

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_{s_n} \tag{2.12}$$

2.2.4 Gradient descent

To improve the network accuracy, an optimization method is used to minimize the loss function. The most used optimization method in neural networks is the gradient descent. It attempts to minimize the loss function using the loss function gradient vector ($\nabla\mathcal{L}$). Gradient descent adjusts the network parameters towards the opposite of the gradient direction, in a way to find the nearest local minimum. Equation 2.13 shows how the gradient descent updates the network parameters, where i is the current iteration, θ_i are all the network current parameters (all weights and biases), θ_{i+1} are the updated parameters, $\nabla_\theta\mathcal{L}$ is the loss function gradient with respect to the current parameters and α is a scalar representing the learning rate. The loss function gradient is computed by the back-propagation algorithm.

$$\theta_{i+1} = \theta_i - \alpha\nabla_{\theta_i}\mathcal{L} \tag{2.13}$$

The learning rate (α) is a small positive parameter that determines how fast the optimal parameters for the model are calculated. A small learning rate may lead to more accurate parameters, but it takes more time to compute it. High learning rates can generate overshoots and miss the local minimum.

Many networks use large training sets. Because they need to generalize the model.

Using the entire set to make a single gradient update (also named a step) would take much time. Thus, to improve the training process, some optimization algorithms for machine learning usually use subsets of the training samples, called mini-batches. This process can improve the training velocity and reduce the computer memory usage (GOODFELLOW et al., 2017).

Stochastic gradient descent, the most used optimization method in ANNs, uses the mini-batches approach by randomly taking different training subsets for each step, expecting that each of them gives a good approximation of the complete set (GOODFELLOW et al., 2017). Momentum is another usual optimization method, designed to accelerate learning by accumulating the exponential decaying moving average of the past gradients and continuing to move in their direction (POLYAK, 1964).

2.2.5 Back-propagation

The back-propagation algorithm is a widely known learning process for neural networks that determines the loss function gradient with respect to each weight and bias (RUMELHART et al., 1986). This procedure propagates the loss function gradients from the last layer to the first one using the calculus chain rule (SECCO, 2014). The back-propagation provides the network parameter gradients with low computational cost to be used in an optimization method that minimizes the network error (GOODFELLOW et al., 2017). As a result of the parameters adjustment, the network hidden layers captures important features of the task domain (RUMELHART et al., 1986).

The loss function gradient with respect to each network parameter for a single training sample (Equation 2.14) is a vector composed by the partial derivative on each weight and bias for a single training sample.

$$\nabla_\theta \mathcal{L}_{s_n} = \begin{bmatrix} \frac{\partial \mathcal{L}_{s_n}}{\partial w_{11}^{(1)}} \\ \frac{\partial \mathcal{L}_{s_n}}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial \mathcal{L}_{s_n}}{\partial b^{(L)}} \end{bmatrix} \tag{2.14}$$

Equations 2.15 and 2.16 are obtained by applying the chain rule on the weights and bias partial derivatives for a single training sample, where $w_{jk}^{(l)}$ is the connection weight between the j-th neuron of layer l and the k-th neuron of layer $l-1$, $b_j^{(l)}$ is the bias value for the j-th neuron of layer l and $h_j^{(l)}$ is the output of j-th neuron of layer l, defined by Equation 2.17.

$$\frac{\partial \mathcal{L}_{s_n}}{\partial w_{jk}^{(l)}} = \frac{\partial \mathcal{L}_{s_n}}{\partial h_j^{(l)}} \frac{\partial h_j^{(l)}}{\partial w_{jk}^{(l)}} \tag{2.15}$$

$$\frac{\partial \mathcal{L}_{s_n}}{\partial b_j^{(l)}} = \frac{\partial \mathcal{L}_{s_n}}{\partial h_j^{(l)}} \frac{\partial h_j^{(l)}}{\partial b_j^{(l)}} \tag{2.16}$$

$$h_j^{(l)} = f^{(l)}(\sum_{k=1}^{n_{l-1}} w_{jk}^{(l)} h_k^{(l-1)} + b_j^{(l)}) \tag{2.17}$$

The partial derivative of $h_j^{(l)}$ by chain rule is given by Equations 2.18 and 2.19, where $f'^{(l)}$ is the activation function derivative and $z_j^{(l)}$ is the network input on the j-th neuron of layer l, presented in Equation 2.20.

$$\frac{\partial h_j^{(l)}}{\partial w_{jk}^{(l)}} = h_k^{(l-1)} f'^{(l)}(z_j^{(l)}) \tag{2.18}$$

$$\frac{\partial h_j^{(l)}}{\partial b_j^{(l)}} = f'^{(l)}(z_j^{(l)}) \tag{2.19}$$

$$z_j^{(l)} = \sum_{k=1}^{n_{l-1}} w_{jk}^{(l)} h_k^{(l-1)} + b_j^{(l)} \tag{2.20}$$

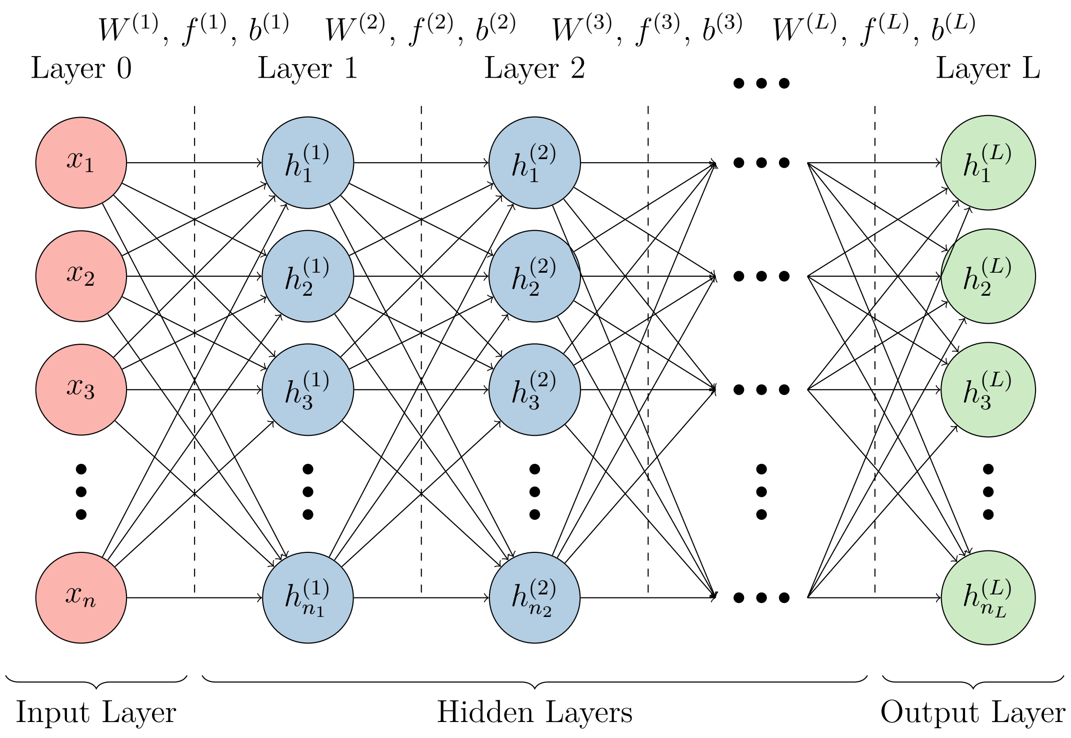
Replacing the partial derivatives above in the loss function derivatives (Equations 2.15 and 2.16) gives Equations 2.21 and 2.22. For $l = 1$, $h_k^{(0)}$ is equal to the k-th input (x_k).

$$\frac{\partial \mathcal{L}_{s_n}}{\partial w_{jk}^{(l)}} = h_k^{(l-1)} f'^{(l)}(z_j^{(l)}) \frac{\partial \mathcal{L}_{s_n}}{\partial h_j^{(l)}} \tag{2.21}$$

$$\frac{\partial \mathcal{L}_{s_n}}{\partial b_j^{(l)}} = f'^{(l)}(z_j^{(l)}) \frac{\partial \mathcal{L}_{s_n}}{\partial h_j^{(l)}} \tag{2.22}$$

The loss, for only one training sample, is defined as a function of the last layer output and of this layer target. But each layer output is a function of all the neurons from the previous layer, as shown by Figure 2.3. Therefore, the chain rule allows the calculation of the partial derivatives of the loss function with respect to one layer neuron based on the derivatives of the next layer. Except for the last layer (L), whose derivatives can be computed directly. These calculi are described on Equation 2.23.

Figure 2.3 - Diagram of generic multi-layer feed-forward ANN showing all the layers parameters, including the zero weights.



SOURCE: Adapted from Secco (2014).

$$\frac{\partial \mathcal{L}_{s_n}}{\partial h_j^{(l)}} = \begin{cases} \sum_{i=1}^{n_{l+1}} w_{ij}^{(l+1)} f'^{(l+1)}(z_i^{(l+1)}) \frac{\partial \mathcal{L}_{s_n}}{\partial h_i^{(l+1)}} & \text{for } 1 \leq l < L \\ \frac{\partial \mathcal{L}_{s_n}}{\partial h_j^{(L)}} & \text{for } l = L \geq 0 \end{cases} \tag{2.23}$$

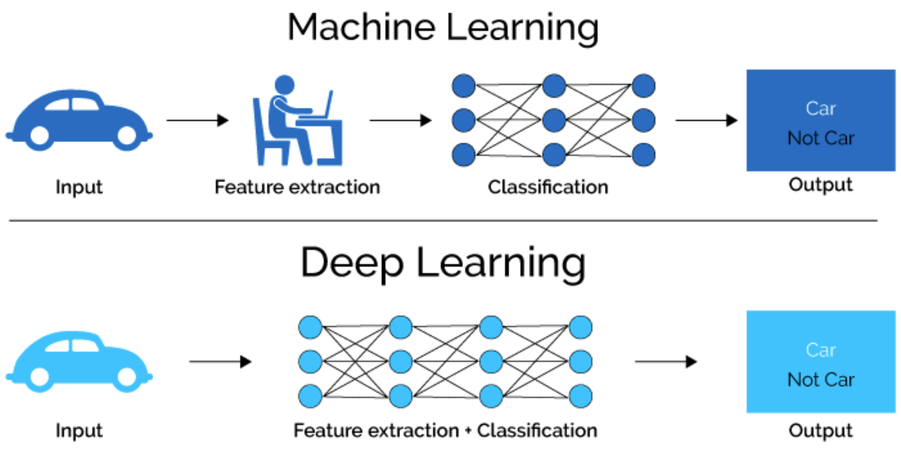
In practice, the derivatives from the last layer are computed first, which enables the flow back through the entire network using the already known derivatives to compute derivatives of the previously layer. This process computes the derivatives from each layer at a low computation cost. Back-propagation is computed on every gradient descent iteration until the network learns the desired task.

In summary, the back-propagation calculates the loss function gradient with respect to each network weight and bias for a single training sample ($\nabla_\theta \mathcal{L}_{s_n}$) using Equations 2.21, 2.22 and 2.23. The complete loss function gradient, used in optimization methods, computes the average of all the training samples.

2.3 Deep Learning

Deep Learning (DL) is a Machine Learning (ML) branch that trains the computer to perform tasks such as the human brain, including speech recognition and image classification. It generates high-level abstractions of data with a deep network composed of multiple processing layers with linear and nonlinear transformations. One advantage of DL is that it requires little human interference because even feature extraction is learned by the machine, as exemplified in Figure 2.4. One of the most known DL model is the CNN, due to the exceptional results in computer vision, mainly in image classification.

Figure 2.4 - The differences between Machine Learning and Deep Learning.



SOURCE: Xenonstack (2018).

The AlexNet CNN (KRIZHEVSKY et al., 2012) won the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) competition in 2012, where its results significantly outperformed all previous competitors, obtained an error of 15.3% against 26.2% from the second-best contest entry, whose approach averages the predictions of several classifiers, such as Scale Invariant Feature Transform (SIFT) based on (LOWE, 2004), trained on Fisher Vectors (FVs) computed from different types of densely-sampled features. The AlexNet uses a deep CNN approach with 5 layers, 60 millions parameters and 650000 neurons. This result changed the direction of object recognition and feature extraction research in the following years. Since then, ILSVRC contest winners have used the DL and CNN approach with more than 95% accuracy.

The studies of ANN, ML, DL and CNN have achieved outstanding results in several applications, becoming the state-of-the-art in many areas. In remote sensing, for example, DL has great results in hyper-spectral image classification (CHEN et al., 2014).

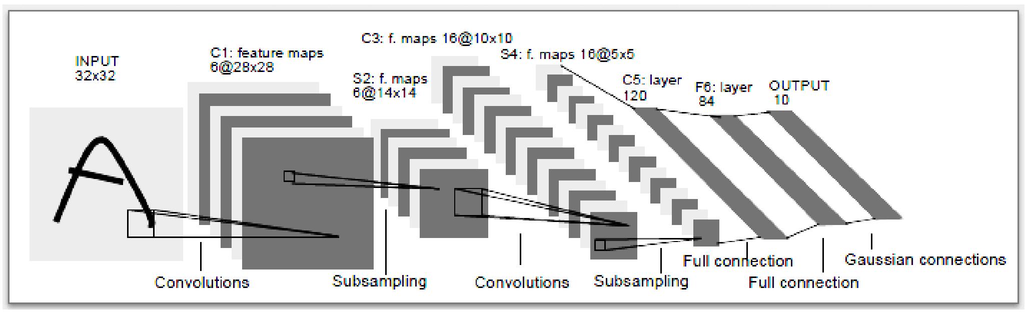
2.4 Convolutional Neural Network

CNN is a type of ANN inspired by the human visual system and designed to work with two-dimensional structured data. CNN combines convolution operations with normal ANNs. The convolutional and pooling (subsampling) layers perform feature extraction, thus retaining spatial information throughout their layers.

LeCun et al. (1989) designed the first CNN for pattern recognition, more specifically for images of hand-written numbers, that performed better than the best manual model at that time. The advances on computer processing power and the availability of large, organized and free access image databases contribute to the recent growth of CNN research. The ImageNet imaging database, for example, has over 1.2 million images for classification (DENG et al., 2009).

Figure 2.5 presents the LeCun et al. (1998) CNN architecture, where the first hidden layers are a successive sequence of convolutional and pooling (subsampling) layers, followed by fully connected layers and finally by the Softmax layer. The parameters of this network are learned through the back propagation mechanism.

Figure 2.5 - The LeCun et al. (1998) CNN architecture.



SOURCE: LeCun et al. (1998).

2.4.1 Convolutional layer

A convolutional layer performs discrete convolution operations with a filter and a two-dimensional structured data (stacked matrices), then add a bias and apply an activation function. Each convolution filter produces a linear combination of the neighborhood (called local receptive field) defined by the filter size and applies it on the entire input, so each output depends only on its local receptive field and shares its parameters with all the input.

Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivalent representations (GOODFELLOW et al., 2017). This is accomplished by small kernels shared among all the inputs, what reduces the memory requirements of the model, requires less computing operations, and improves its statistical efficiency (GOODFELLOW et al., 2017).

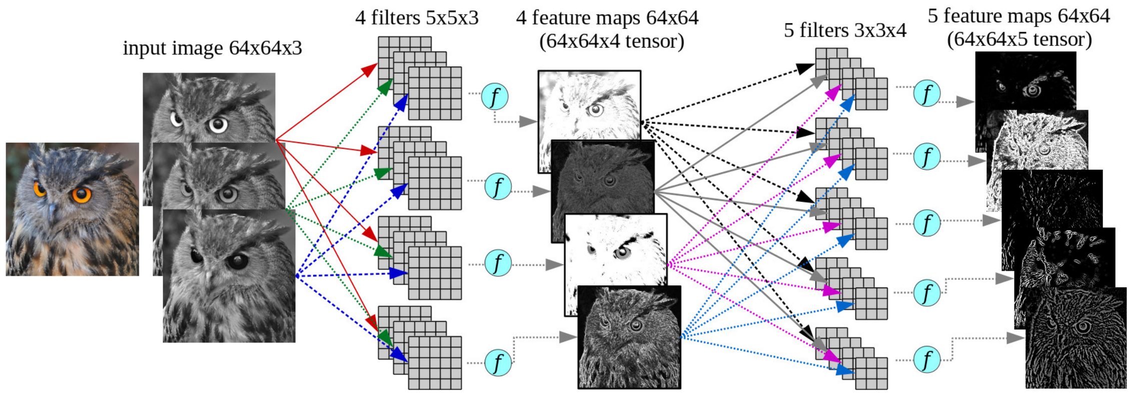
The filters used by the convolution operations are called kernels and constitute the layer weight. Their outputs are stacked matrices described by three dimensions: $(height, width, depth)$. Since the network uses the same kernel for each matrix, the same kind of feature is extracted, so the output of a convolution can see as a feature extraction operation, often called feature maps. Filters in subsequent layers have the previous feature maps as inputs, thereby extracting more complex features.

A convolutional layer is defined by the: number of filters, filter size, stride, padding, and activation function. Stride means the step size used by the filter when moving across the input and the padding is the extra border added to sides of the input. Equation 2.24 calculates output dimensions (O_h, O_w, O_d), for an input tensor (I_h, I_w, I_d) passing through a convolutional layer with N filters of size (K_h, K_w), stride (S_h, S_w) and padding (P_h, P_w), this layer would have a total of $(K_h * K_w + 1) * N$ parameters, composed by $(K_h * K_w) * N$ weights and N bias.

$$(O_h, O_w, O_d) = (\frac{I_h - K_h + 2P_h}{S_h} + 1, \frac{I_w - K_w + 2P_w}{S_w} + 1, N) \tag{2.24}$$

It's usual to use filters with the same height and width and a zero-padding. The zero-padding, border filled by 0, is commonly used to maintain the height and width input dimensions in the output. The most usual activation function is the ReLU. Figure 2.6 shows an example of a set of convolutional layers working in sequence. It is important to notice that each filter has a distinct bias.

Figure 2.6 - Illustration of two convolutional layers.



The first with 4 filters $5 \times 5 \times 3$ that gets as input an RGB image of size $64 \times 64 \times 3$, and produces a vector of feature maps. A second convolutional layer with 5 filters $3 \times 3 \times 4$ gets as input the vector from the previous layer of size $64 \times 64 \times 4$ and produces a new $64 \times 64 \times 5$ vector of feature maps. The circle after each filter denotes an activation function, e.g. ReLU.

SOURCE: Ponti et al. (2017).

2.4.2 Pooling layer

The pooling layer, also called subsampling layer, does downsampling of the features maps from the previous layer, capturing the main statistical information of small windows from the input. A pooling layer is defined by the: size, stride and pooling operation. The pooling operation runs independently on each feature map and keeps the depth dimension. This layer has no parameters, because it only does a fixed calculus, so there is no back-propagation on this layer.

A pooling operation makes a summary statistic of the local receptive field for the outputs. The most known is the max pooling operation, which captures the maximum value of a local receptive field. Another usual pooling operation is the weighted average based on the distance from the central point (GOODFELLOW et al., 2017). For many tasks, pooling is essential for handling inputs of varying size, such as, when classifying images of variable size, because the classification layer must have a fixed number of neurons (GOODFELLOW et al., 2017).

Equation 2.25 calculates output dimensions (O_h, O_w, O_d), for a input tensor (I_h, I_w, I_d) passing through a pooling layer of size (K_h, K_w) and stride (S_h, S_w), this layer has no parameters to be adjusted.

$$(O_h, O_w, O_d) = (\frac{I_h - K_h}{S_h} + 1, \frac{I_w - K_w}{S_w} + 1, I_d) \tag{2.25}$$

2.4.3 Fully connected layer

After all the convolutional and pooling layers it is usual to reorganize the last feature maps into a vector and feed it into a fully connected layer. The feature map vector works like neurons in a usual ANN. A fully connected layer is a layer where each neuron in the layer has a connection to all neurons in the adjacent layers by weighted connections and each neuron outputs a scalar. These layers are a common multi-layer ANN with weighted connections, activation functions, and biases in each neuron. These layers have the lager number of parameters, because each neuron is connected to all the next layer neurons. These layers are often followed by the Softmax layer where the classification output happens.

2.5 Regularization

An important issue in machine learning is how to make the CNN robust not only to training data but also to new input data, therefore preventing overfitting. Overfitting occurs when the network has a very good performance on the training data (low error rate) and a poor performance in presence of new data. In this sense we can say that the network became too specialized in the training data (memorizing the samples) and, at the same time, not being capable of generalization in order to cope with new data (learning the features of interest).

To avoid this problem, the training data is slit in two sets: the training set (also called train set), used to learn the network parameters, and the validation set (also called eval set), used to evaluate the generalization. A good generalization implies a low rate of error on the validation set. This fact highlights the importance to keep a third part of the data to actual test the final network. That compose the test set, which, in turn, can't be used on the training process.

Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error (GOODFELLOW et al., 2017). These strategies are known collectively as regularization. Regularization is a set of techniques which make slight modifications to the learning algorithm in a way that the model generalizes better. The most used regularization techniques are: L1 and L2 regularization, data augmentation, dropout, and early stopping.

2.5.1 L2 and L1 regularization

Many regularization approaches are based on limiting the capacity of models by adding a penalty parameter to the loss function. In neural networks, typically, only the weights at each layer are penalized by regularization term, the biases usually are not regularized (GOODFELLOW et al., 2017). The L2 and L1 regularization add a regularization term (R) on the loss function, in a way to reduce the network weights values. It assumes that a neural network with smaller weights leads to simpler models, therefore reducing the overfitting. Equation 2.26 shows the expanded loss function by adding this term, weighted by λ parameter to control the regularization influence. The L2 uses the L2-norm of all the network weights ($R = \frac{1}{2} \sum_i \mid w_i \mid^2$) while the L1 used L1-norm ($R = \sum_i \mid w_i \mid$) as a regularization term.

$$\tilde{\mathcal{L}} = \mathcal{L} + \lambda R \tag{2.26}$$

2.5.2 Data augmentation

Data augmentation is a technique that creates new data from the original one to increase the amount of data with relevant features and generalize the model (GOODFELLOW et al., 2017).

Dataset augmentation has been a particularly effective technique for object recognition problems (GOODFELLOW et al., 2017). Images include an enormous range of factors of variation, many of which can be easily simulated. Operations like translation, rotation, flips, noise addition and lighting changes are often used to improve generalization in images datasets.

2.5.3 Dropout

Dropout provides a computationally inexpensive but powerful method of regularizing a broad family of models (SRIVASTAVA et al., 2014). Dropout modifies the network during forward pass on the training stage, eliminating a random set of neurons from the hidden layers to force the network to be redundant, in a way that it must classify correctly even without some neurons. Dropout provides an inexpensive approximation to training multiple networks versions on the same data and averaging these networks.

2.5.4 Early stopping

Usually, the network stops the training after decreasing the training set error to a fixed value or after a fixed number of epochs. An epoch is one cycle of training over the entire training dataset. Unfortunately, this approach may not prevent the overfitting. In order to prevent the overfitting, a technique named Early stopping is employed. It performs a kind of cross-validation strategy that stops the training stage when the error on the test dataset consecutively increases for a certain number of iterations, regardless of the training error behavior (GOODFELLOW et al., 2017).

2.6 Transfer learning

Training a complete CNN starting with random parameters demands a large database, huge computational resources and many days of training. Many applications do not have sufficiently large datasets and enough computational resources available. Therefore, in many applications it is not feasible to train a CNN from scratch (with randomly initialized weights), and even data augmentation won't be sufficient since it only creates versions of the same images.

In this context, it is common to use the weights from a model already trained in a large dataset, such as ImageNet dataset (DENG et al., 2009) (1.2 million images), using multiple high power Graphics Processing Unit (GPUs) for several days. The pre-trained model can be used to transfer learning by its features maps. The intuition behind transfer learning is that if a model was trained on a large and general enough dataset, this model has effectively learned how to analyze images, providing a generic model for it.

