Pontificia Universidad
JAVERIANA
Colombia

Trabajo de grado

Low Computational-complexity algorithm for the estimation of traffic parameters using spatio-temporal images and convolutional neural networks for real-time traffic monitoring.

Author :
Guillaume Poullain

Director:
Francisco Carlos Calderon Ph.D.

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERA
DEPARTAMENTO DE ELECTRONICA
BOGOTA D.C.
2019

Advertencia

"La Universidad no se hace responsable por los conceptos emitidos por los alumnos en sus trabajos de grado, solo velará porque no se publique nada contrario al dogma y la moral católicos y porque el trabajo no contenga ataques y polémicas puramente personales, antes bien, se vean en ellas el anhelo de buscar la verdad y la justicia."
Reglamento de la Pontificia Universidad Javeriana, articulo 23, de la Resolucion 13, de Julio de 1965.

Acknowledgement

Firstly, I would like to thank my thesis director who helped me throughout that work and allowed me to work on a promising project. He provided me the basic key and understandings to develop the idea and was always available to help me tackle my issues and questions.

Also, I'm very grateful to the Pontificia Universidad Javeriana that accepted me here and gave me the chance to study in one of the best universities in Colombia. I would also thank my own university in France: INP-ENSEEIHT that accepted my wish to study abroad for my last year of university.

Finally, I would like to thank all my family and friends who always believed in me. And, especially, my parents who taught me that studying hard will always be rewarding and who always supported me to promote what I wanted to do the most.

Contents

1 Introduction **1**

2 Theoretical framework **3**
2.1 Fundamentals of traffic engineering . . . 3
2.2 Generality about counting systems . . . 3
2.3 Digital image processing . . . 3
2.4 Computer vision . . . 3
2.5 Semantic segmentation . . . 3
2.6 Spatio-temporal images . . . 4
2.7 OpenCV . . . 4
2.8 Deep learning and convolutional neural network . . . 4
2.8.1 Deep learning . . . 4
2.8.2 Convolutional neural networks . . . 5
2.9 Evaluation of supervised learning methods . . . 5
2.9.1 Precision, recall and intersection over union . . . 5
2.9.2 Average precision . . . 6
2.10 Training parameters . . . 6
2.10.1 Batch and subdivision . . . 6
2.10.2 Learning rate . . . 6
2.10.3 Anchors . . . 6
2.11 YOLO . . . 6

3 Development of the project **8**
3.1 Videos from *"Secretaría Distrital de Movilidad"* . . . 8
3.2 Open source algorithm for classifying and counting . . . 8
3.3 Creation of the dataset . . . 8
3.3.1 Spatio-temporal images . . . 9
3.3.2 Labeling images . . . 9
3.3.3 Transformation of the labels to YOLO input . . . 13
3.4 Convolutional neural network . . . 13
3.4.1 Presentation of the dataset . . . 13
3.4.2 YOLOv2 . . . 14
3.4.3 Tiny-YOLOv2 . . . 15
3.4.4 YOLOv3 . . . 15
3.4.5 Tiny-YOLOv3 . . . 16
3.4.6 YOLOv3 with a larger data set . . . 16
3.5 Counting objects . . . 17
3.6 Presentation of the graphic interface . . . 17

4 Results **19**
4.1 Training process . . . 19
4.2 Counting process . . . 22
4.2.1 Results with the main camera used for training . . . 22
4.2.2 Results of the model trained with a larger data set . . . 25
4.3 Complementary time test . . . 26

5 Conclusions and future works **27**

Bibliography **27**

Appendices **30**

List of Figures

2.1 Example of spatio-temporal image used in photo-finish, image taken from [15]. 4
2.2 Block diagram representing the machine learning process, image taken from [18]. 4
2.3 Training loop of a neural network,image taken from [18]. 5
2.4 Precision, recall and IoU in the case of bounding box detection, image taken from [19]. . 6

3.1 Example of detection of the open source algorithm, with the bounding box and class for each detected object. 8
3.2 The line of the STI in figure 3.2b and the obtained STI. 9
3.3 Block diagram representing the automatic labelling process. 9
3.4 The line of the STI on the modify video in figure 3.4a and the gotten STI in figure 3.4b. . 10
3.5 Block diagram representing the image processing of the labelling image. 11
3.6 The blue line is a parallel to the x-axis in the original image and to the y-axis in the STI; the red line is the line where the spatio-temporal images are created, these lines are also marked on the STI to point out that the angle alpha is the same in both images. 11
3.7 The yellow block of figure 3.6b is now separated in the corresponding two object-bocks thanks to the histogram suppression algorithm. 12
3.8 Examples of bounding boxes obtained with the automatic labeling algorithm. Each class is represented by a colour, in green