Transfer learning is a machine learning method that uses a pre-trained model developed for a task as starting point for a model to solve a similar task. The pre-trained model can be used to perform feature extraction or fine-tuning. The networks' architecture most used in transfer learning for images tasks are: VGG (SIMONYAN; ZISSERMAN, 2014), Inception (SZEGEDY et al., 2015), ResNet (HE et al., 2016) and their versions.

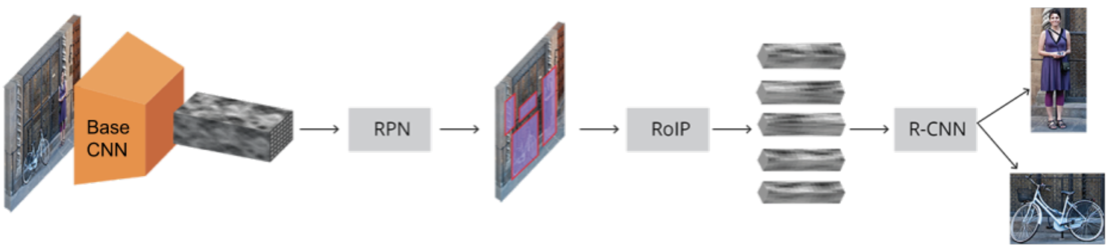
Feature extraction uses features maps from a pre-trained network without changing them. It is usual to use the last feature map vector as inputs for other classifier methods, such as Support Vector Machine (SVM). Fine-tuning uses parts of a pre-trained model in a new model and adjusts some or all of the previously trained weights in the new model training stage, allowing the weights to be fine-tuned for the new dataset.

2.7 Faster R-CNN

Faster R-CNN is a network designed to detect and classify objects in near real time, achieving the state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012 and MS COCO 2014 datasets (LIN et al., 2014; EVERINGHAM et al., 2010; REN et al., 2015). The Faster R-CNN is the evolution of two other networks: R-CNN (GIRSHICK et al., 2014) and Fast R-CNN (GIRSHICK, 2015). These three networks were developed to detect objects in rectangular regions (boxes) and classify them. They obtained top accuracy in PASCAL VOC 2007 dataset, being their main challenge was to reduce the processing time to real time. Their base algorithm looks for regions that are more likely to contain objects and performs the classification only for those regions. The major difference between these networks was the region selection approach.

Both R-CNN and Fast R-CNN networks use the Selective Search (UIJLINGS et al., 2013) algorithm to look for regions containing objects. The Selective Search presents high accuracy, but it is a slow and time-consuming process, which affects the entire network time performance. The great advantage of Faster R-CNN was the development of a Region Proposal Network (RPN) that shares convolution layers with the classifier network. This innovation considerably reduced the computational cost, making the algorithm much faster.

Figure 2.7 - Faster R-CNN architecture overview.



SOURCE: Tyrolabs (2018).

An overview of Faster R-CNN architecture is shown on Figure 2.7, whose components, respectively, are:

- Pre-trained base CNN, that works as a feature map extractor and outputs a feature map of an image.
- RPN, which receives the features maps from the previous component and generates proposals bounding boxes.
- Region of Interest (RoI) pooling, which receives the proposals bounding boxes and generates a fixed-size feature map for each proposal.
- Region-based CNN (R-CNN), that classifies each bounding box from the RoI pooling as one of the desired classes or as a background and adjusts its bounds with a logistic regression.

2.7.1 Pre-trained base network

The pre-trained base CNN, first component of the Faster R-CNN, uses convolution layers of a generic CNN pre-trained for classification task, outputting an intermediate layer of that CNN. Therefore, this component receives an image and outputs a feature map of this image. Typically, the pre-trained CNN are: VGG-16 (SIMONYAN;

ZISSERMAN, 2014), Inception (SZEGEDY et al., 2015), ResNet (HE et al., 2016) or their versions.

It is worth to emphasize that since there are no fully connected layers in this component, thus the input image can have any spatial dimensions (width and height), but its depth must be fixed to 3. After all, convolution and pooling operators don't require a fixed image size. The feature map spatial dimensions depend on the input dimensions and the CNN used.

2.7.2 Region Proposal Network

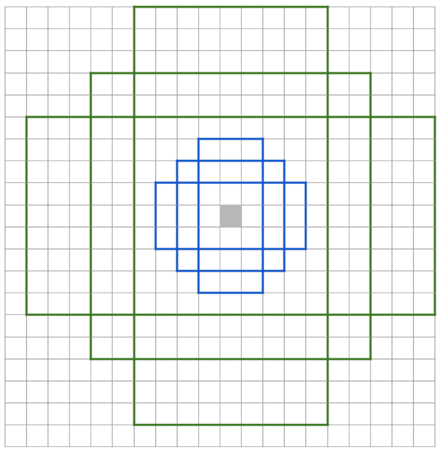
RPN generates a variable quantity of proposals regions (rectangular bounding boxes) more likely to contain objects. These proposals have distinct spatial sizes. They are generated based on a set of fixed-size bounding boxes, called anchors. A fixed number of k anchors are defined on a configuration file, based on the desired scales and aspect ratios. The anchors definitions ensure the model capability to detect different types of objects. Ren et al. (2015) uses 9 anchors, defined by the combination of 3 scales (128, 256 and 512) and 3 aspect ratios ($1:1$, $1:2$, $2:1$).

For each point in a feature map, k regions are generated. Initially, these regions are centered on their respect feature map point and have spatial dimensions (width and height) determined by the anchors scales and aspect ratios previously defined, exemplified on Figure 2.8. The RPN had a logistic regression layer to ajust these regions center (x_{center},y_{center}), width and height to better fit the objects. The RPN also has a classification layer that scores those regions in two classes: *background* and *foreground*, called *objectness* score.

Usually, an amount of regions overlaps that same object, and the Non-Maximum Suppression (NMS) is applied to avoid this redundancy. NMS ranks regions by their *objectness* score and eliminates the ones that have an Intersection over Union (IoU) larger than the threshold with a higher score region. After NMS, the top-N regions sorted by score are outputted as proposals regions, also called RoI. The proposals bounding boxed are defined by the adjusted center, width and height.

In the RPN training, the IoU, detailed on 3.5.2, between all proposals and ground truths are used to select two groups of objects for training. The proposals with an IoU$>$0.7 are considered *foreground* and those with IoU$<$0.3 are *background*, the proposals that don't fit in those criteria aren't used on the RPN training. For the classification layer a mini batches randomly sample from those proposals are used. This

Figure 2.8 - Example of 6 anchors of a point in a feature map (gray) generated by the combination of 2 scales (green and blue) and 3 aspect ratios.



SOURCE: Author's production.

mini batch are built in a way to maintain a certain balance between the two classes (background an foreground). For the logistic regression layer only the foreground regions are used. The ground truths that have the highest IoU with a foreground are used to calculate the spatial adjustments.

2.7.3 Region of interest pooling

The RoI pooling converts each proposal (which have arbitrarily size) into a fixed size feature map. A fixed size is essential for the classifier, because the fully connect layers on its structure requires fixed-size feature map ($512 \times 7 \times 7$) as input. First, the feature map (from the base network) region that belongs to each RoI are cropped and then resized to a fixed size feature map of $14 \times 14 \times 512$ ($w \times h \times d$) by applying a interpolation method (usually bi-linear). After that, a 2×2 max pooling layer is used to generate a fixed size feature map of $7 \times 7 \times 512$.

2.7.4 Region-based CNN

The R-CNN receives the fixed size feature map of each proposal and classifies each of them in one of the desired classes or as a background and adjusts its bounding box coordinates for each class. Two fully-connected layers are used in this part. One output of this network is $N+1$ class scores for each bounding box that pass through a softmax, where N is the total number of classes plus the background class. The other output is $4N$ regression adjustments ($\Delta_{x_{center}}$,$\Delta_{y_{center}}$,Δ_{width}, Δ_{height}) for each bounding box. The background is added as a class, because there will be proposals that don't contain any desired object.

The NMS is also used in Region-based CNN to avoid the redundancy caused by proposals with overlaps. The proposal must be grouped by class removing the background proposals, for then applying the NMS for each group of class. A threshold for class score and a limit for the number of objects per class can be used.

The R-CNN training is similar to the RPN, but considering different possible classes. The proposals with an IoU>0.5 with any ground truth are assigned to that ground truth. Those with 0.1<IoU<0.5 are label as background and the ones that don't fit in these rules aren't used on the training. This training focus on improving the network to solve more doubtful cases. For the classification layer a mini batch random sample from those proposals is used. This mini batch has about 25% of foreground and 75% of background. For the regression is calculated for each class using only the foregrounds assigned to the ground truth of that respective class.

2.7.4.1 Loss function

The Faster R-CNN have trainable layers on the pre-trained base CNN, RPN and R-CNN. The training of the pre-trained base network is optional. It can improve the network results but the training will spend more time. The RPN and R-CNN layers use a multi-task loss function that joins the classification loss with the bounding-box regression loss. Both use almost the same loss function, given by Equation 2.27, but each one with their own numbers of classes. The RPN classes are only: *foreground* and *background*.

$$\mathcal{L} = \frac{1}{N_{cls}} \sum_i \mathcal{L}_{cls}(p_i, p_i^*) + \lambda[p_i^* \geq 1] \frac{1}{N_{reg}} \sum_i \mathcal{L}_{reg}(t_i, t_i^*) \tag{2.27}$$

In Equation 2.27 the bounding box regions are compared with their respective ground truth. The bounding box regions are indexed by i, which predicted score label (for the ground truth class) is p_i and the predicted vector representing its 4 bounding box coordinates is t_i. The bounding box region respective ground truth class is labeled as p_i^* and is located by t_i^* coordinates. Label 0 must be assigned for *class background.* The expression $[p_i^* \geq 1]$ gives 1 when the condition is valid and 0 otherwise, this implies that the regression is not computed when the predictions are assigned as backgrounds. The classification loss $\mathcal{L}_{cls}$ is a cross-entropy over the classes and the regression loss $\mathcal{L}_{reg}$ is the robust smooth L1 (GIRSHICK, 2015). Both losses are normalized, respectively, by $Ncls$ (batch size) and $Nreg$ (the total number of regions inputted on layer). These losses are combined by a weighted sum using the λ parameter, which is capable to prioritize the classification over localization or vice-versa. Ren et al. (2015) uses $\lambda = 1$ to equally weight the losses.

2.7.4.2 Applications

Object detection based on DL has been used to detect vehicles in UAV aerial images (AMMOUR et al., 2017; PÉREZ et al., 2014). Ammour et al. (2017) CNN-based system outperforms the state-of-the-art methods on detecting and counting cars in very high spatial resolution UAV images. This system takes small segments from the UAV image, extracts features from these segments using an CNN and classify the segments using a SVM classifier into two classes: *car* and *no-car*.

Faster R-CNN has been successfully used to detect vehicles on images of surveillance cameras in urban areas (ZHANG et al., 2018) and of low-altitude UAV (XU et al., 2017). Xu et al. (2017) show that Faster R-CNN can achieve promising car detection on low-altitude UAV imagery under urban environment compared with other methods. Xu et al. (2017) also show that Faster R-CNN holds great potential for car detection in parking lots and that Faster R-CNN was robust to illumination changes and cars' in-plane rotation.

3 MATERIALS AND METHODS

As mentioned in Chapter 1, the goal of this dissertation is to develop a system based on DL capable of detecting people and vehicles on image captured by sensors arranged in captive balloons. The main part of DL methods is the training step, in which one of the most important aspects is the construction of an appropriate database. This chapter starts with the description of the materials used in this work: the sensors, the DL API and the database. Then, it describes the evaluation metrics used and, in the end, explains the methodology used.

3.1 Image sensors

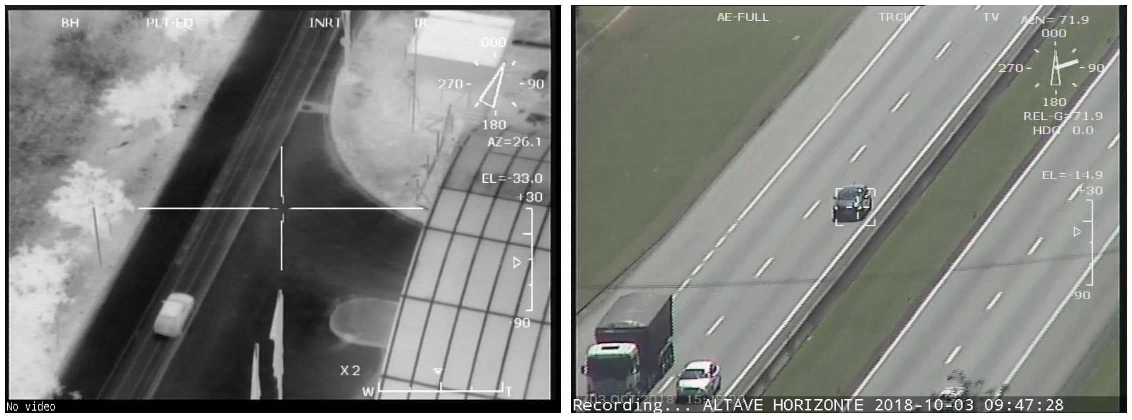
The database of images captured by captive balloons was provided by Altave Company. The aforementioned company uses captives balloons for continuous monitoring of large areas. The current surveillance standard involves a person continuously watching the video feed continuously to detect possible discrepancies. There is only one camera arrangement in Altave's captive balloons. This camera has two sensors: one infrared and one visible. Table 3.1 describes the spectral bands and grid size (pixels) of these sensors. The infrared sensor has two possible acquisition modes: *white hot* and *black hot*, changing the hotter to cooler gray scale from white to black or from black to white, respectively. The infrared acquisition mode is determined by the operator. Figure 3.1 shows image samples for each sensor from Altave's captive balloons.

Table 3.1 - Sensors spectral bands and grid size (pixels).

Sensor	**Spectral Band**	**Grid size (pixels)**
Infrared	3.0 - 5.0 μm	640×480
Visible	0.4 - 0.7 μm	752×576

SOURCE: Adapted from Altave (2019).

Figure 3.1 - Samples of images from Altave's captive balloons infrared and visible sensors, respectively.



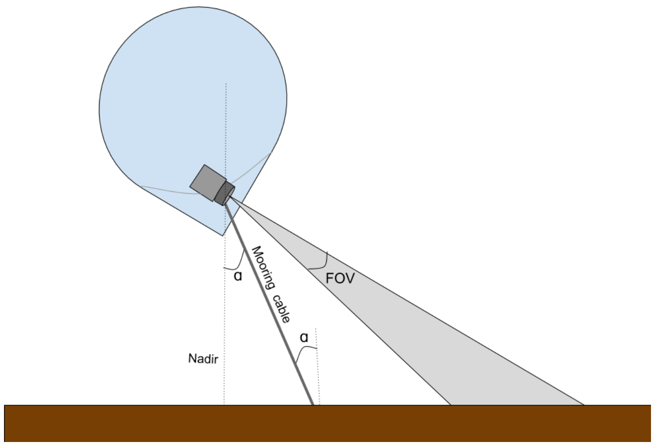
SOURCE: Altave (2019).

Altave image acquisition hardware can only provide videos from one sensor at a time, which are not orthorectified. Therefore, when the visible sensor is selected, the system outputs only RGB videos, and when the infrared sensor is selected, it outputs only gray scale videos. Hence, it is impossible to construct a multi-band image. This work uses 3-bands 8-bits color (0 to 255) images extracted from these videos. The visible sensor provides RGB images and the infrared sensor provides 3-bands grayscale images, repeating the same data on the three channels.

Although the captive balloons are tethered in the ground, they have 6 degrees of freedom. They can execute 3 rotations (roll, pitch and yaw) and 3 translations (x, y and z). Because of the tether, these movements are limited, but are sufficient to classify the aerostat as a non-fixed platform. In consequence, the images acquired by their sensors are non-static.

The camera arranged in the captive balloon is able to pan, tilt and zoom. These capabilities, combined with the balloon movements, generates a great variety of fields of view (FOV), depending on the camera and the balloon position, what changes depending on the wind direction and strength. Figure 3.2 shows an example of a balloon FOV, normally off nadir because its nadir view only captures scenes close to its ground base. These angular FOV and camera movements produce images with objects in different scales and points of view, creating the need of a technology capable to deal with these images particularities.

Figure 3.2 - Schematic that shows the captive balloon (in blue) tethered to the ground and an example of a FOV captured by the camera sensor (in gray) arranged in it. The angle α is measured between the nadir and balloon mooring cable.



SOURCE: Author's production.

3.2 TensorFlow Object Detection API

This work was developed using open source tools, such as TensorFlow (ABADI et al., 2016) and Python programming language. The main tool used to implement the CNN object detection model was a TensorFlow Object Detection API (TF, 2019c), which can create state-of-the-art neural networks for object detection proposals. This API is an open source framework built on top of TensorFlow that simplifies the construction, training and evaluation process for object detection models (TF, 2019c).

This framework was created by Huang et al. (2017) to evaluate the speed/accuracy trade-off among the object detection models. The framework recreates the following networks meta-architectures: SSD (LIU et al., 2016), Faster R-CNN (REN et al., 2015) and R-FCN (DAI et al., 2016). The speed/accuracy trade-off on these three meta-architectures models concludes that, in general, the SSD and R-FCN models are the faster ones, while Faster R-CNN is the highest accuracy, but is slower (HUANG et al., 2017).

This framework can perform the training, evaluation and testing steps on 3-band images with any width and height for one of these models. It also allows the visualization of the model loss function on the TensorBoard, during the training. There are configurations files available for each model, with some parameters to be set. Although, this framework simplifies the model building, it is limited by the parameters available on the configuration file, *e.g.*: the early stopping technique is not implemented for Faster R-CNN model.

3.3 Computer

A dedicated computer is essential to run the training of object detectors CNN, because it can reduce the training time from days to hours. In this work, the free platform provided by Google Colaboratory (Colab) was used to train and run the network. Colab provides a free Jupyter Notebook environment for every 12 hours. The Colab version used in this work was configured with Python 3.6, TensorFlow 1.15 and the following hardware:

- GPU: Tesla K80 12 GB;
- CPU: single core Intel(R) Xeon(R) @ 2.30GHz;
- RAM: 12.5 GB;
- Disk: 310 GB.

3.4 Database

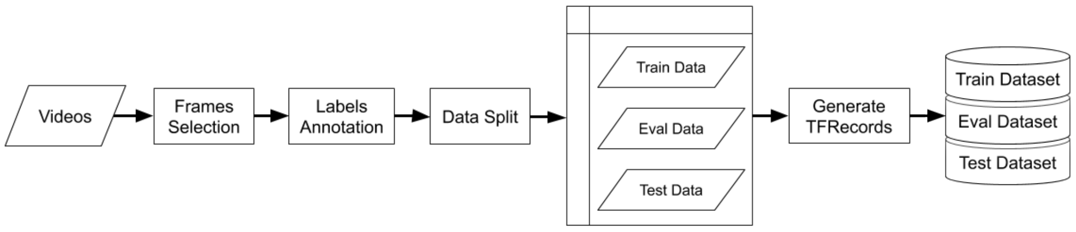
One of the most important tasks in DL training is the building of an appropriate database. CNNs learn from the samples fed to them, so the samples must be generic enough to cover all possibilities, but they also have to avoid ambiguity. Another important feature for CNN training database is the balance among classes.

Two databases were constructed, one for each sensor described on Section 3.1 (visible and infrared). This strategy was used because the acquisition hardware can't capture images from both sensor simultaneously. These databases have only two classes: *vehicle* and *person*. The *vehicle* class includes cars, vans, trucks and buses. Bicycles and motorcycles were not included into vehicle class because there were few samples with these objects, and they are easily confused with the *person* class. The choice to use a wide class, such as vehicle instead of using classes like car and truck, came from the fact that car and trucks are very similar from some points of views, what could cause some ambiguity in the model.

In this document, the term *object* refers to all objects from these two classes detected on the images. It is important to notice that, in object detection problems, a unique image can contain multiple objects from different classes, which may overlap.

Figure 3.3 shows the building process to generate a complete database. This process was performed for the infrared and visible sensors separately. The inputs provided by Altave were videos. The first step was to manually select frames from these videos in a PNG (Portable Network Graphics) format.

Figure 3.3 - Database building process diagram.



SOURCE: Author's production.

The ground truth information is generated by manually labeling each object on the database images. The LabelImg (TZUTA LIN, 2019) tool was used to annotate the bounding boxes on the images. It is a graphical open source image annotation tool, based on Python and Qt. This tool supports two annotations formats: PASCAL VOC and YOLO. In this work, the PASCAL VOC format was used to save the annotations (labeled bounding boxes coordinates) as Extensible Markup Language (XML) files.

The databases were randomly split into three datasets: Train, Eval and Test, respectively, with 63%, 15% and 22% of images from the complete database. The train dataset is used to fine-tune the network; the eval set to evaluate the model during the fine-tune training, and to avoiding making the network too specialized (overfitting); and the test is used for accuracy assessment. The TensorFlow Object Detection API receives inputs on TFRecord format (DAT TRAN, 2018), which is a simple record-oriented binary format supported by Tensorflow (TF, 2019a) containing the ground truth data and the associated images. Each dataset (Train, Eval and Test) was converted to a TFRecord file. The open source Python scripts made by Dat Tran (2018) were used to convert the XML annotations into CSV (Comma Separated Values) files, and finally the CSV files together with the images generated the TFRecord files.

As previously discussed, the database must cover all possible objects variety without being ambiguous in a way to avoid overfitting. For this purpose, some criteria were defined for the frame selection task:

- the objects must have enough details to be recognized by a human being without considering the video features (such as speed) and context, *e.g.*: a rectangular box moving fast in a highway is probably a vehicle, but it must not be labeled as vehicle;

- the objects must have at least 32 pixels on width and height;

- the overlapped objects must be completely clear, one object can not be used to find another;

- the repetition of very similar images must be avoided;

- the infrared sensor must have black hot and white hot samples; and

- the database must have objects on a variety of angles, zooms, points of views, and noise.

A database following the above criteria was built from the videos provided by Altave. Tables 3.2 and 3.3 contain a summary of the two databases built, one for the infrared and other for the visible sensor. Each database contain about 700 images but the visible sensor database is slightly greater because there were more visible sensor videos.

Table 3.2 - Infrared database summary.

Dataset	Nº Images	Images	Nº Vehicle	Nº Person	Vehicle	Person
Train	410	63%	467	343	58%	42%
Eval	98	15%	116	62	65%	35%
Test	141	22%	165	136	55%	45%
Total	649	100%	583	343	63%	37%

SOURCE: Author's production.

Table 3.3 - Visible database summary.

Dataset	Nº Images	Images	Nº Vehicle	Nº Person	Vehicle	Person
Train	456	63%	675	689	49%	51%
Eval	108	15%	212	81	72%	28%
Test	160	22%	294	146	67%	33%
Total	724	100%	887	689	56%	44%

SOURCE: Author's production.

In the training process, it is important to keep the number of objects of each class fed into the network approximately equal (balanced). Hence, the network should not prioritize any class over the others. The captive balloon was placed close to a highway. Therefore, the number of *vehicle* samples is greater than the number of *people* samples. The data samples were randomly distributed in the datasets in a way to ensure a reasonable balance on the training dataset, as seen on Tables 3.2 and 3.3.

3.5 Evaluation metrics

Evaluation metrics are used to measure the quality of the model. Since object detection problems predict locations and classes of multiple objects in a unique image, and both need to be evaluated, the common accuracy metric used in classification problems can't be applied in object detection tasks. Furthermore, object detectors may take into account the background, which works as a class, but with some particularities. Even though the ground truth contains the background indirectly (all the pixel not contained on the bounding boxes are considered as background), it is impossible to count the number of background bounding boxes.

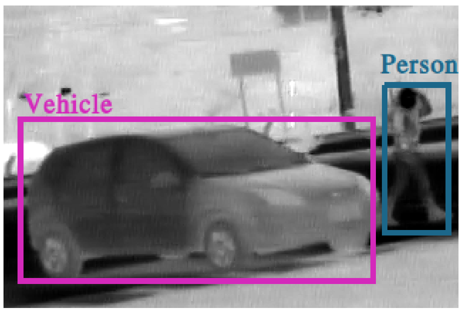
An important trade-off in object detection tasks is the quantity of ground truth detected versus the quantity of background wrongly detected as an object. A metric called mean Average Precision (mAP) was created to cover all these particularities and to take into account this trade-off. There are some variations on mAP implementation. COCO mAP (LIN et al., 2014), PASCAL VOC mAP (EVERINGHAM et al., 2010) and the confusion matrix are the main metrics used to evaluate an object detector model. The COCO API (COMMON OBJECTS IN CONTEXT - COCO, 2018) provides the visualization of some evaluation metrics during the training. The following sections describe the definitions used and the specifications of the mAP metrics.

3.5.1 Ground truth

The foundation of an evaluation metric is the ground truth data that was used to compare with predictions. The ground truth of a single image on the dataset is composed by a bunch of objects, each one with two annotations: the class label and the bounding box location (x, y, width and height). Figure 3.4 shows an example of a ground truth image, the colors and labels on the image are just for visualization.

On Faster R-CNN networks, each predicted object has bounding box location and a main class label together with its Confidence Score (CS). CS can be interpreted as the probability of a class label assigned to the predicted object. A CS threshold is used in evaluation metrics to eliminate predictions with lower CS. All the ground truth objects are considered to have a 100% CS.

Figure 3.4 - Example of a ground truth image.

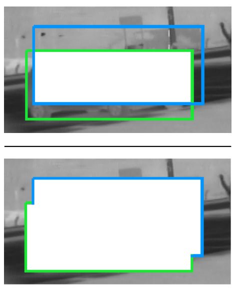


SOURCE: Author’s production.

3.5.2 Intersection over Union

A metric commonly used to determine the correctness of object location is given by the ratio between the intersection and the union of two regions, also known as IoU (Equation 3.1). IoU is a quality metric to evaluate the common area between a ground truth bounding box (B_{gth}) and its corresponding predicted bounding box (B_{pred}). IoU score ranges from 0 to 1, the closer the two boxes areas are, the higher the IoU score.

$$IoU = \frac{area(B_{pred} \cap B_{gth})}{area(B_{pred} \cup B_{gth})} = \tag{3.1}$$



3.5.3 True Positive, False Positive and False Negative

In object detection tasks only the predictions with CS above a certain threshold (T), $CS \geq T_{CS}$, are considered. The location of an object is considered correct if IoU between the ground truth and its prediction is above a certain threshold (T), $IoU \geq T_{IoU}$. More specifically, three detection results can arise from these thresholds:

- True Positive (TP), the prediction object has same class label and similar location, in other words, $CS \geq T_{CS}$ and $IoU \geq T_{IoU}$.

- False Positive (FP), the prediction object class is different from the ground truth class ($CS \geq T_{CS}$ for a wrong class) and similar location ($IoU \geq T_{IoU}$). This implies that the ground truth was miss classified. Another FP occurs when $CS \geq T_{CS}$ but $IoU < T_{IoU}$. This implies that the algorithm detected a background region on the image that does not belong to any class.

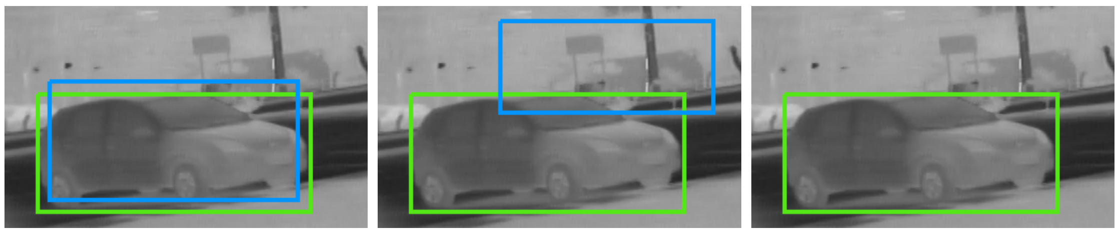
- False Negative (FN), the ground truth sample is not detected, so it is considered a background.

- True Negative (TN) refers to background object correctly detect, but since there is no sample for background, it is not take into account.

It is possible to have multiple boxes detected for the same ground truth object. In this case, the box with the highest IoU is considered a TP, while the remaining are considered FP.

Figure 3.5 exemplifies TP, FP and FN. Actually, the FP example has one FP and one FN, because as the blue box has an IoU lower than its threshold (thus being a FP) and no object detected by the network matches the labeled one so it has a FN on the ground truth vehicle.

Figure 3.5 - Examples of TP, FP and FN instances, from left to right. The green rectangles are the ground truth object and the blue ones are the predictions, all boxes sampled belong to *vehicle* class.



SOURCE: Author's production.

3.5.4 Precision × recall

Precision, defined by Equation 3.2, measures the false positive rate, which is the number of correct detections divided by all boxes detected. Precision represents the model ability to identify relevant objects. Precision scores indicate the correct prediction probability, ranging from 0 to 1. Higher scores imply that most detected objects match the ground truth ones. Therefore, most positive predictions are, in fact, correct.

$$Precision = \frac{TP}{TP + FP} = \frac{\text{all correct detections}}{\text{all boxes detected}} \tag{3.2}$$

Recall, defined by Equation 3.3, measures the false negative rate, which is the number of correct detections divided by all ground truth boxes. Recall represents the model ability on finding relevant objects. Recall scores indicate the probability of ground truth object being correctly detected, ranging from 0 to 1. Higher scores imply that most ground truth objects were detected. Therefore most object labeled were correctly detected.

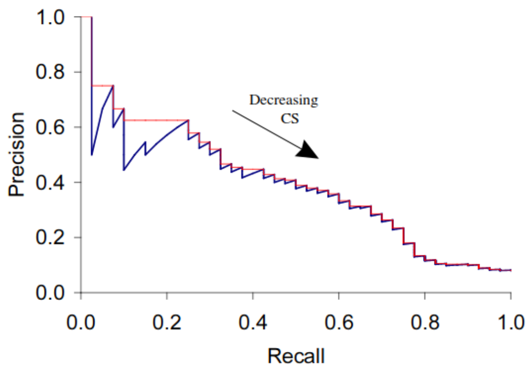
$$Recall = \frac{TP}{TP + FN} = \frac{\text{all correct detections}}{\text{all ground truth boxes}} \tag{3.3}$$

There is an important trade-off between precision and recall metrics depending on the CS and IoU thresholds. High CS and IoU thresholds increase the model robustness on positive samples, but decreases the number of correct predictions. In summary, when the FN increases the FP decreases, the recall get worse while the precision improves.

In order to evaluate the performance of an object detector as the CS threshold changes, a precision $\times$ recall curve is built for each object class. In this curve, the recall constantly increases when the CS threshold decreases, while precision tends to decrease. To avoid noise on this curve, the precision values for a certain recall level ($p(r)$) are interpolated ($p_{interp}(r)$) for the maximum precision found for any recall level $r' \geq r$, as in Equation 3.4. Figure 3.6 shows an example of a precision $\times$ recall curve and its interpolation curve.

$$p_{interp}(r) = \max_{r' \geq r} p(r') \tag{3.4}$$

Figure 3.6 - Example of a generic precision $\times$ recall curve (blue line) and its interpolation curve (red line).



SOURCE: Adapted from Manning et al. (2009).

3.5.5 Mean Average Precision

Even though the precision × recall curve is useful to evaluate an object detector model, it is not easy to compare models using a curve. In order to summarize this curve in a single number, the area under its interpolation curve is estimated, resulting in a measure called Average Precision (AP). AP is the averaged precision across all recall values.

Thus, Equation 3.5 calculates the AP, where $r_1, r_2, ..., r_k$ are the recall levels in ascending order. After calculating the AP, the mAP is simply the AP average over all N model classes. Equation 3.6 shows the expression for mAP calculation. It is important to notice that an IoU threshold must be chosen to calculate the regular AP and, in consequence, the regular mAP.

$$AP = \sum_{i=1}^{k}(r_i - r_{i-1})p_{interp}(r_i) \tag{3.5}$$

$$mAP = \frac{1}{N}\sum_{n=1}^{N} AP_i \tag{3.6}$$

The PASCAL VOC mAP sets IoU threshold on 0.5 and is named $mAP@.50IOU$. COCO average mAP (in short COCO mAP) , in turn, sets different IoU thresholds, starting at 0.5, going up to 0.95 with a 0.05 step, averaging the computed mAP values afterwards. It is usually called $mAP@[.50 : .05 : .95]IOU$. Therefore, COCO mAP not only averages AP over all classes but also over an set of IoU thresholds. There's also a strict mAP, that sets IoU threshold on 0.75 and is named $mAP@.75IOU$. As the name suggests, higher IoU means a more demanding metric for assessing location.

This work only used $mAP@.50IOU$ and $mAP@.75IOU$ to evaluate the model predictions as, on the final application, because the purposed application uses a fixed IoU. A complete mAP like $mAP@[.50 : .05 : .95]IOU$ would be a meaningful measure to evaluate a new object detector model, which is not this work's goal.

3.5.6 Average Recall

In addition to precision-recall curve, there is the recall × IoU curve, which summarizes the recall distribution across a range of IoU thresholds, usually from 0.5 up to 1. COCO challenge (LIN et al., 2014) uses this curve to calculate a metric called

Average Recall (AR), which is computed by doubling the area under the recall × IoU curve, as shown in Equation 3.7. The AR is calculated for each class, then the average over all classes is computed, but COCO keeps the same name for the average. AR is useful for evaluating the effectiveness of predicted detections and the model localization accuracy. COCO uses three variations on AR metric by limiting the maximum numbers of detections per image on 1, 10 and 100, respectively called $AR@1$, $AR@10$ and $AR@100$.

$$AR = 2\int_{0.5}^{1} recall(IoU)dIoU \tag{3.7}$$

3.5.7 Confusion matrix

The traditional confusion matrix is also used in this work to evaluate and visualize the model performance. A script provided by Santiago Valdarrama (2019) was used to generate a confusion matrix, where:

- the rows represent the ground truth;
- the columns represent the predictions;
- each class has a row and column;
- the last row and column belongs to the background; and
- the last row and column element is always empty, because the background ground truth doesn't exist;

The background row measures the quantity of predicted objects that do not match any ground truth object, thus not belonging to any ground truth class. The background column computes the quantity of ground truth objects (on each class), that weren't detected.

The confusion matrix eliminates predictions with CS lower than 0.5 and only considers matches with IoU higher than 0.5. The confusion matrix can be computed for each class separately to facilitate the computation of their corresponding the precision, recall and accuracy. The accuracy in object detection problems is estimated by Equation 3.8, because it is not possible to estimate TN in this type of problem. The confusion matrix and accuracy are the most meaningful evaluation measures for security task, because in the final usage a fixed value of CS must be defined to filter what actually is or not a threat (person or vehicle).

$$accuracy = \frac{TP}{TP + FP + FN} \tag{3.8}$$

3.6 Methodology

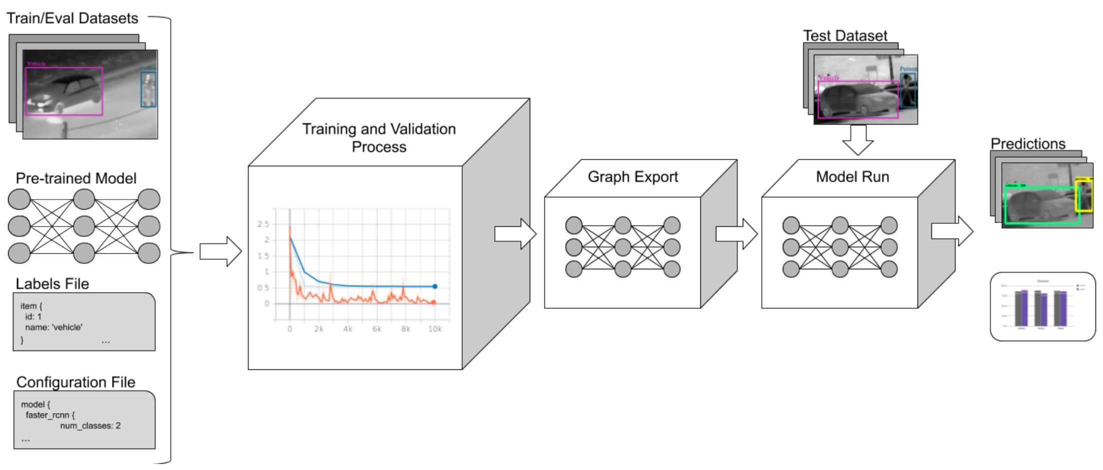
In the last years, neural networks have become the leading method for high quality object detection (HUANG et al., 2017). Therefore, this work uses a methodology based on CNNs to perform object detection. CNNs object detectors models are able to locate multiple objects on an image and label them in one of the model classes. One advantage of using DL techniques, such as CNN, is their robustness on images with complex scenes, as long as they are properly trained. Considering the complexity of scenes captured by sensors arranged in captive balloons, CNNs are the most appropriate approach to perform object detection on those images. This approach can learn the variety of angles, zooms, points of view and even noises presented on the image database built on this work.

This work prioritizes accuracy over speed, because the model will be implemented in a computer with GPU and high RAM capacity, as described in Section 3.3. For this reason, the Faster R-CNN (REN et al., 2015) meta-architecture, which tends to lead to slower but more accurate results (HUANG et al., 2017), was chosen and implemented to perform object detection in captive balloons images.

3.6.1 Methodology process

The methodology applied to build a Faster R-CNN object detector using TensorFlow Object Detection API is shown on Figure 3.7. It is based on the transfer learning from a pre-trained model to a fine-tuning model, optimizing the network to work better on samples like the ones held on the train dataset. Two networks were built, one for each sensor, and they were designed to detect only two classes: *person* and *vehicle*. This strategy was adopted because the acquisition hardware does not allow the simultaneous use of multiple sensors. The same pre-trained model and configuration file was used for both sensors, for the purpose of further comparison.

Figure 3.7 - Schematic showing the main steps of the methodology.



```
item {
  id: 1
  name: 'vehicle'
}                    ...
```

```
model {
  faster_rcnn {
            num_classes: 2
...
```

SOURCE: Author’s production.

The first step is the model training and validating process. The training step is the most critical procedure. It is important to use the correct optimization parameters and techniques to avoid overfitting, and the eval dataset is used for this task. In this step, TensorFlow API receives as inputs: the train and eval datasets, the pre-trained model, the labels file, and the configuration file.

The configuration file defines the model parameters, such as the number of classes, the model architecture, and the optimization parameters. The model architecture must be the same as the pre-trained model, and it is composed by the meta-architecture (Fater R-CNN) and the feature extractor CNN model. The labels file must enumerate all classes used by the final model, according to the labels defined in the dataset.

The model starts with the pre-trained model weights and biases, then it is trained by the train dataset samples. During the training, the eval dataset is used to verify if the model is overfitting or not. After the training, the last state of the model and its complete architecture are exported to a graph, process known as freezing the trained model. The graph is the final model, and it is ready to predict multiple objects locations and labels in input images. In order to test the model accuracy, the test dataset is given to the graph and its predictions are compared with the test dataset ground truth to generate the accuracy confusion matrix and other evaluation metrics.

3.6.2 Fine-tuning

Faster R-CNN is a model with a large number of nodes and layers, what increases the capacity of learning and leads to a more accurate model. In consequence, it requires more computational capacity, and training large models from scratch is computationally expensive, may take many days even on powerful computers. Furthermore, larger models requires large training datasets. Tiny datasets, like the ones used in this work, quickly overfit in large models, because their capacity of learning is much bigger than the dataset content. A dataset to train a model like this from scratch must contain more than a thousand images.

In order to speed up training and avoid overfitting, pre-trained models were used as initial state in the start of the training. The TensorFlow API (TF, 2019b) provides some object detectors models trained on large public datasets, such as COCO (LIN et al., 2014) and Kitti (GEIGER et al., 2013). All these datasets are composed by RGB images or videos. A model trained on COCO dataset was used in this work, because COCO has a large dataset with classes similar to the ones proposed, such as *person*, *car*, *truck* and *bus*. Although it has classes like *person* and *car*, it can't achieve accurate results in the dataset built in this work because of its particularities. Hence, the model must be fine-tuned to this particular dataset to improve the accuracy on this type of data.

Table 3.4 contains the trade-off: speed versus quality of the main Faster R-CNN architectures models available on TensorFlow API that were trained on COCO dataset (TF, 2019b). The speed is the running time for a 600×600 image on a GPU computer with an Nvidia GeForce GTX TITAN X. The quality is the COCO mAP measure performed by the model on COCO validation subset. The model highlighted in this table (faster_rcnn_resnet101_coco) was chosen for this work because it has a high accuracy and a fair speed; the next more accurate model has about 6 times more running time.

Table 3.4 - Trade-off: speed(ms) versus COCO mAP evaluation for Faster R-CNN architectures models trained on COCO dataset available on TensorFlow API.

Model Name	Speed(ms)	COCO mAP
faster_rcnn_inception_v2_coco	58	0.28
faster_rcnn_resnet50_coco	89	0.30
faster_rcnn_resnet101_coco	**106**	**0.32**
faster_rcnn_inception_resnet_v2_atrous_coco	620	0.37
faster_rcnn_nas	1833	0.43

SOURCE: Adapted from TF (2019b).

3.6.3 Configuration parameters

TensorFlow Object Detection API has some settings that may vary depending on the model and are selected by the configuration of the input file on training. The API adjusts the model nodes and layers based on these configurations - *e.g.*: by setting 2 classes, it builds a model prepared to output the results for only two classes, reducing the number of nodes on the last layer. The configuration parameters chosen to perform the training on the Fater R-CNN meta-architecture were:

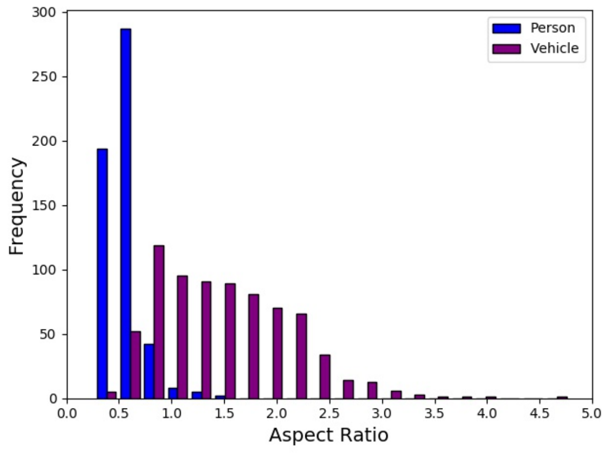
- number classes: 2;
- keep aspect ratio resizer: minimum dimension 400;
- feature extractor type: faster_rcnn_resnet101;
- aspect ratios: 0.5, 1.0 and 2.0;
- scales: 0.3, 0.5, 0.8 and 2.0;
- first stage max proposals: 300;
- use dropout: true;
- data augmentation options: random horizontal flip;
- batch size: 1; and
- learning rate: initial value 3×10^{-4}, then reduces to 3×10^{-5} at step 7000 and finally reduces to 3×10^{-6} at step 8000.

There are other configurations, but those were kept on default. The same parameters were used for the infrared and visible training. As explained previously, the model was built to work with only two classes: *vehicle* and *person*, so the parameter number of classes must be 2.

Since, a pre-trained model (faster_rcnn_resnet101_coco) was used, the feature extractor type must have the same architecture, the faster_rcnn_resnet101 one, to maintain consistency. In order to keep the image size and ratio, the minimum keep aspect ratio resizer dimension was reduced to 400, since the infrared sensor height is 480 and the visible is 576.

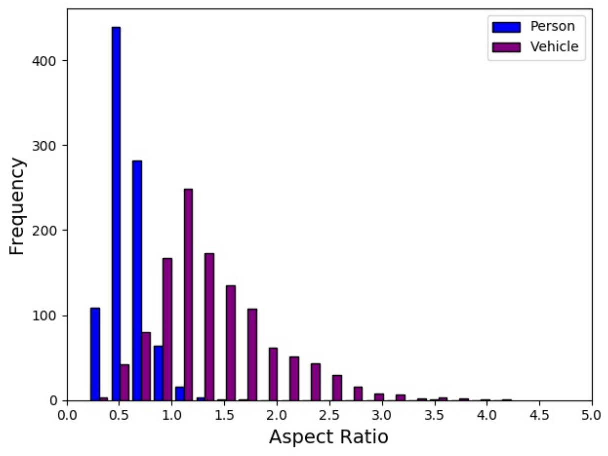
The anchors aspect ratios and scales were chosen based on the distribution of these features on the databases objects. Ren et al. (2015) recomends the selection of three or four values for each parameter. The scale parameters on this API are based on the area of the base anchor (256×256 pixels). Figures 3.8 and 3.9 show the aspect ratio distributions for all the objects on the infrared and visible datasets. Both aspect ratio charts have the maximum frequency of about 0.5 and 1.0, and thus, these values were selected. The aspect ratio 2.0 was selected, because it can better cover the remaining data.

Figure 3.8 - Infrared database aspect ratio frequency for the *person* and *vehicle* classes.



SOURCE: Author’s production.

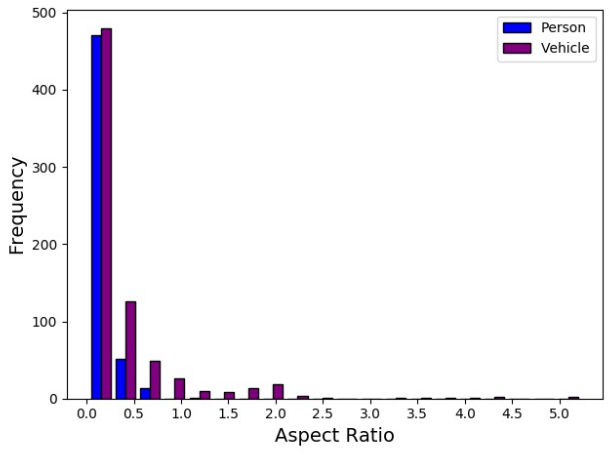
Figure 3.9 - Visible database aspect ratio frequency for the *person* and *vehicle* classes.



SOURCE: Author's production.

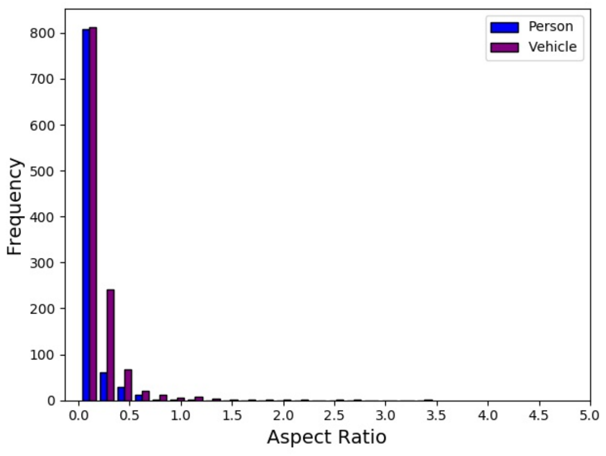
Figures 3.10 and 3.11 show the scale distributions for all the objects on the infrared and visible datasets. A higher scale is better than a small one, because the model works better when the complete object is inside the anchor, after all, it helps the detection of the presence of the object, thus the model can adjust its box location. As both scale charts have peaks between 0.25 and 0.5 scale, the scales 0.3 and 0.5 were selected to cover this range of scale. Furthermore, the 0.8 and 2.0 scales were also selected because they can better fit the remaining data.

Figure 3.10 - Infrared database scale frequency for the *person* and *vehicle* classes.



SOURCE: Author's production.

Figure 3.11 - Visible database scale frequency for the *person* and *vehicle* classes.



SOURCE: Author's production.

Huang et al. (2017) conclude by their experiments on the Faster R-CNN model that the number of box proposals per image must be from 10 up to 300. Their tests found that only 10 proposals can quickly lead to high accuracy, but higher values of proposals lead to even more accurate models, taking more time to train, however. As this work prioritizes accuracy over speed, the first stage max proposals was set to 300.

The training of large CNN models with small datasets can easily lead to overfitting the model. There are some techniques used to avoid it, such as dropout, data augmentation, and early stopping. Although TensorFlow Object Detection API does not have early stopping technique implemented, the loss, mAP, and AR curves of the eval dataset were analyzed during training with the TensorBoard platform to manually stop the training in case the model overfits.

TensorFlow Object Detection API enables the use of dropout and data augmentation, these approaches were applied during the training to improve the model learning, while avoid overfitting. Only the random horizontal flip option was selected for data augmentation, because the other options modify the image spectral properties or crop the objects, which could generate inappropriate data by losing the main object feature.

The default configurations for Faster R-CNN uses 0.9 momentum and stochastic gradient descent with mini-batches of size 1, which are the better configurations for training this model, thus they were kept. A batch size 1 could seem small, but it is not because a single image has multiple objects with different sizes. Furthermore, the existence of multiple objects makes just batch size 1 to use high loads of memory, being another reason to keep batch size on 1.

Even thought the pre-trained model had objects in distinct points of view from the visible and infrared datasets, it had classes such as *car*, *truck*, and *person*, that are very similar to the classes proposed in this work (*person* and *vehicle*). Thus, the model overfits in fewer steps, under 25000. The recommended values of learning rate were kept (3×10^{-4}, 3×10^{-5} and 3×10^{-6}), but the steps were changed to make a better fit. Therefore, the model starts with learning rate on 3×10^{-4}, reduces at step 7000, because the network starts stabilizing, and reduces again at step 8000, because at this point the network just need small adjustments to optimize the predictions without overfitting.

4 RESULTS AND DISCUSSIONS

This chapter presents the results and discussions for the two trained networks: the one with the infrared database (infrared network) and the other with the visible database (visible network). For each network we discuss the loss function behavior during training, the evaluation metrics during training, and the final results achieved. Finally, some images are shown in order to view and discuss the results achieved. In the end, there is a final discussion comparing both networks.

All the curves generated during training were smoothed by a TensorBoard default smooth function with 0.6 weight parameter. This function is an exponentially weighted moving average, essential to reduce the oscillatory behavior in a way to observe the data overall trend, which helps the search for the optimal training step. Equation 4.1 gives the smooth output vector ($[s_0, s_1, ..., s_n]$) values for a input vector $[x_0, x_1, ..., x_n]$ for a w weight parameter.

$$s_t = \begin{cases} x_0, & t = 0 \\ w \times s_{t-1} + (1 - w) \times x_t, & t > 0 \end{cases} \tag{4.1}$$

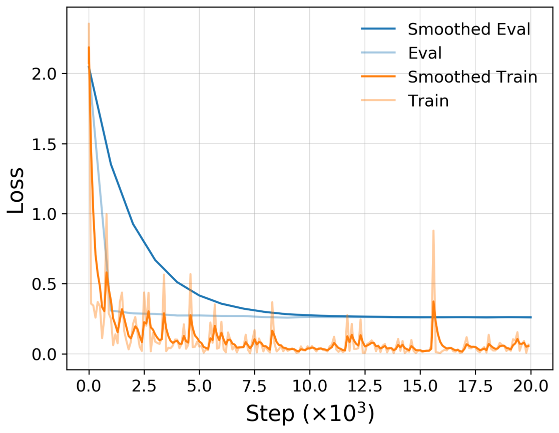
4.1 Infrared network

4.1.1 Loss function

A Faster R-CNN model was trained on TensorFlow Object Detection API according to the methodology explained on section 3.6.1 using only the infrared database. The model was trained up to $20k$ (20×10^3) steps, what takes 1h and 35min. Figure 4.1 shows the loss for the eval and train datasets during the training, where the train loss was taken every 100 steps and the eval loss was taken every 1000 steps. As can be observed in Figure 4.1, the loss rapidly converges, as the training starts from a pre-trained model with similar classes. The oscillatory behavior on the training loss is a consequence of batch size 1.

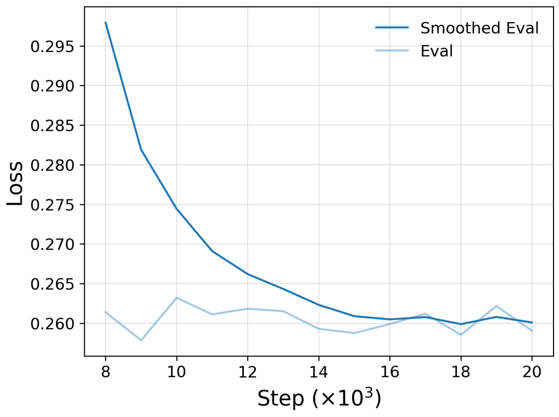
Figure 4.2 also shows the loss, but only for eval values from step 8k to the end (step 20k) in a large scale. On this scale it is clearly seen, in the smoothed curve, that the eval loss tends to decrease up to step 16k, so the training on this step was selected to build the final model.

Figure 4.1 - Infrared network eval and train dataset losses.



SOURCE: Author's production.

Figure 4.2 - Infrared network eval loss in large scale from steps: $8k - 20k$.

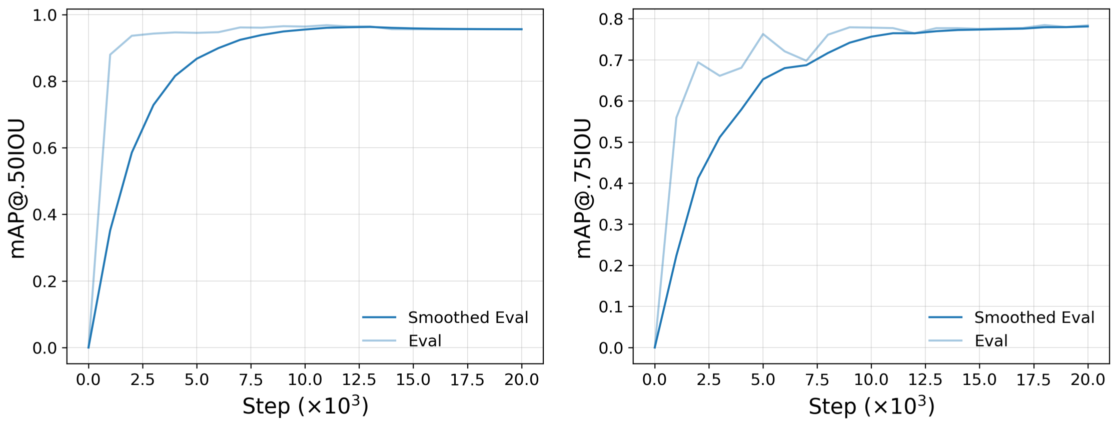


SOURCE: Author's production.

4.1.2 Evaluation metrics

The infrared network eval dataset mAP (Mean Average Precision) behavior during training is presented on Figure 4.3. The $mAP@.50IOU$ curve peaks on step $13k$ with mAP of 0.963, then stabilizes at about step $16k$ around mAP 0.956, so it decreases 0.7% from the peak. The $mAP@.75IOU$, in turn, presents a small growth until step $20k$, where it has mAP 0.781, and on step $16k$ it has mAP 0.777, so it increases by 0.4%.

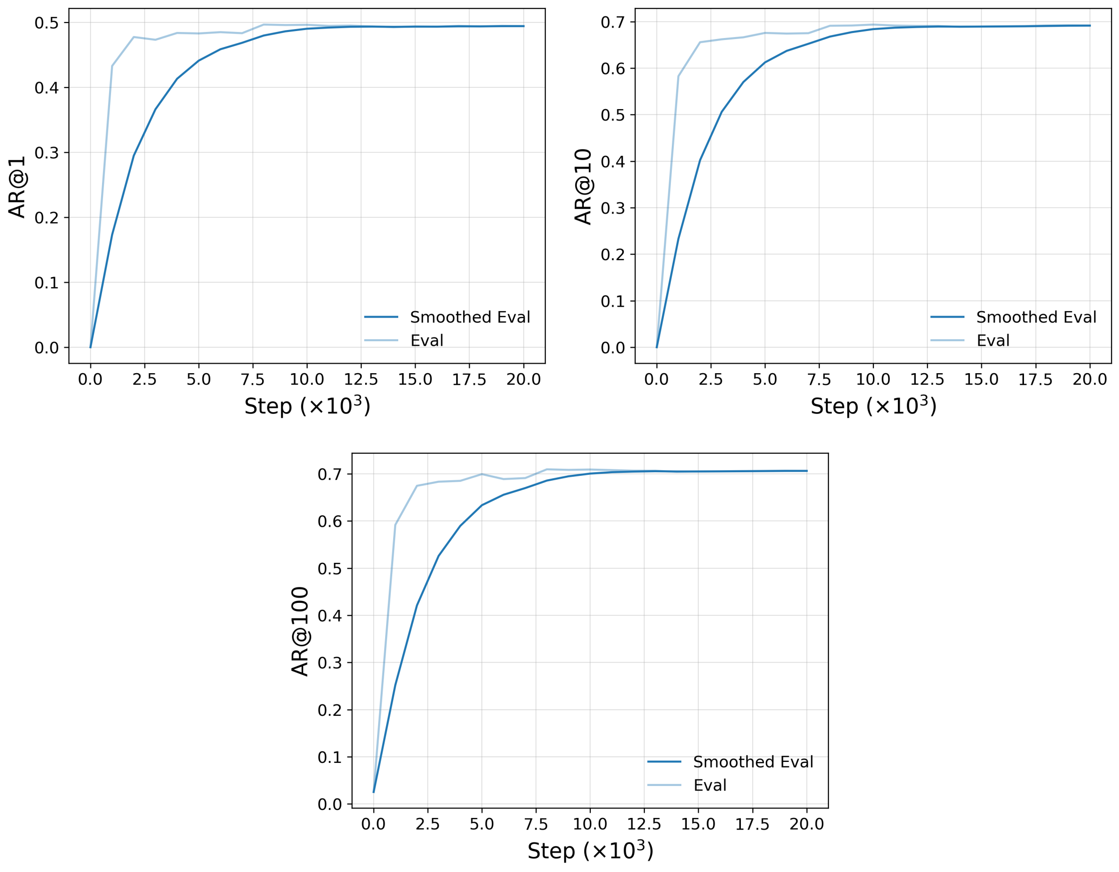
Figure 4.3 - Infrared network eval dataset mAP behavior during training.



On the left, $mAP@.50IOU$; on the right, $mAP@.75IOU$.
SOURCE: Author's production.

The infrared network eval dataset AR (Average Recall) behavior during training is presented on Figure 4.4. The $AR@1$ curve stabilizes around step $16k$ around AR 0.494. The $AR@10$ curve presents a small growth up to step $20k$, where it has AR 0.691, and on step $16k$ it has AR 0.690, so it increases on 0.1%. The $AR@100$ curve also stabilizes at about step $16k$ around AR 0.706.

Figure 4.4 - Infrared network eval dataset AR behavior during training.



On the top left, AR@1; on the top right, AR@10; on the bottom center, AR@100.
SOURCE: Author's production.

All these curves present the expected behavior, since the AR starts very small and tends to stabilize during training. Their analysis show that any step after $13k$ represents a suitable trained model, and the steps $13k$, $16k$ and $20k$ show the best performance on for each curve, respectively. Therefore, the choice of the model trained until step $16k$, based on the loss curve and explained on Section 4.1.1, is reinforced by mAP and AR curves as the optimal number of steps.

As previously explained, the training on step $16k$ was chosen to build the final infrared network, which was exported to a graph. The results achieved by this graph on the infrared datasets are shown on Tables 4.1 through 4.7. As expected, Tables 4.1 and 4.2 show that the infrared train dataset metrics are better than the others, as the model was fitted on it. Furthermore, the eval dataset performed better than the test dataset, even though they are close. This is also expected, since the eval dataset is used as reference to choose the best training step. Both mAP metrics reached high performance, being $mAP@.50IOU$ metric the highest one, as expected, because it has a lower IoU. It reaches 0.940 mAP, which is very close to the maximum value (1.0). Therefore, the $mAP@.50IOU$ demonstrates that the model got an excellent performance on object detection.

Table 4.1 - Infrared database mAP metrics.

	mAP@.50IOU	**mAP@.75IOU**
Train dataset	0.994	0.984
Eval dataset	0.956	0.777
Test dataset	0.940	0.718

SOURCE: Author's production.

The AR metrics reached by the model (Table 4.2) indicate that the model predictions were accurate on classification and localization. The AR@100 metric was the highest AR, reaching 0.706 AR.

Table 4.2 - Infrared database AR metrics.

	AR@1	**AR@10**	**AR@100**
Train dataset	0.561	0.917	0.925
Eval dataset	0.494	0.690	0.706
Test dataset	0.411	0.688	0.696

SOURCE: Author's production.

Tables 4.3, 4.4 and 4.5 contain the confusion matrix (considering T_{CS}=0.50 and T_{IOU}=0.50) obtained on the infrared datasets for the background and classes: *vehicle* and *person*. The train dataset presented 96.2% of accuracy, which is expected because the model learning was based on it. The eval and test datasets reach, respectively, 85.3% and 87.1% of accuracy meaning that the model presented a high performance. Usually, the eval dataset presents a better result than the test one, but the opposite is also possible.

Table 4.3 - Infrared train dataset confusion matrix for background and classes: *vehicle* and *person*.

		Predicted		
		Vehicle	Person	Background
Ground truth	Vehicle	460	0	7
	Person	0	343	0
	Background	21	4	—

SOURCE: Author's production.

Table 4.4 - Infrared eval dataset confusion matrix for background and classes: *vehicle* and *person*.

		Predicted		
		Vehicle	Person	Background
Ground truth	Vehicle	109	0	7
	Person	0	60	2
	Background	16	4	—

SOURCE: Author's production.

The test dataset confusion matrix shows that the model correctly detected most of the *vehicle* and *person* objects, only failing on 9 *vehicles* and 8 *people*. Furthermore, 16 objects were incorrectly detected as *vehicle* and 9 as *person*. Still, the model was able to achieve 87.1% accuracy, which proves its high performance.

Table 4.5 - Infrared test dataset confusion matrix for background and classes: *vehicle* and *person*.

		Predicted		
		Vehicle	Person	Background
Ground truth	Vehicle	156	0	9
	Person	1	128	7
	Background	16	9	—

SOURCE: Author's production.

Tables 4.6 and 4.7 also contains the test dataset confusion matrix, but for each class separately. In these matrices, the predictions that were wrong about the class are considered background. These matrices provide a clearer view of each class performance. So, the model achieves accuracy of 86.2% for *vehicle* and 88.3% for *person* class. This result proves that the model present high accuracy on both classes, being a little better for detecting *person* than *vehicle*. This probably happens because *person* objects in infrared images contrast with the background due to its higher temperature. There are also much more types of *vehicle* objects in a wider range of angles of view on the database, making it hard for the network to find a pattern.

Table 4.6 - Infrared test dataset confusion matrix for background and *vehicle* class.

		Predicted	
		Vehicle	Background
Ground truth	Vehicle	156	9
	Background	16	—

SOURCE: Author's production.

Table 4.7 - Infrared test dataset confusion matrix for background and *person* class.

		Predicted	
		Person	Background
Ground truth	Person	128	8
	Background	9	—

SOURCE: Author's production.

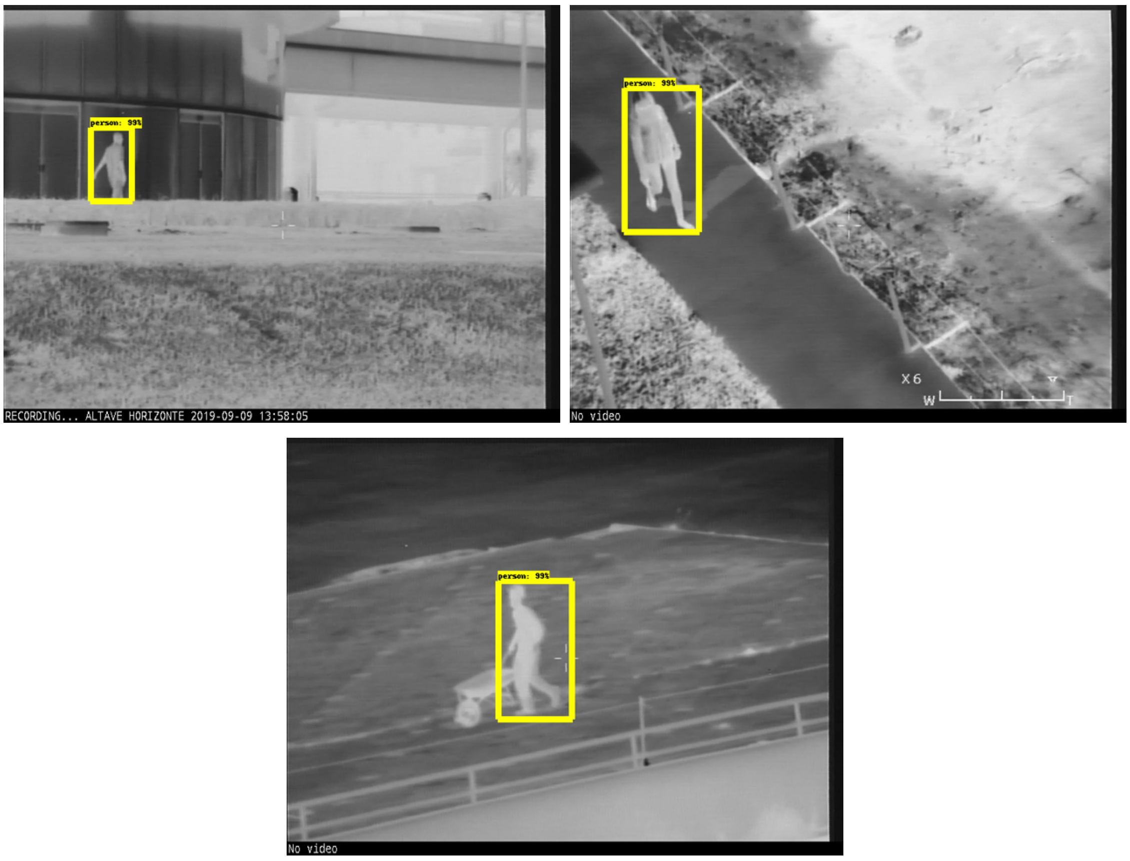
4.1.3 Images

This section presents some examples of images from the infrared test dataset with their respective predicted objects to provide a visualization of the results achieved by the trained infrared network. The images showed in this chapter are from the test dataset, same used on the confusion matrix. Therefore, for all images analysis T_{CS}=0.50 and T_{IOU}=0.50 were used to define if an object was correctly detected or not. The images are presented in two sections: Section 4.1.3.1 only contains images in which all objects were correctly detected and Section 4.1.3.2 contains images with at least one mistake. In all images, yellow and green boxes represent, respectively, *person* and *vehicle* objects predicted by the network. Blue and purple boxes, in turn, represent, respectively, the ground truth of *person* and *vehicle* objects not predicted by the network. It is worth mentioning that just the ground truth that helps the visualization were drawn on the images.

4.1.3.1 Correct detections

Figure 4.5 presents some examples of images with *person* objects in a variety of angles of view and positions. These correctly predicted objects demonstrate the high performance achieved by the network on detecting *person* objects.

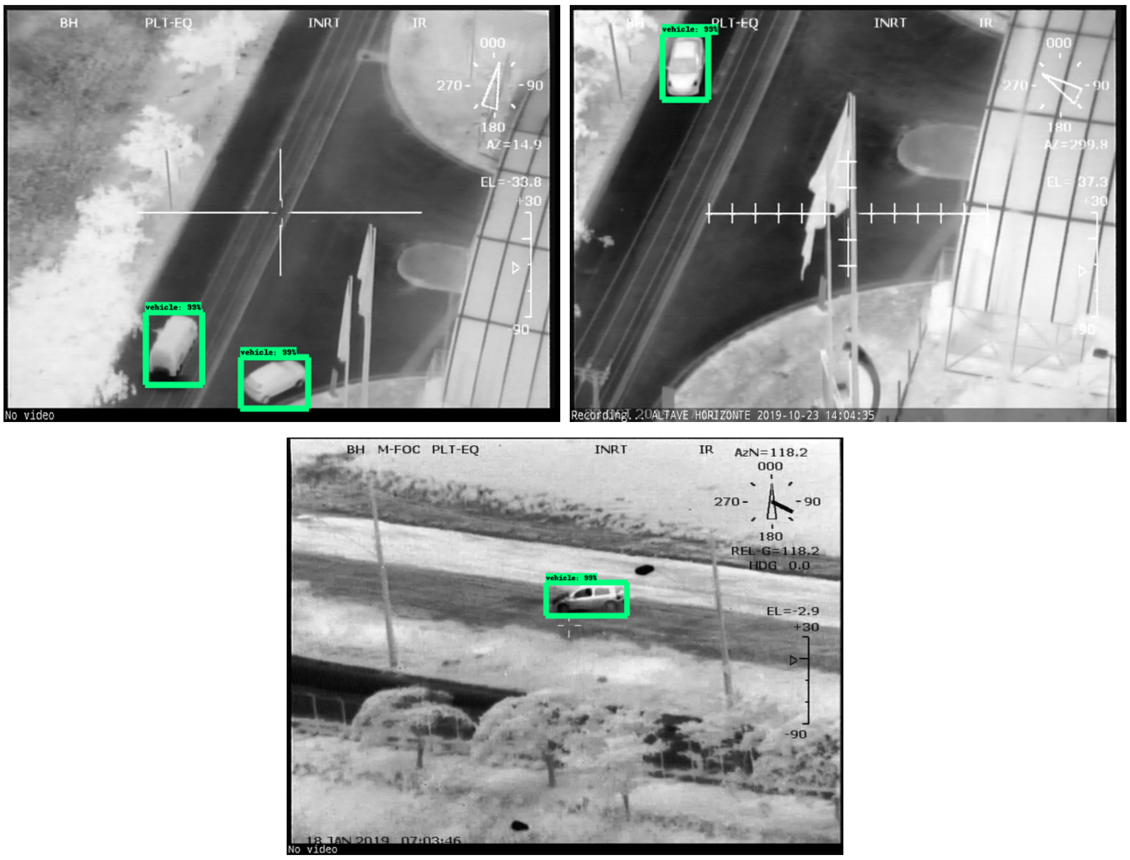
Figure 4.5 - Correct detection examples of infrared images with *person*.



SOURCE: Author's production.

Figure 4.6 presents some examples of images with *vehicle* objects in a variety of angles of view and positions. All these objects correctly predicted demonstrates the high performance achieved by the network on detecting *vehicle* objects.

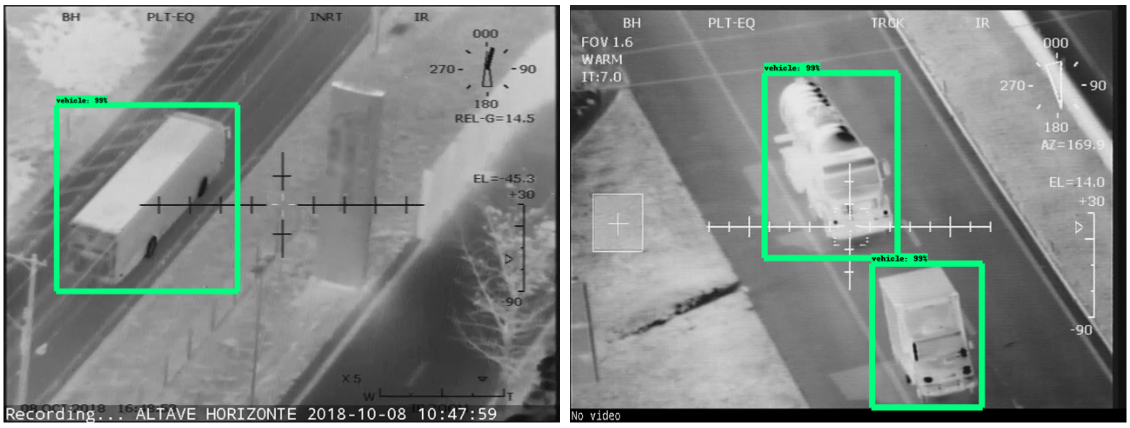
Figure 4.6 - Correct detection examples of infrared images with *vehicle*.



SOURCE: Author's production.

Figure 4.7 presents some examples of images with objects from *vehicle* class like buses and trucks in distinct points of view correctly detected. These objects correctly predicted demonstrate the high performance achieved by the network in detecting a variety of types of *vehicles*, with distinct shapes and points of view.

Figure 4.7 - Correct detection examples of infrared images with truck and bus.

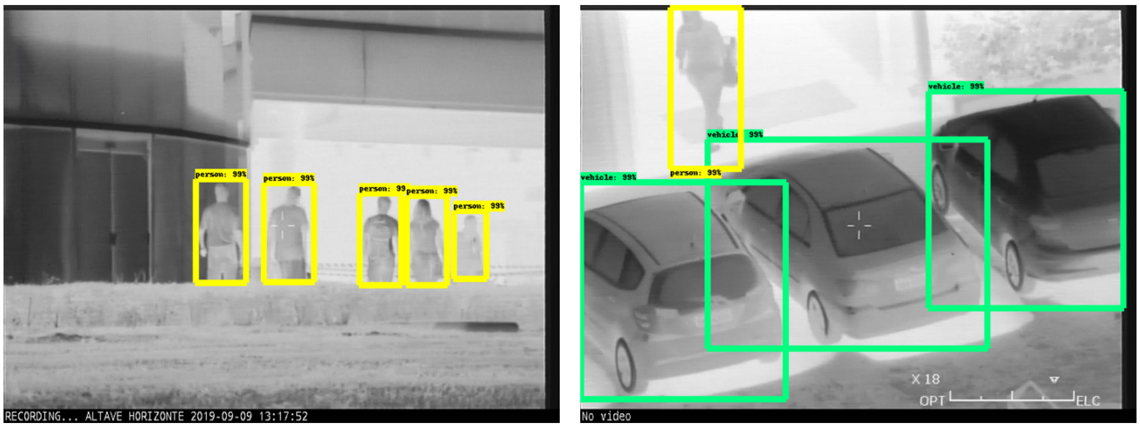


Object from *vehicle* class like bus (left) and trucks (right). Logos were blurred, just for visualization.

SOURCE: Author's production.

Figure 4.8 presents some examples of images that contain multiples objects. This figure demonstrates that the network can detect multiple objects in an image, including overlapped object boxes. The network can deal with overlapping bounding boxes, as long as the objects are not hiding most of the other.

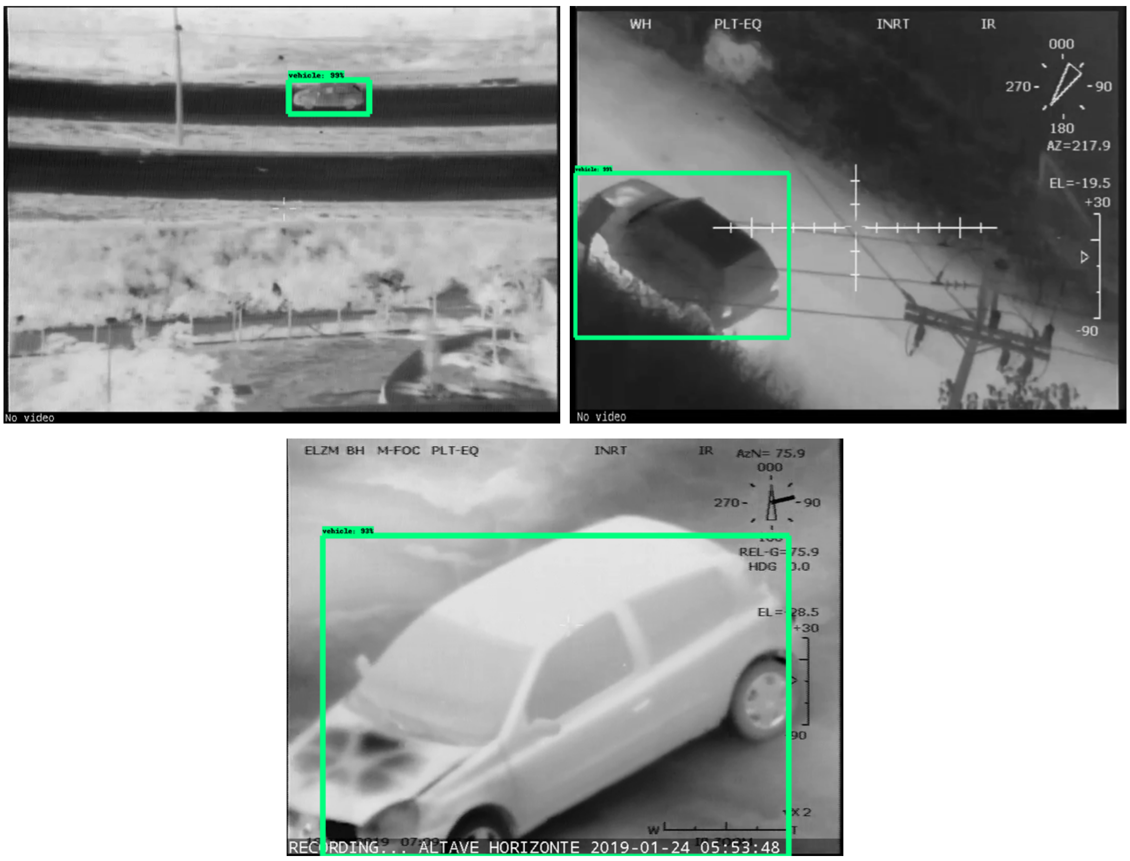
Figure 4.8 - Correct detection examples of infrared images with multiple objects.



SOURCE: Author's production.

Figure 4.9 presents some examples of images that contain *vehicles* in distinct sizes. This Figure suggests that, as intended, the network is able to detect objects in a variety of sizes, from small (left) to medium (right) and even the large (bottom) ones. The wide range of anchor sizes chosen in the model configuration covers almost all object sizes, contributing to these positives results.

Figure 4.9 - Correct detection examples of infrared images with very distinct *vehicle* scales.

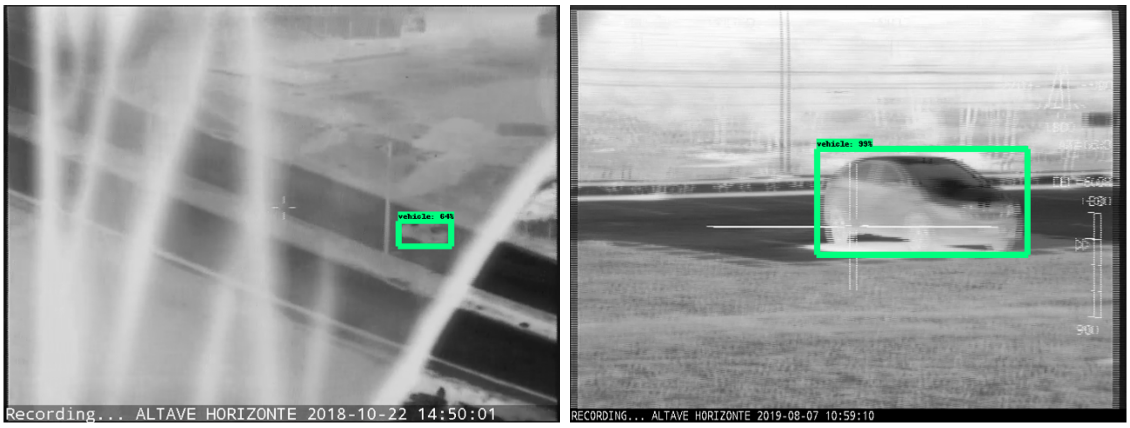


Vehicles on a variety of sizes: small (top left), medium (top right) and large (bottom center).

SOURCE: Author's production.

Figure 4.10 presents some examples of images with noisy objects. Therefore, this Figure evidences that, as intended, the network is able to detect objects with some noise, like the ones caused by the balloon fabric overlap and the camera motion.

Figure 4.10 - Correct detection examples of infrared images with noise.

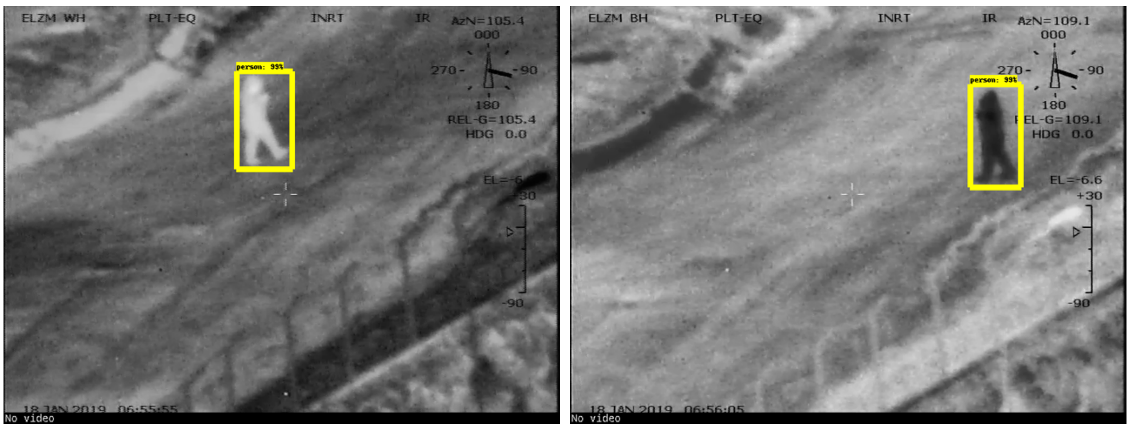

vehicle: 64%
Recording... ALTAVE HORIZONTE 2018-10-22 14:50:01
vehicle: 99%
RECORDING... ALTAVE HORIZONTE 2019-08-07 10:59:10

Vehicles with some noise caused by the balloon fabric overlap (left) and the camera motion (right).

SOURCE: Author's production.

Figure 4.11 presents some examples of infrared images in white hot and black hot modes. Thus, this Figure demonstrates that the network is able to detect objects in both infrared modes, as proposed.

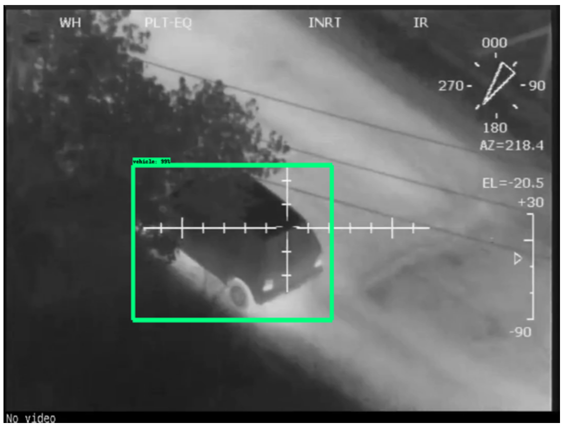
Figure 4.11 - Correct detection examples of infrared images with white hot (left) and black hot modes (right).



SOURCE: Author's production.

Figure 4.12 demonstrates the network ability to detect partially hidden objects, like the *vehicle* on this image, which was partially covered by a tree. Partially hidden objects can be correctly detected by the network, as long as their main features can be seen. After all, the network doesn't maintain any information from previous images.

Figure 4.12 - Correct detection example of infrared images with a partially hidden *vehicle*.

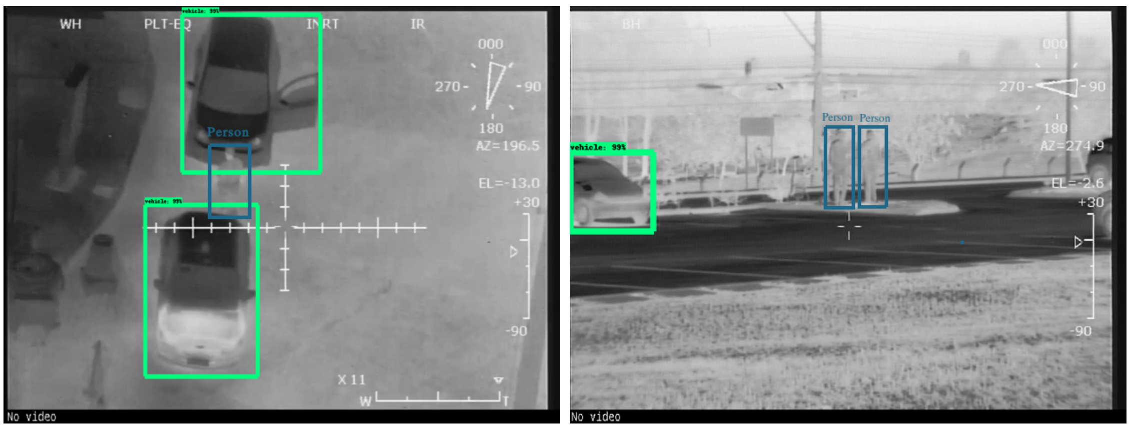


SOURCE: Author's production.

4.1.3.2 Mistakes

Figure 4.13 shows *person* objects present in the ground truth that the network wasn't capable to detect. The *person* objects in both images had less contrast, making them less apparent in this context, thus their main features were difficult to see. This impacts the network detection, affecting these mistakes.

Figure 4.13 - Mistaken detection examples of infrared images with undetected *person*.

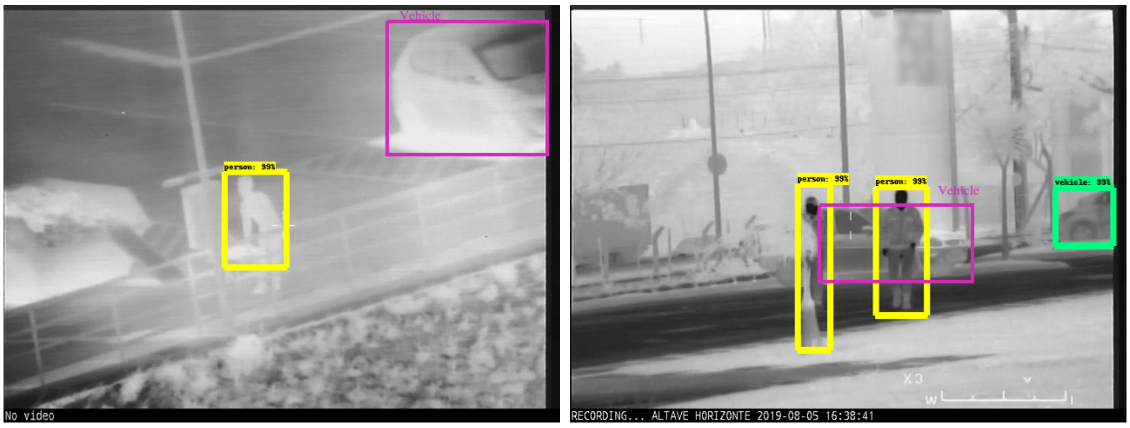


On the left, two *vehicles* correctly detected and *person* not detected, in blue. On the right, a car correctly detected and two *person* objects not detected, in blue.

SOURCE: Author's production.

Figure 4.14 shows *vehicle* objects present in the ground truth that the network was not capable of detecting. The *vehicle* on Figure 4.14 (left) had a high brightness on the wheels, what has an effect similar to hiding the wheels. The *vehicle* on Figure 4.14 (right) is partially hidden by the people in front of it, and all of its wheels are covered. The network is not so robust to hidden objects, mainly when important object features, like the wheels, are hidden. Thus, this limitation leads to the mistakes.

Figure 4.14 - Mistaken detection examples of infrared images with undetected *vehicle*.

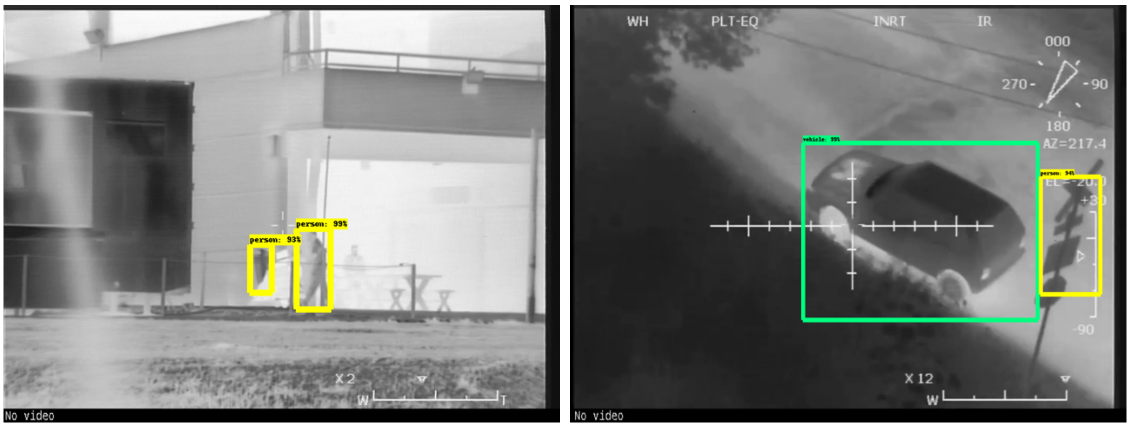


On the left, a *person* correctly detected and *vehicle* not detected (in purple). On the right, two *person* objects and a *vehicle* correctly detected and a *vehicle*, in purple, not detected. Logos were blurred, just for visualization.

SOURCE: Author's production.

Figure 4.15 shows some incorrect predictions, on which part of the background was detected as *person* objects. The trash can and the traffic sign were wrongly detected as *person* objects, as they present some features, probably related to their shape, that make the network believe they are *person* objects. Their shape has some features that resembles head and arms, what could lead to the mistakes. To avoid this type of mistakes, more images with objects that resemble *person* objects are essential to provide the network enough information to learn features that only belong to *person*.

Figure 4.15 - Mistaken detection examples of infrared images with background detected as *person*.

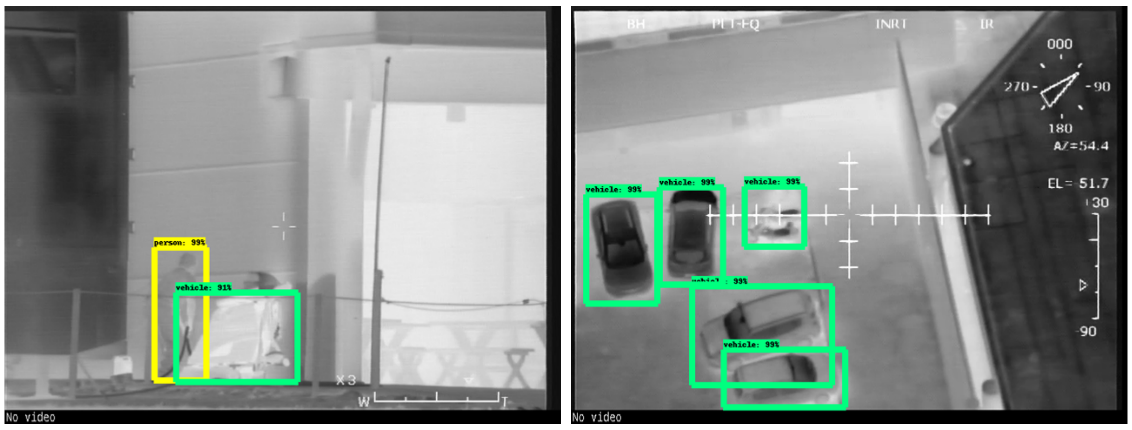


On the left, a *person* correctly detected and a trash can detected as a *person*. On the right, a *vehicle* correctly detected and a traffic sign detected as a *person*.

SOURCE: Author's production.

Figure 4.16 shows some incorrect predictions where parts of the background were detected as *vehicles*. The trash bag and the two motorcycles, wrongly detected as *vehicles*, present some features that make the network believe they are *vehicle*. The motorcycle wheels have the same features as the *vehicle* ones, what could lead to the mistake. To avoid this, more images with *vehicles* objects and with objects that there aren't *vehicles* but could resemble them are essential to provide the network enough information to enable it to learn the features that only belong to *vehicle*.

Figure 4.16 - Mistaken detection examples of infrared images with background detected as *vehicle*.

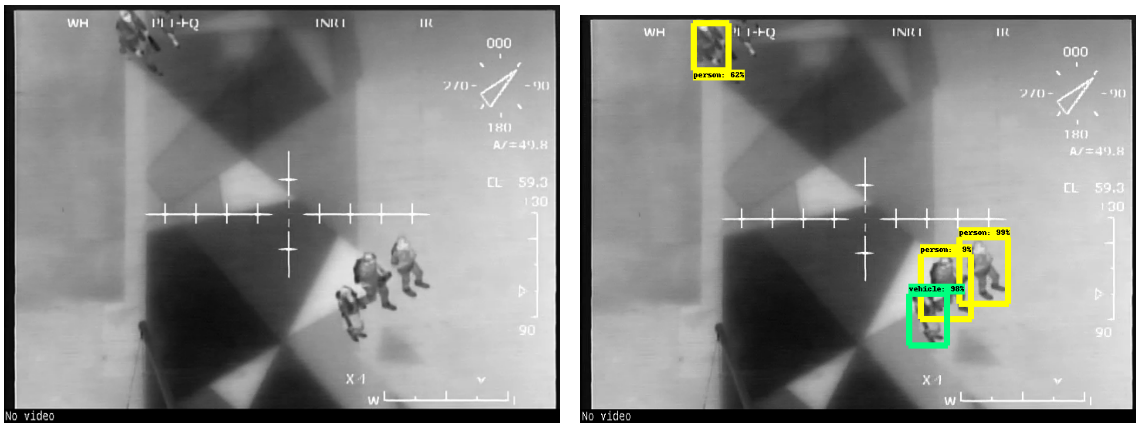


On the left, a *person* correctly detected and a trash bag detect as a *vehicle*. On the right, some *vehicles* correctly detected and two motorcycles detected as a *vehicle*.

SOURCE: Author's production.

Figure 4.17 shows an image where a *person* was labeled as a *vehicle*. As seen in the confusion matrix (Table 4.5), this was the sole example where the network made the mistake of classifying a *person* as a *vehicle*. There aren't mistakes with a *vehicle* detected as *person* in the test dataset. The *person* wrongly detected on this image has its legs close and is carrying something that makes the detection difficult, even for a human being to identify it as a *person* (without considering the overall context).

Figure 4.17 - Mistaken detection of infrared images with a *person* object predicted as a *vehicle*.

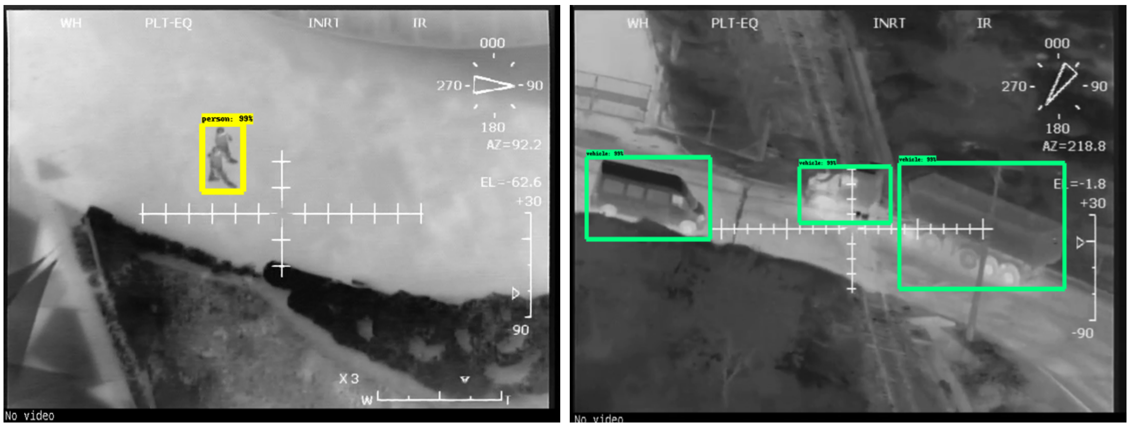


On the left, original image. On the right, some *people* correctly detected but one *person* detected as a *vehicle*.

SOURCE: Author's production.

Figure 4.18 (left) shows two *person* objects detected as a single *person* object. Figure 4.18 (right) shows a *vehicle* (truck) detected as two *vehicles*, the truck front part was detected as one *vehicle* and its back part as another one. This type of error is not a relevant problem for the final application, although they appear in the accuracy measure, because it is much more important to detect the objects than to locate or count them.

Figure 4.18 - Mistaken detection examples of infrared images with ground truths joined as one object or split in two objects.



On the left, two *person* objects joined in just one. On the right, a *vehicle* (truck) split into two objects.

SOURCE: Author's production.

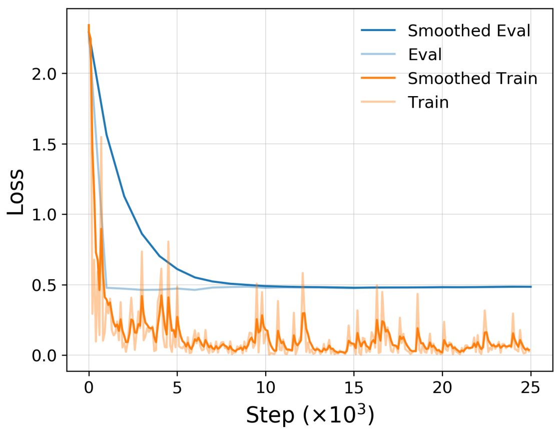
4.2 Visible network

4.2.1 Loss function

A Faster R-CNN model was trained on TensorFlow Object Detection API according to the methodology process explained on section 3.6.1 using only the visible database. The model was trained up to step 25k, which takes 2h and 03min. Figure 4.19 shows the loss for the eval and train datasets during training, where the train loss was taken every 100 steps and the eval loss was taken every 1000 steps. As can be observed in Figure 4.19, the loss rapidly converges as the training starts from a pre-trained model with similar classes and the same spectral bands (RGB images). The oscillatory behavior on the training loss is a consequence of batch size 1.

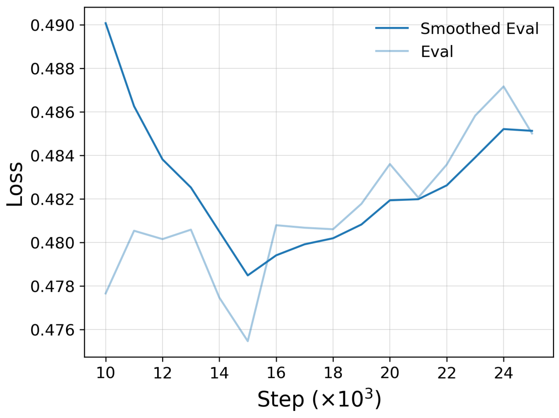
Figure 4.20 also shows the loss, but only for eval values from step 10k up to the end (step 25k) in a larger scale. On this scale it is clearly seen, in the smoothed curve, that the eval loss tends to decrease up to step 15k, so the training on this step was selected to build the final model to prevent overfitting. It should be noted that the values plotted in solid colors are the smoothed curve, while the transparent colors represent the original values. The smoothed curve uses TensorBoard default smooth function, which sets the smooth parameter to 0.6 to reduce the oscillatory behavior, helping the search for the optimal step.

Figure 4.19 - Visible network eval and train dataset losses.



SOURCE: Author's production.

Figure 4.20 - Visible network eval loss in large scale from steps: $10k - 25k$.

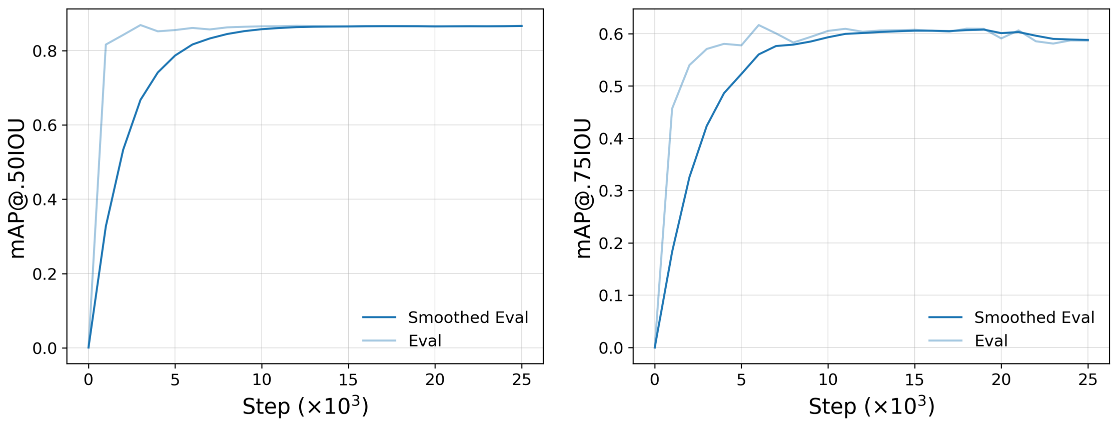


SOURCE: Author's production.

4.2.2 Evaluation metrics

The visible network eval dataset mAP behavior during training is presented on Figure 4.21. The $mAP@.50IOU$ curve stabilizes from step $13k$ up to the end $(25k)$ at mAP 0.865. The $mAP@.75IOU$ stabilizes from step $15k$ up to $19k$ at mAP 0.606, and then decreases. The $mAP@.75IOU$ curve also presents a little peak on step $19k$, which has a mAP of 0.608, so it increases by 0.3%.

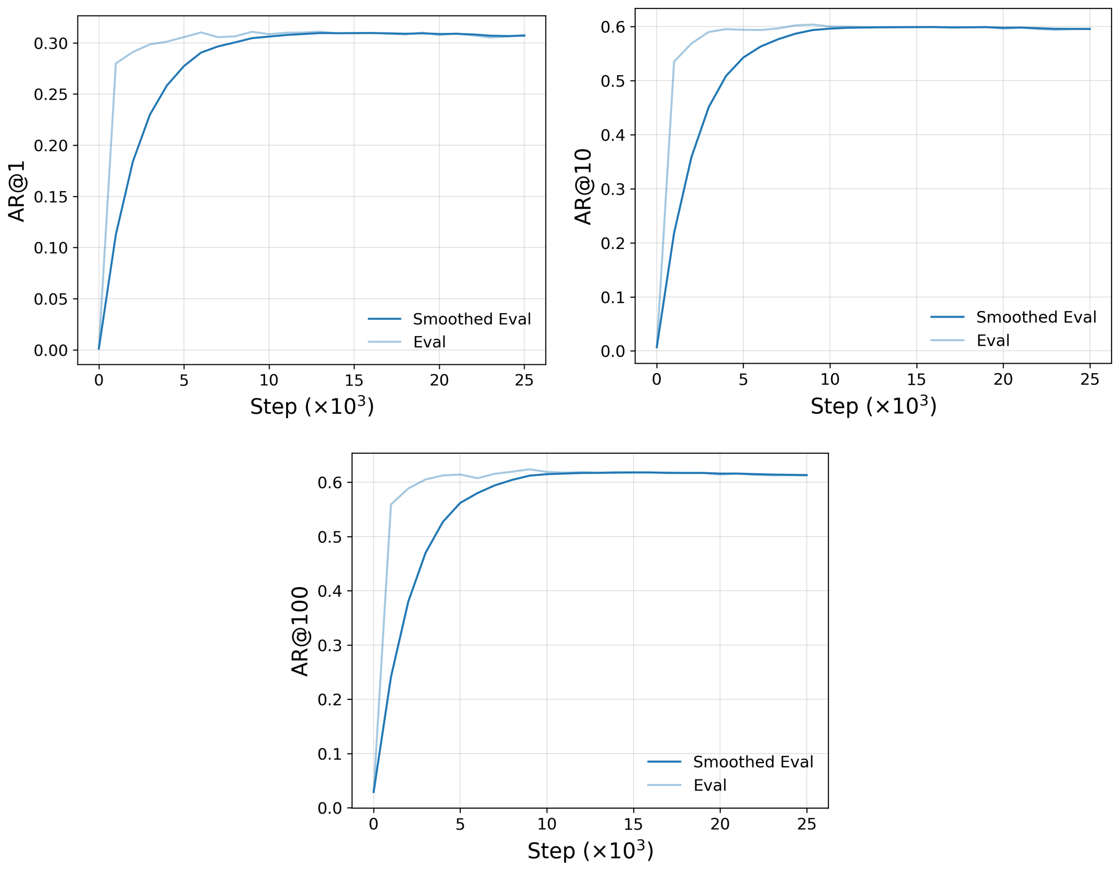
Figure 4.21 - Visible network eval dataset mAP behavior during training.



On the left, $mAP@.50IOU$; on the right, $mAP@.75IOU$.
SOURCE: Author's production.

The visible network eval dataset AR behavior during training is presented on Figure 4.22. The $AR@1$ curve stabilizes from step $13k$ up to $16k$ around AR 0.310, and then begins to decrease up to the end $(25k)$. The $AR@10$ curve presents a similar behavior, stabilizing from step $13k$ up to $19k$ around AR 0.599, and then has a small reduction up the end $(25k)$. The $AR@100$ also presents a similar behavior, stabilizing from step $12k$ up to $17k$ around AR 0.618, and then the curve starts to decrease.

Figure 4.22 - Visible network eval dataset AR behavior during training.



Top left, AR@1; top right, AR@10; bottom center, AR@100.

SOURCE: Author's production.

All these curves present the expected behavior, starting very small, tending to stabilize during training and tend to decrease afterwards. This decrease means that the overfitting is making the network so specialized on the training data, that it loses its ability to generalize. Their analysis show that any step between $13k$ and $19k$ represent suitable trained models. Therefore, the choice of the model trained up to step $15k$, based on the loss curve and explained on Section 4.2.1, is in agreement with the mAP and AR curves.

As previously explained, the train on step $15k$ was chosen to build the final visible network, thus it was exported to a graph. The results achieved by this graph on the visible datasets are shown on Tables 4.8 through 4.14. As expected, Tables 4.8 and 4.9 show that the visible train dataset metrics are better than the others, as the model was fitted to it. The test dataset performed better than the eval dataset (by a short margin) what is not common but is possible, given that the dataset was randomly generated. Both mAP metrics reached high performance, being $mAP@.50IOU$ metric the highest one, as expected, because it has a lower IoU. It reaches 0.933 mAP, which is very close to the maximum value (1.0). Therefore, the $mAP@.50IOU$ demonstrates that the model got an excellent performance on object detection.

Table 4.8 - Visible database mAP metrics.

	mAP@.50IOU	**mAP@.75IOU**
Train dataset	0.990	0.968
Eval dataset	0.865	0.606
Test dataset	0.933	0.692

SOURCE: Author's production.

The AR metrics reached by the model, Table 4.9, indicate that the model predictions were accurate on classification and localization. The AR@100 metric was the highest AR, reaching 0.678 AR.

Table 4.9 - Visible database AR metrics.

	AR@1	**AR@10**	**AR@100**
Train dataset	0.353	0.855	0.895
Eval dataset	0.310	0.599	0.618
Test dataset	0.317	0.658	0.678

SOURCE: Author's production.

Tables 4.10, 4.11 and 4.12 contain the confusion matrix (considering T_{CS}=0.50 and T_{IOU}=0.50) obtained on the visible datasets for the background and classes: *vehicle* and *person*. The train dataset presented 95.8% of accuracy, which is expected because the model learning was based on it. The eval and test datasets reach, respectively, 81.6% and 86.1%, of accuracy meaning that the model presented a high performance. Normally, the eval dataset present a better result than the test one, but the opposite is also possible.

Table 4.10 - Visible train dataset confusion matrix for background and classes: *vehicle* and *person*.

		Predicted		
		Vehicle	Person	Background
Ground truth	Vehicle	664	2	9
	Person	0	687	2
	Background	30	16	—

SOURCE: Author's production.

Table 4.11 - Visible eval dataset confusion matrix for background and classes: *vehicle* and *person*.

		Predicted		
		Vehicle	Person	Background
Ground truth	Vehicle	194	2	16
	Person	3	68	10
	Background	16	12	—

SOURCE: Author's production.

The test dataset confusion matrix shows that the model correctly detected most of the *vehicle* and *person* objects, only failing on 14 *vehicles* and 10 *people*. Furthermore, 21 objects were incorrectly detected as *vehicle* and 22 as *person*. Still, the model was able to achieve 86.1% accuracy, which proves its high performance.

Table 4.12 - Visible test dataset confusion matrix for background and classes: *vehicle* and *person*.

		Predicted		
		Vehicle	Person	Background
Ground truth	Vehicle	280	0	14
	Person	1	136	9
	Background	21	22	—

SOURCE: Author's production.

Tables 4.13 and 4.14 also contain the test dataset confusion matrix, but for each class separately. In these matrices, the predictions that were wrong about the class are considered background. These matrices provide a clearer view of each class performance. So, the model achieves accuracy of 88.9% for *vehicle* and 81.0% for *person* class. This result proves that the model present high accuracy on both classes, being better for detecting *vehicle* than *person*. This happens because people seen from a long distance on RGB images aren't as clearly defined as vehicles.

Table 4.13 - Visible test dataset confusion matrix for background and *vehicle* class.

		Predicted	
		Vehicle	Background
Ground truth	Vehicle	280	14
	Background	21	—

SOURCE: Author's production.

Table 4.14 - Visible test dataset confusion matrix for background and *person* class.

		Predicted	
		Person	Background
Ground truth	Person	136	10
	Background	22	—

SOURCE: Author's production.

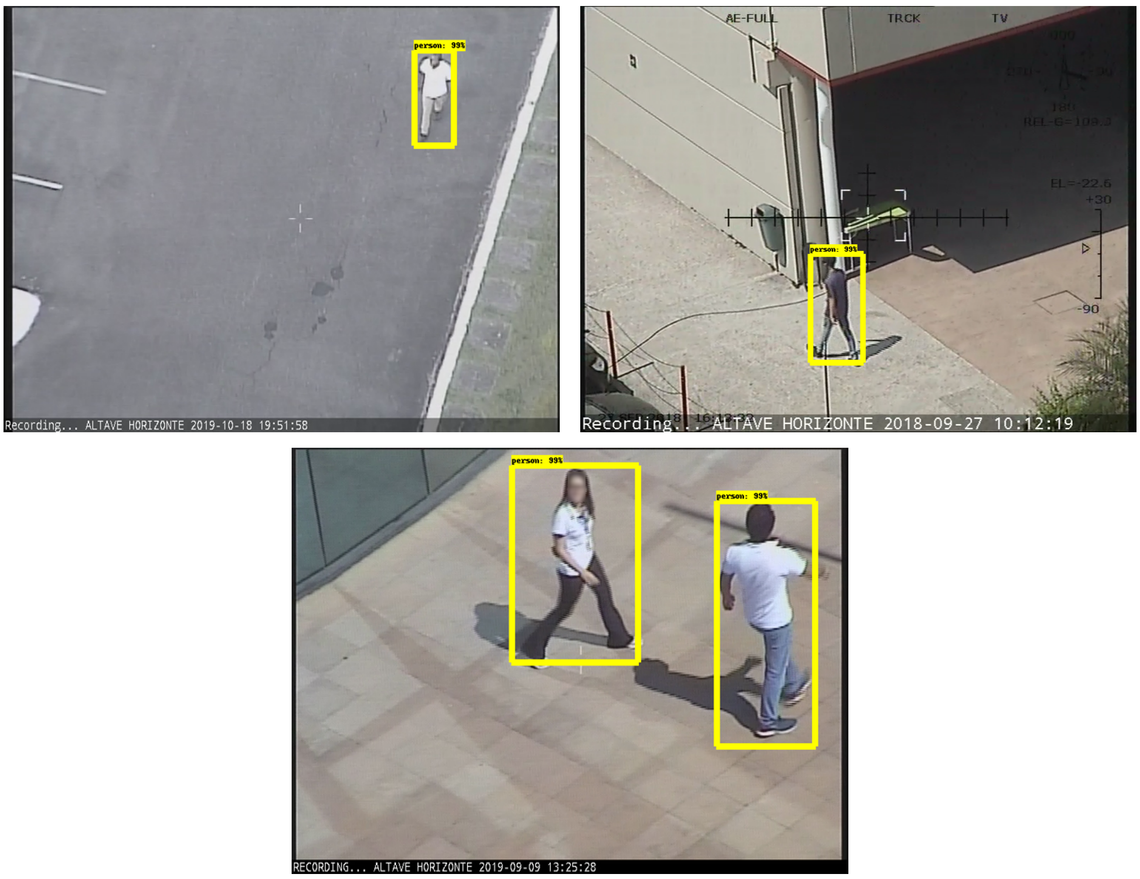
4.2.3 Images

In order to provide a visualization of the results achieved by the visible network built, this section presents some examples of images from the visible test dataset with their respective predicted objects. The images showed in this chapter are from the test dataset, same used on the confusion matrix. Therefore, for all images analysis T_{CS}=0.50 and T_{IOU}=0.50 were used to define if an object was correctly detected or not. The images are presented in two sections: Section 4.2.3.1 only contains images in which all objects were correctly detected and Section 4.2.3.2 contains images with at least one mistake. In all images, yellow and green boxes represent, respectively, *person* and *vehicle* objects predicted by the built network. Blue and purple boxes, in turn, represent the ground truth of *person* and *vehicle* objects not predicted by the network, respectively. It is worth mentioning that just the ground truth that helped the visualization were drawn on the images.

4.2.3.1 Correct detections

Figure 4.23 presents some examples of images with *person* objects in a variety of angles of view and position. All these objects correctly predicted demonstrate the high performance achieved by the network on detecting *person* objects.

Figure 4.23 - Correct detection examples of visible images with *person*.

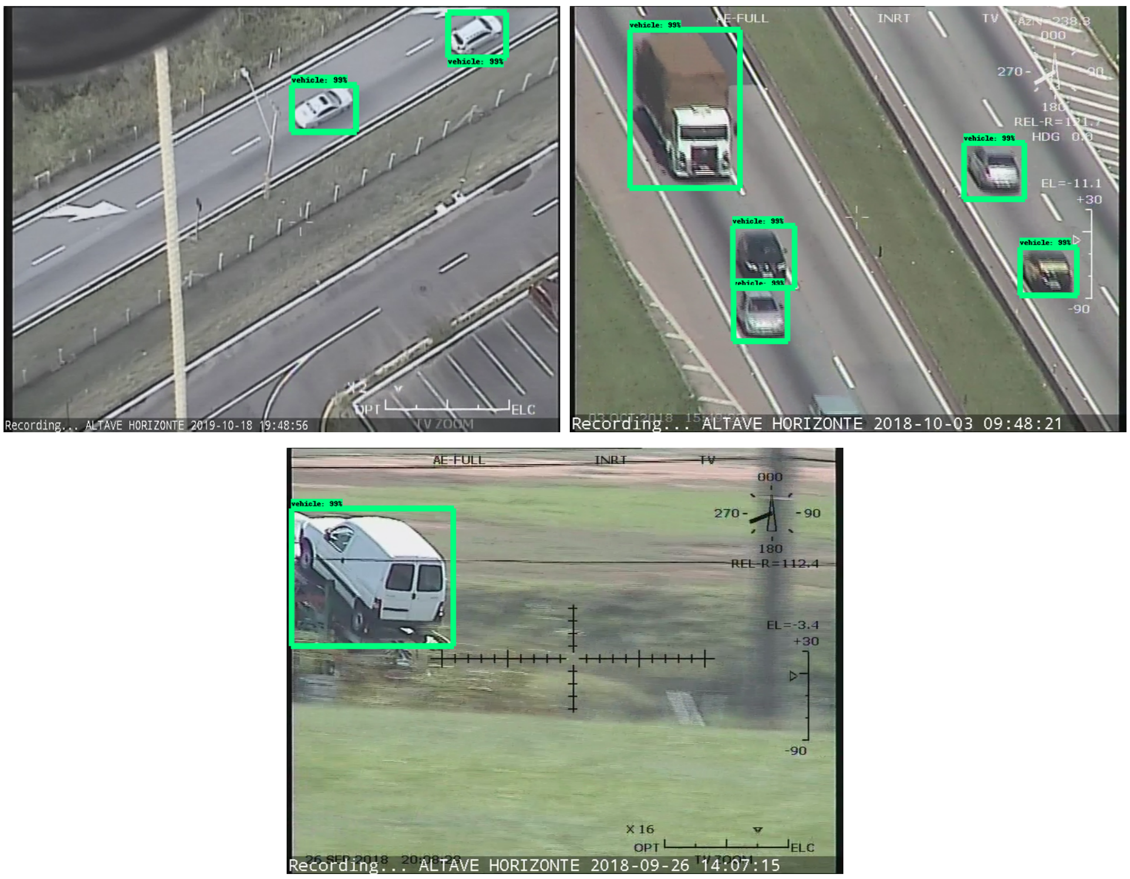

person: 99%
Recording... ALTAVE HORIZONTE 2019-10-18 19:51:58
AE-FULL
TRCK
TV
EL=-22.6
+30
-90
person: 99%
Recording... ALTAVE HORIZONTE 2018-09-27 10:12:19
person: 99%
person: 99%
RECORDING... ALTAVE HORIZONTE 2019-09-09 13:25:28

For privacy's sake, faces were blurred, just for visualization.

SOURCE: Author's production.

Figure 4.24 presents some examples of images with *vehicle* objects in a variety of angles of view and position. All these objects correctly predicted demonstrates the high performance achieved by the network on detecting *vehicle* objects.

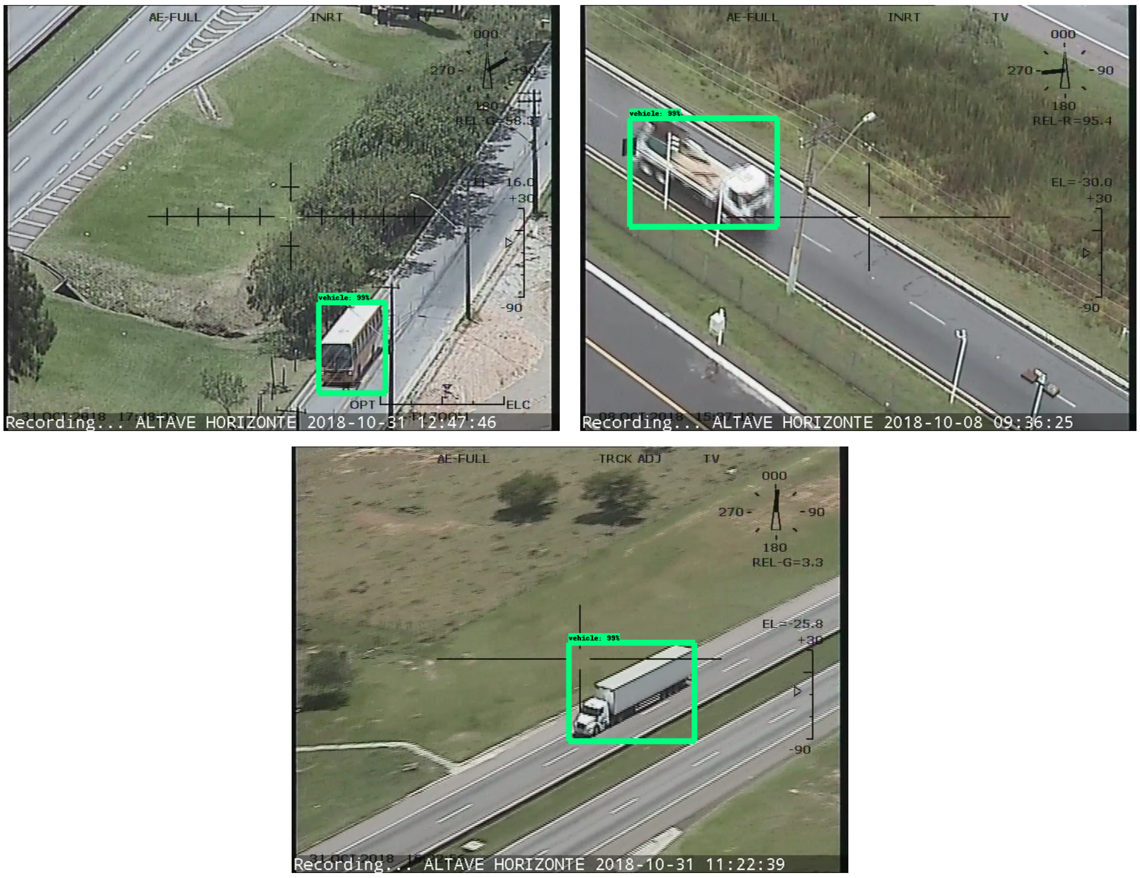
Figure 4.24 - Correct detection examples of visible images with *vehicle*.


vehicle: 99%
Recording... ALTAVE HORIZONTE 2019-10-18 19:48:56
Recording... ALTAVE HORIZONTE 2018-10-03 09:48:21
Recording... ALTAVE HORIZONTE 2018-09-26 14:07:15

SOURCE: Author's production.

Figure 4.24 and 4.25 present some examples of images with objects from *vehicle* like bus and trucks in distinct points of view correctly detected. This demonstrates the high performance achieved by the network on detecting a variety of types of *vehicles*, with distinct shapes and points of view.

Figure 4.25 - Correct detection examples of visible images with *vehicles*, such as truck and bus.

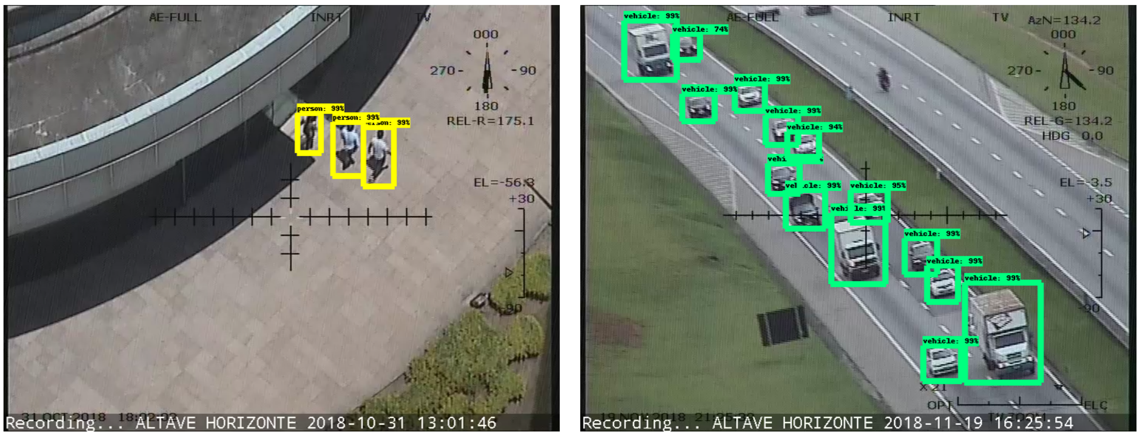

AE-FULL
INRT
TV
vehicle: 99%
Recording... ALTAVE HORIZONTE 2018-10-31 12:47:46
Recording... ALTAVE HORIZONTE 2018-10-08 09:36:25
TRCK ADJ
REL-G=3.3
EL=-25.8
Recording... ALTAVE HORIZONTE 2018-10-31 11:22:39

Object from *vehicle* class like bus (top left) and trucks (top right and bottom center).

SOURCE: Author's production.

Figure 4.26 presents some examples of images that contain multiples objects. This Figure demonstrates that the network is able to detect multiple objects in an image, including overlapped on objects boxes. The network can deal with overlapping on bounding boxes, as long as the objects are not hiding a significant part of the other.

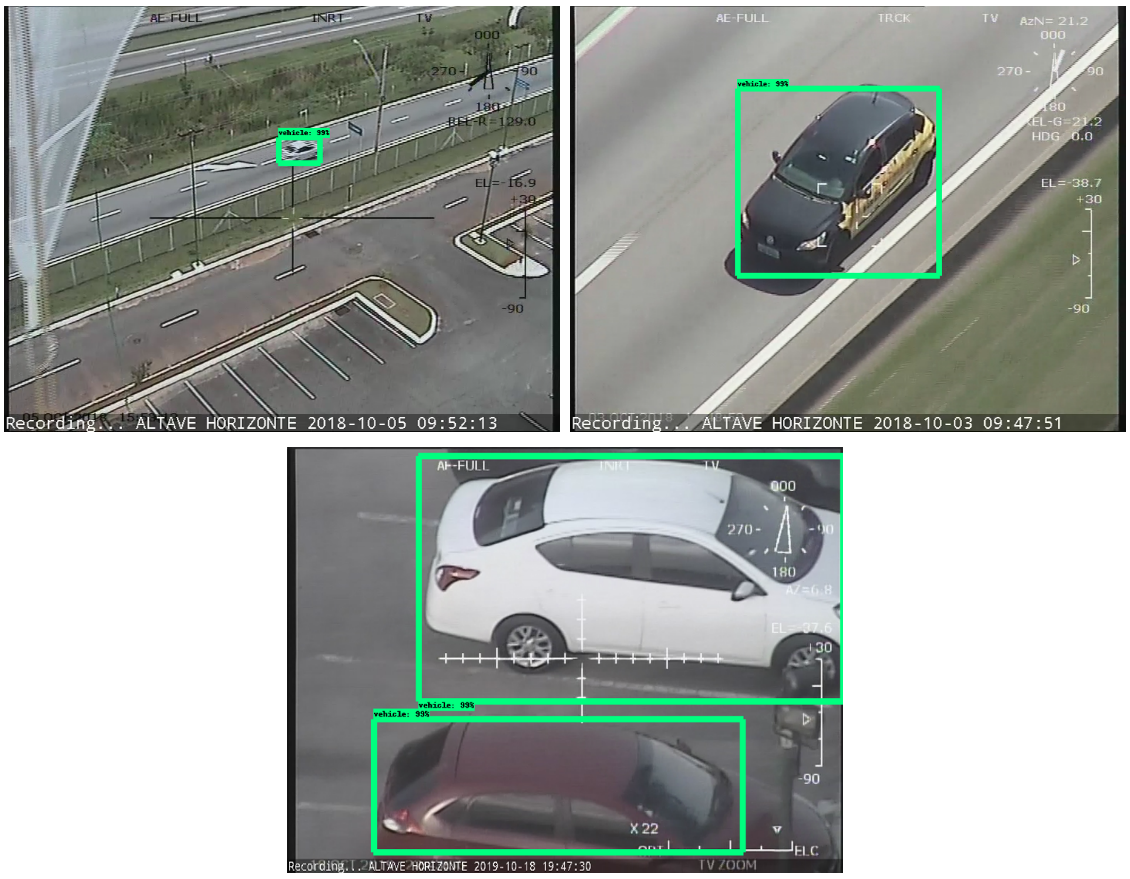
Figure 4.26 - Correct detection examples of visible images with multiple objects.


person: 99%
vehicle: 99%
Recording... ALTAVE HORIZONTE 2018-10-31 13:01:46
Recording... ALTAVE HORIZONTE 2018-11-19 16:25:54

SOURCE: Author's production.

Figure 4.27 presents some examples of images that contain *vehicles* in distinct sizes. This Figure proves that, as intended, the network is able to detect objects in a variety of sizes, from small (left) to medium (right) and even the larger (bottom) ones. The wide range of anchor sizes chosen in the model configuration, able to cover almost all objects sizes, directly impacts on these positives results.

Figure 4.27 - Correct detection examples of visible images with very distinct *vehicle* scales.

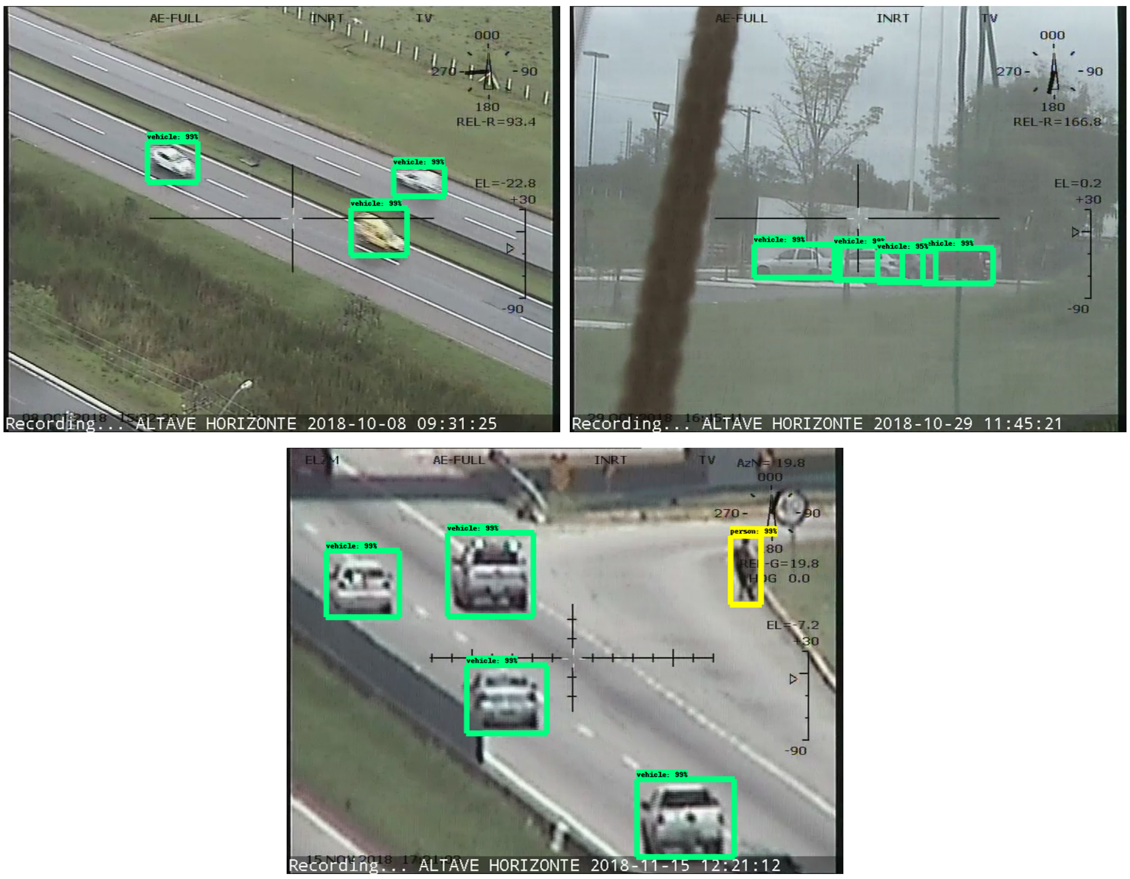

vehicle: 99%
Recording... ALTAVE HORIZONTE 2018-10-05 09:52:13
vehicle: 99%
Recording... ALTAVE HORIZONTE 2018-10-03 09:47:51
vehicle: 99%
vehicle: 99%
Recording... ALTAVE HORIZONTE 2019-10-18 19:47:30

Vehicles on a variety of sizes: small (top left), medium (top right) and large (bottom center).

SOURCE: Author's production.

Figure 4.28 presents some examples of images with some noisy objects. Therefore, this Figure evidences that, as intended, the network is able to detect objects with some noise, like the ones caused by the smooth camera movements, the balloon fabric overlap and fog on the environment.

Figure 4.28 - Correct detection examples of visible images with noisy.

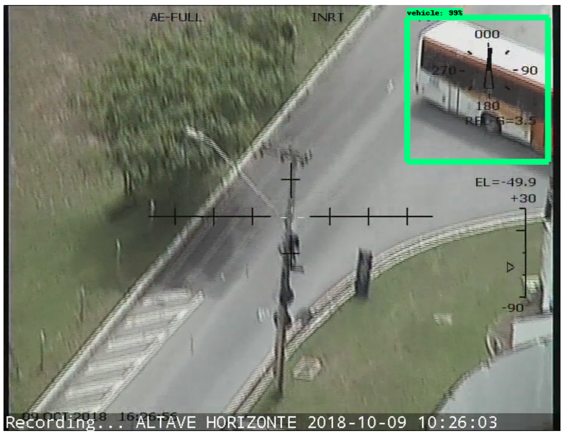

vehicle: 99%
person: 99%
Recording... ALTAVE HORIZONTE 2018-10-08 09:31:25
Recording... ALTAVE HORIZONTE 2018-10-29 11:45:21
Recording... ALTAVE HORIZONTE 2018-11-15 12:21:12

Vehicles with some noise caused by: the smooth camera movements (top left), the balloon fabric overlap (top right) and fog on the environment (bottom center).

SOURCE: Author's production.

Figure 4.29 demonstrates the network ability to detect partially hidden objects, like the *vehicle* (bus) in this image, which was partially outside of the camera field of view. Partially hidden objects can be correctly detected by the network, as long as their main features can still be seen. After all, the network does not maintain any information from previous images.

Figure 4.29 - Correct detection example of visible images with partially hidden *vehicle*.

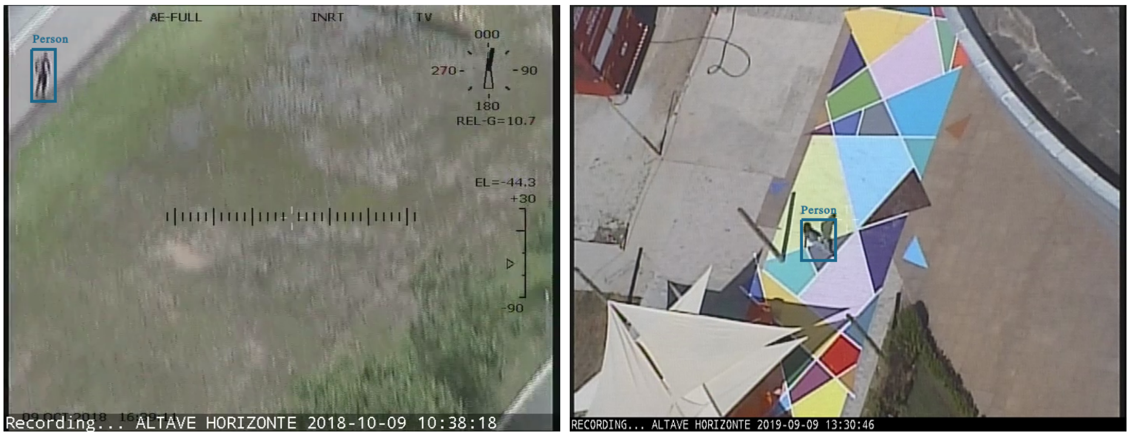


SOURCE: Author's production.

4.2.3.2 Mistakes

Figure 4.30 shows *person* objects present in the ground truth that the network could not detect. The *person* objects in both images were very small and in positions that should be difficult even for a human to recognize. The *person* on Figure 4.30 (right) was walking on a colored ground that did not contrast with its clothes, making it less apparent in this context. Complex background, like this one, compromises the quality of network predictions, influencing this type of mistakes.

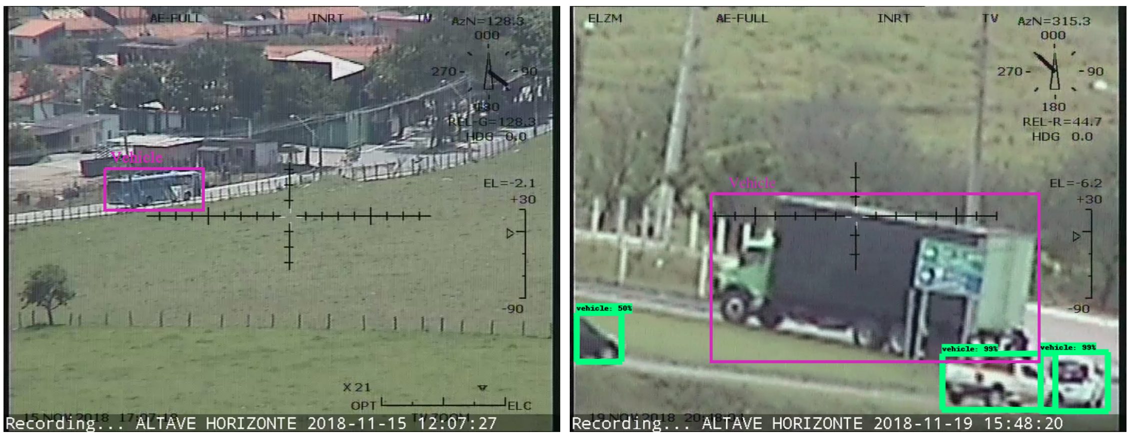
Figure 4.30 - Mistaken detection examples of visible images with undetected *person*.


Person
AE-FULL
INRT
TV
000
270
-90
180
REL-G=10.7
EL=-44.3
+30
-90
Recording... ALTAVE HORIZONTE 2018-10-09 10:38:18
Person
RECORDING... ALTAVE HORIZONTE 2019-09-09 13:30:46

A *person* not detected (in blue) on each image.
SOURCE: Author's production.

Figure 4.31 shows *vehicle* objects present in the ground truth that the network wasn't capable to detect. The *vehicle* on Figure 4.31 (left) is a bus seen in a small scale due to the distance, and there is a small fence and a light pole on in front of it, characteristics that make the *vehicle* harder to detect. The *vehicle* on Figure 4.31 (right) presents a blur noise that the network has difficulties to capture the main features, leading to a mistake. These images show that, even though the network is able to detect objects with noise, it has some limitations on dealing with it.

Figure 4.31 - Mistaken detection examples of visible images with undetected *vehicle*.

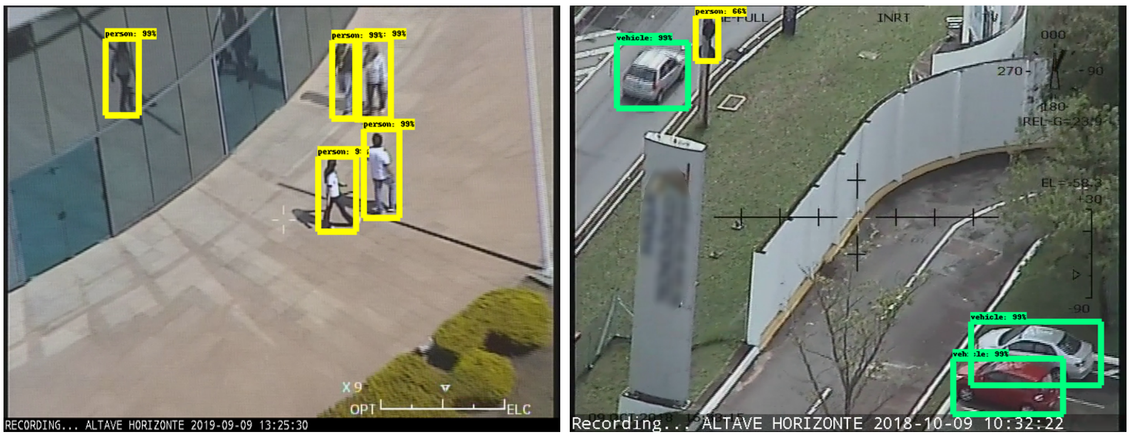

Vehicle
Recording... ALTAVE HORIZONTE 2018-11-15 12:07:27
vehicle: 50%
vehicle: 99%
vehicle: 99%
Vehicle
Recording... ALTAVE HORIZONTE 2018-11-19 15:48:20

On the left, a *vehicle* (bus) not detected (in purple). On the right, three *vehicle* correctly detected and a *vehicle* (truck), in purple, not detected.

SOURCE: Author's production.

Figure 4.32 shows some incorrect predictions, on which part of the background was detected as *person* objects. The person reflection in the glass and the traffic sign were wrongly detected as *person* objects. They present some features that make the network believe they are of the *person* class. The reflection on the glass, Figure 4.32 (left), is very similar to a real *person*, thus it is very difficult for the network to distinguish it from the real one. The traffic sign, in turn, has some features that resembles head and arms that could lead to the mistake. To avoid this kind of mistakes, more images with objects like *person* objects and others that resemble people are essential to provide the network enough information to enable it to learn the features that only belong to *person*.

Figure 4.32 - Mistaken detection examples of visible images with background detected as *person*.

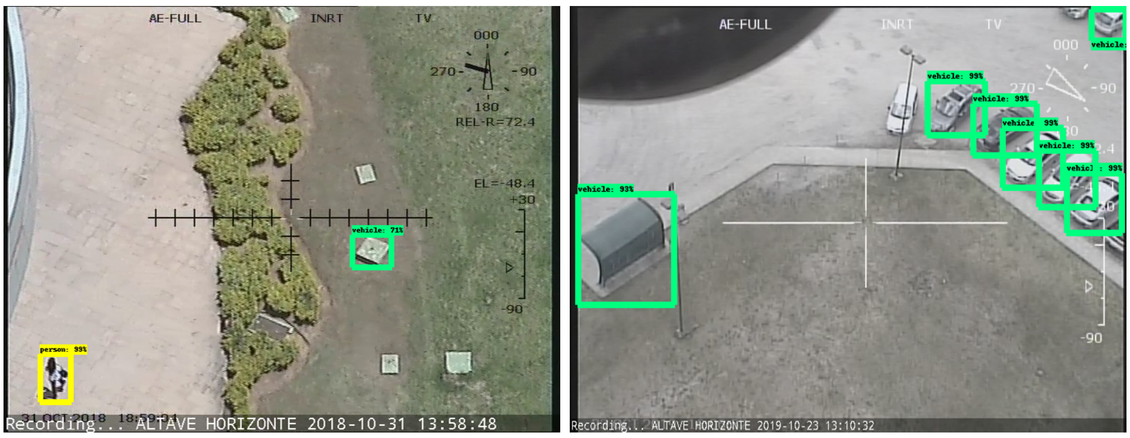

person: 99%
person: 99%
person: 99%
person: 99%
RECORDING... ALTAVE HORIZONTE 2019-09-09 13:25:30
person: 66%
vehicle: 99%
vehicle: 99%
Recording... ALTAVE HORIZONTE 2018-10-09 10:32:22

On the left, four *person* objects correctly detected and a person reflection in the glass wrongly detected as a *person*. On the right, three *vehicles* correctly detected and a traffic sign detected as a *person*. Logos were blurred, just for visualization.

SOURCE: Author's production.

Figure 4.33 shows some incorrect predictions where parts of the background were detected as *vehicles*. The box and bus stop, wrongly detected as *vehicles*, present some features that make the network believe they are *vehicles*. The bus stop on Figure 4.33 (right) is similar to a semi-trailer and could lead to the mistake. To avoid this kind of mistakes, more images with *vehicles* objects and others that resemble *vehicle* are essential to provide the network enough information to enable it to learn the features that only belong to *vehicle*.

Figure 4.33 - Mistaken detection examples of visible images with background detected as *vehicle*.



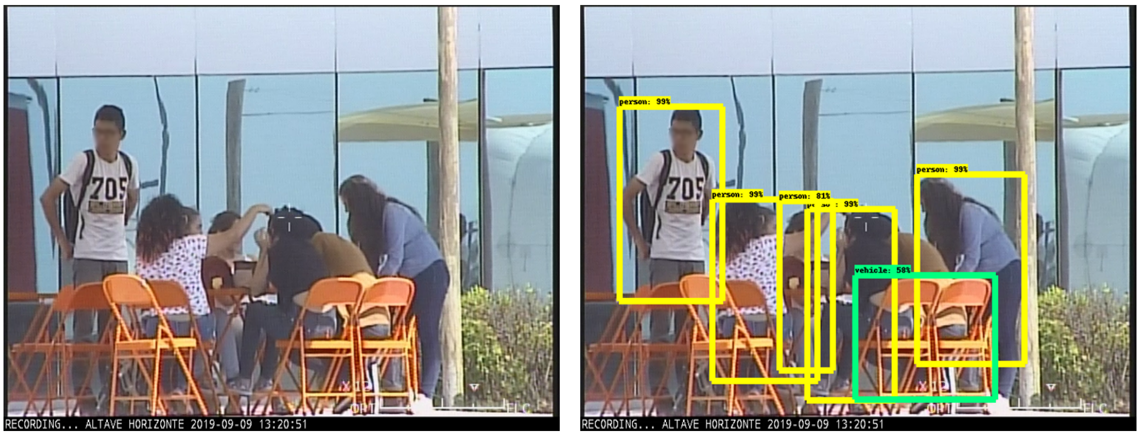
AE-FULL
INRT
TV
vehicle: 71%
person: 99%
Recording... ALTAVE HORIZONTE 2018-10-31 13:58:48
vehicle: 93%
vehicle: 99%
Recording... ALTAVE HORIZONTE 2019-10-23 13:10:32

On the left, a *person* correctly detected and a box detect as a *vehicle*. On the right, some *vehicles* correctly detected, a *vehicle* not detected and a bus stop detected as a *vehicle*.

SOURCE: Author's production.

Figure 4.34 shows the image in which a *person* was predicted as a *vehicle*. As shown in the confusion matrix (Table 4.12), this was the sole example where the network makes the mistake of detecting a *person* as a *vehicle* - there are no mistakes with a *vehicle* detected as *person* in the test dataset. This is a very complex scene, being difficult even for humans to perfectly locate the objects box boundaries. The background detected as a *person* actually had combination of arms, legs and head from three people. The *person* predicted as *vehicle*, in turn, actually comprises of two chairs with people seated on them. Therefore, although according to the IoU criterion it was considered a *person* predicted as *vehicle*, by visual inspection it should be considered a background detected as *vehicle*.

Figure 4.34 - Mistaken detection of visible images with a *person* object predicted as a *vehicle*.

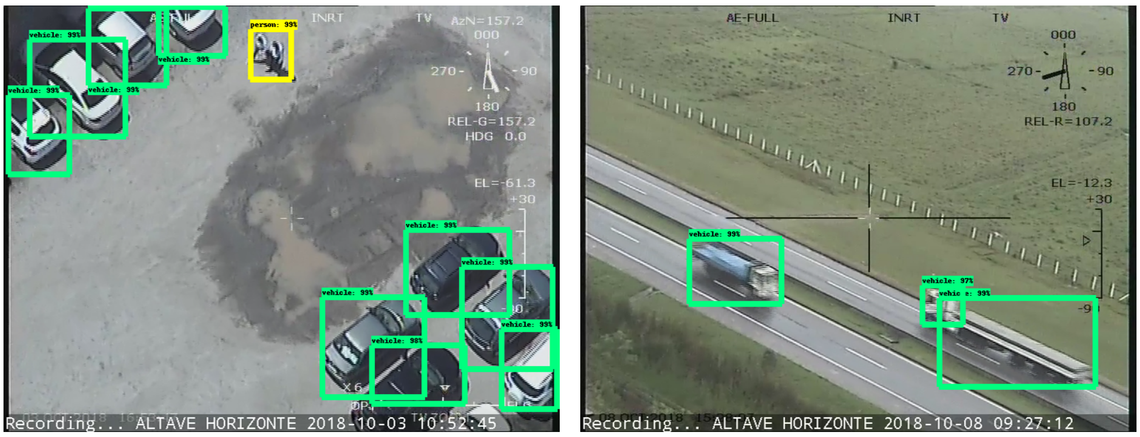

RECORDING... ALTAVE HORIZONTE 2019-09-09 13:20:51
person: 99%
person: 99%
person: 81%
person: 99%
vehicle: 58%
RECORDING... ALTAVE HORIZONTE 2019-09-09 13:20:51

On the left, original image. On the right, four *people* correctly detected, one background deteced as a *person* and two chairs detected as a *vehicle* that had IoU> 0.50 with a *person* ground truth, therefore they account as a *person* wrongly detected as a *vehicle*. For privacy's sake, faces were blurred, , just for visualization.

SOURCE: Author's production.

Figure 4.35 (left) shows two *people* detected as just one, so they were joined in a single *person* object. Figure 4.35 (right) shows a *vehicle* (truck) detected as two *vehicles*, the truck front part was detected as one *vehicle* and its semi-trailer as another one. This type of error is not a problem for the final application, although they appear in the accuracy calculation, because it is much more important to detect the objects than to locate or count them.

Figure 4.35 - Mistaken detection examples of visible images with ground truths joined as one object or split into two objects.



On the left, two *person* objects joined in just one and some *vehicles* correctly detected. On the right, a *vehicle* (truck) split into two *vehicles* and a *vehicle*(truck) correctly detected.

SOURCE: Author's production.

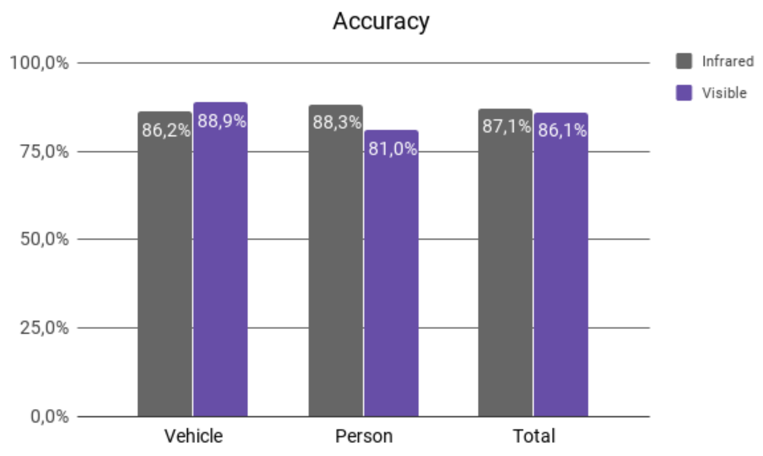
4.3 Comparison

The comparison between these networks shows that the infrared one presents a better performance on *person* class, while the visible fares better on *vehicle* class, as seen in Figure 4.36. The better performance on *person* objects in infrared happens because *person* contrasts with the background on infrared images due to the high natural temperature of the body, while in visible remote sensing images they may have low contrast with the background, specially if the background colors match the clothes colors. The marginally higher performance on *vehicle* objects in visible probably occurs because *vehicles* are, in general, on very distinct surfaces (*e.g.*: asphalt, grass or dirty roads), with different colors compared to the car.

The overall (total) accuracy of both networks were similar, being the difference of only 1.0%. A small difference like that could vary by a simple addition of new images on the train and test datasets or even by a new random split in the building of the database. Therefore, the accuracy difference wasn't big enough to conclude which network would perform better on the final application.

In general, both the infrared and visible networks achieved high performance on their respective test dataset. The networks take about 0.2 seconds to detected objects on a new image. Thus, they can run at about 5 frames per second, which is enough for real-time monitoring of large areas to avoid possible threats.

Figure 4.36 - Comparison of the test dataset accuracy between the infrared and visible networks.



SOURCE: Author’s production.

5 CONCLUSIONS

This work developed a system based on DL capable of detecting people and vehicles (possible threats) with remote sensing images from infrared and visible sensors arranged in captive balloons. The purpose of this system was to help humans make site surveillance by continuously monitoring large courtyard areas. This work used videos provided by Altave Company, which are obtained from their captive balloon system. The image acquisition hardware can only provide videos from one sensor each time.

The DL technique used for object detectors was based on CNN, more specifically Faster R-CNN. The strategy used in this work was to train two networks, one for each sensor (infrared and visible). Hence, two databases containing about 700 images each were manually built. The Faster R-CNN meta-architecture, implemented on TensorFlow API, was used to build the object detector network. Training large CNN from scratch requires large image databases and, consequently, demands a large computational load. Due to these reasons, the two networks were fine-tunned with a pre-trained network, trained over ordinary RGB COCO images.

After proper fine-tuning of the two networks built, they were able to detect people and vehicles on images from Altave captive balloons in *quasi* real time. The accuracy, mAP, and AR metrics results obtained from the built networks show the high performance of the model. The total accuracy was 87.1% for the infrared network and 86.1% for the visible. These accuracies were obtained only by the usage of independent frames, thus they could be improved by taking advantage of the sequence of frames. These high accuracies demonstrated that the Faster R-CNN pre-trained only in ordinary RGB images can be fine-tuned to work satisfactorily on 3-band RGB visible remote sensing images and even on 1-band infrared images, as long as they are properly converted for 3-band grayscale images by repeating the infrared band throughout the three channels.

The predictions of some test images demonstrated the networks capabilities and limitations. The networks trained were able to detect multiple *person* and *vehicle* objects in captive balloon images with a variety of angles, positions, types (for vehicles), and scales. The networks were also able to deal with some noise and overlap on the objects. The infrared network was also able to detect objects on both modes, white hot and black hot. Some images presented mistakes generated by splitting parts of one object into two objects or merging two objects from the same class in one large object. This type of mistake is not a relevant problem for surveillance because it is much more important to detect the objects than to locate or count them, however they were taken into account for the evaluation metrics.

5.1 Future works

In future works, the database for both sensors should be increased to at least 2000 images per class. By increasing the database images, the network should provide a better generalization, after learning the features that only belong to *person* and *vehicle* objects. At this point, the technique developed only works on images, and a future work should improve the technique developed to run directly on videos. On videos, the accuracy could be improved by, for instance, analyzing 5 frames and making a verification of continuity of the detections.

Another future work should be to fine-tune the same pre-trained network using both databases in order to compare it with the networks fine-tuned on only one of those databases. This comparison would demonstrate if a single network would have the capability to work satisfactorily with both, visible and infrared, data.

REFERENCES

ABADI, M.; BARHAM, P.; CHEN, J.; CHEN, Z.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; IRVING, G.; ISARD, M.; KUDLUR, M.; LEVENBERG, J.; MONGA, R.; MOORE, S.; MURRAY, D.; STEINER, B.; TUCKER, P.; VASUDEVAN, V.; WARDEN, P.; WICKE, M.; YU, Y.; ZHENG, X. Tensorflow: a system for large-scale machine learning. In: USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 12., 2016, Savannah, GA. **Proceedings...** Savannah, 2016. v. 16, p. 265–283. 33

ALTAVE. **Monitoramento and conectividade inteligente: a solução com melhor custo-benefício para cobertura de grandes áreas**. 2019. Available from: <http://www.altave.com.br/>. Access in: 2 July 2019. 2, 31, 32

AMMOUR, N.; ALHICHRI, H.; BAZI, Y.; BENJDIRA, B.; ALAJLAN, N.; ZUAIR, M. Deep learning approach for car detection in uav imagery. **Remote Sensing**, v. 9, n. 4, p. 312–327, 2017. 29

AZEVEDO, B. A. de. **Design, trimming, stabilization and flight testing of tethered aerostats**. 2016. 211 p. Thesis (Doctoral in Aeronautical and Mechanical Engineering) — Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, 2016. 1, 3

AZEVEDO, B. de; CUNHA, M.; MORALES, M.; GOES, L. Stability and active control of low altitude aerostats. In: AIAA LIGHTER-THAN-AIR SYSTEMS TECHNOLOGY (LTA) CONFERENCE, 20., 2013. **Proceedings...** Daytona Beach, 2013. p. 1299. 1

AZEVEDO, F. A.; CARVALHO, L. R.; GRINBERG, L. T.; FARFEL, J. M.; FERRETTI, R. E.; LEITE, R. E.; JACOB FILHO, W.; LENT, R.; HERCULANO-HOUZEL, S. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. **Journal of Comparative Neurology**, v. 513, n. 5, p. 532–541, 2009. 7

BENDINI, H. N.; MORAES, W. S.; COSTA, S.; LOPES, E. S. S.; KORTING, T. S.; FONSECA, L. M. G. Proposta de sistema de monitoramento da Sigatoka-Negra baseado em variáveis ambientais utilizando o TerraMA2. In: BRAZILIAN SYMPOSIUM ON GEOINFORMATICS, 15., 2014, Campos do Jordão, Brazil. **Proceedings...** São José dos Campos: MCTIC/INPE , 2014. p. 168–173. 1

BIEDERMAN, I. Recognition-by-components: a theory of human image understanding. **Psychological Review**, v. 94, n. 2, p. 115, 1987. 2

CHAGAS, C. S.; VIEIRA, C. A.; FERNANDES FILHO, E.; CARVALHO JUNIOR, W. Utilização de redes neurais artificiais na classificação de níveis de degradação em pastagens. **Revista Brasileira de Engenharia Agrícola e Ambiental**, v. 13, n. 3, p. 319–327, 2009. 7

CHEN, Y.; LIN, Z.; ZHAO, X.; WANG, G.; GU, Y. Deep learning-based classification of hyperspectral data. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 7, n. 6, p. 2094–2107, 2014. 18

COMMON OBJECTS IN CONTEXT - COCO. **COCO API**. 2018. Available from: <https://github.com/cocodataset/cocoapi>. Access in: 11 Jan. 2020. 38

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals and Systems**, v. 2, n. 4, p. 303–314, 1989. 11

DAI, J.; LI, Y.; HE, K.; SUN, J. R-FCN: Object detection via region-based fully convolutional networks. **Advances in Neural Information Processing Systems**, p. 379–387, 2016. 33

DAT TRAN. **Raccoon detector dataset**. 2018. Available from: <https://github.com/datitran/raccoon_dataset>. Access in: 05 Dec. 2019. 36

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: a large-scale hierarchical image database. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 9., 2009. **Proceedings...** Miami Beach, 2009. 18, 24

EVERINGHAM, M.; GOOL, L. V.; WILLIAMS, C. K.; WINN, J.; ZISSERMAN, A. The Pascal Visual Object Classes (VOC) challenge. **International Journal of Computer Vision**, v. 88, n. 2, p. 303–338, 2010. 24, 38

GEIGER, A.; LENZ, P.; STILLER, C.; URTASUN, R. Vision meets robotics: the KITTI dataset. **International Journal of Robotics Research (IJRR)**, v. 32, n. 11, p. 1231–1237, 2013. 46

GIRSHICK, R. Fast R-CNN. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV), 2015, Santiago, Chile. **Proceedings...** Santiago, 2015. p. 1440–1448. 24, 29

GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 27., 2014. **Proceedings...** Columbus, 2014. p. 580–587. 24

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. 2. ed. Cambridge, MA: MIT Press, 2017. ISBN 9780262035613. 9, 10, 12, 14, 19, 21, 22, 23

GUILD, L. S.; COHEN, W. B.; KAUFFMAN, J. B. Detection of deforestation and land conversion in Rondonia, Brazil using change detection techniques. **International Journal of Remote Sensing**, v. 25, n. 4, p. 731–750, 2004. 1

HAUSAMANN, D.; ZIRNIG, W.; SCHREIER, G.; STROBL, P. Monitoring of gas pipelines–a civil UAV application. **Aircraft Engineering and Aerospace Technology**, v. 77, n. 5, p. 352–360, 2005. 1, 2

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 29., 2016, Las Vegas Valley, United States of America. **Proceedings...** Las Vegas Valley: IEEE, 2016. p. 770–778. 24, 26

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: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2017. **Proceedings...** Honolulu, 2017. p. 7310–7311. 33, 34, 44, 50

ISAKSSON, J. **Recognizing microscopic structures: dense semantic segmentation of multiple histopathological classes using fully convolutional neural networks**. 2016. 52 p. Thesis (Master in Mathematical Sciences) — Lund University, Faculty of Engineering, Sweden, 2016. 10

JHA, M.; LEVY, J.; GAO, Y. Advances in remote sensing for oil spill disaster management: state-of-the-art sensors technology for oil spill surveillance. **Sensors**, v. 8, n. 1, p. 236–255, 2008. 1

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Advances in Neural Information Processing Systems**, p. 1097–1105, 2012. 3, 17

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, p. 436–444, 2015. 10

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. **Neural Computation**, v. 1, n. 4, p. 541–551, 1989. 18

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998. xv, 18, 19

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 (ECCV), 13., 2014, Zurich, Switzerland. **Proceedings...** Zurich, 2014. p. 740–755. 24, 38, 42, 46

LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y.; BERG, A. C. SSD: Single Shot Multibox Detector. In: EUROPEAN CONFERENCE ON COMPUTER VISION, 14., 2016, Amsterdam, Netherlands. **Proceedings...** Amsterdam, 2016. p. 21–37. 33

LOWE, D. G. Distinctive image features from scale-invariant keypoints. **International Journal of Computer Vision**, v. 60, n. 2, p. 91–110, 2004. 18

LU, D.; MAUSEL, P.; BATISTELLA, M.; MORAN, E. Land-cover binary change detection methods for use in the moist tropical region of the Amazon: a comparative study. **International Journal of Remote Sensing**, v. 26, n. 1, p. 101–114, 2005. 1

MALMROS, L. **Insect event extraction in LIDAR images using image analysis and convolutional neural networks**. 2018. 59 p. Thesis (Master in Mathematical Sciences) — Lund University, Faculty of Engineering, Sweden, 2018. 7

MANNING, C. D.; RAGHAVAN, P.; SCHUTZE, H. **An introduction to information retrieval**. Cambridge, England: Cambridge University Press, 2009. 158 p. (ISBN:0521865719). Available from: <http://www.informationretrieval.org>. Access in: 23 Dec. 2019. 41

MIGUEL, E. P.; REZENDE, A. V.; LEAL, F. A.; MATRICARDI, E. A. T.; VALE, A. T. do; PEREIRA, R. S. Redes neurais artificiais para a modelagem do volume de madeira e biomassa do cerradão com dados de satélite. **Pesquisa Agropecuária Brasileira**, v. 50, n. 9, p. 829–839, 2015. 7

PAVLAKIS, P.; TARCHI, D.; SIEBER, A. J. On the monitoring of illicit vessel discharges using spaceborne sar remote sensing-a reconnaissance study in the mediterranean sea. **Annales des télécommunications**, v. 56, n. 11-12, p. 700–718, 2001. 1

PÉREZ, A.; CHAMOSO, P.; PARRA, V.; SÁNCHEZ, A. J. Ground vehicle detection through aerial images taken by a uav. In: INTERNATIONAL CONFERENCE ON INFORMATION FUSION), 17., 2014, Salamanca, Spain. **Proceedings...** Salamanca: IEEE, 2014. p. 1–6. 29

POLYAK, B. T. Some methods of speeding up the convergence of iteration methods. **USSR Computational Mathematics and Mathematical Physics**, v. 4, n. 5, p. 1–17, 1964. 14

Ponti, M. A.; Ribeiro, L. S. F.; Nazare, T. S.; Bui, T.; Collomosse, J. Everything you wanted to know about deep learning for computer vision but were afraid to ask. In: SIBGRAPI CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES TUTORIALS (SIBGRAPI-T), 30., 2017, Niterói, Brasil. **Proceedings...** Niterói: IEEE, 2017. p. 17–41. ISSN 2474-0705. 20

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster R-CNN: Towards real-time object detection with region proposal networks. **Advances in Neural Information Processing Systems**, p. 91–99, 2015. 24, 26, 29, 33, 44, 48

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 9, p. 533–536, 1986. 14

SANTIAGO VALDARRAMA. **Confusion matrix in object detection with TensorFlow**. 2019. Available from: <https://github.com/svpino/tf_object_detection_cm>. Access in: 24 Dec. 2019. 43

SANTOS, J. S. **Modeling, identification and control of a tethered airship**. 2018. 128 p. Thesis (Doctoral in Aeronautical and Mechanical Engineering) — Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, 2018. 1

SECCO, N. R. **Training artificial neural network to predict aerodynamic coefficients of airliner wing-fuselage configurations**. 2014. 130 p. Thesis (Master in Science in Aerodynamics, Propulsion and Energy) — Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, 2014. 14, 16

SECCO, N. R.; MATTOS, B. S. de. Artificial neural networks to predict aerodynamic coefficients of transport airplanes. **Aircraft Engineering and Aerospace Technology**, v. 89, n. 2, p. 211–230, 2017. 7

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv**, v. 1409.1556, 2014. 24, 26

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 1, p. 1929–1958, 2014. 23

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 28., 2015, Boston, United States of America. **Proceedings...** Boston: IEEE, 2015. p. 1–9. 24, 26

TENSORFLOW. **TensorFlow**. 2019. Available from: <https://www.tensorflow.org>. Access in: 05 Dec. 2019. 36

_____. **Tensorflow detection model zoo**. 2019. Available from: <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>. Access in: 10 Dec. 2019. 46, 47

_____. **TensorFlow object detection API**. 2019. Available from: <https://github.com/tensorflow/models/tree/master/research/object_detection>. Access in: 05 Dec. 2019. 33

TYROLABS. **Faster R-CNN: down the rabbit hole of modern object detection**. 2018. Available from: <http://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/>. Access in: 13 Feb. 2019. 25

TZUTA LIN. **LabelImg**. 2019. Available from: <https://github.com/tzutalin/labelImg>. Access in: 05 Dec. 2019. 35

UIJLINGS, J. R.; SANDE, K. E. V. D.; GEVERS, T.; SMEULDERS, A. W. Selective search for object recognition. **International Journal of Computer Vision**, v. 104, n. 2, p. 154–171, 2013. 24

VELTE, M. **Semantic image segmentation combining visible and near-infrared channels with depth information**. 2015. 114 p. Thesis (Master in Computer Science) — Bonn-Rhein-Sieg University of Applied Sciences, Sankt Augustin, 2015. 7

XENONSTACK. **Automatic log analysis using deep learning and AI**. 2018. Available from: <http://www.xenonstack.com/blog/log-analytics-deep-machine-learning/>. Access in: 22 Aug. 2019. 17

XU, Y.; YU, G.; WANG, Y.; WU, X.; MA, Y. Car detection from low-altitude uav imagery with the faster r-cnn. **Journal of Advanced Transportation**, v. 2017, p. 10, 2017. 29

ZHANG, Y.; SONG, B.; DU, X.; GUIZANI, M. Vehicle tracking using surveillance with multimodal data fusion. **IEEE Transactions on Intelligent Transportation Systems**, v. 19, n. 7, p. 2353–2361, 2018. 29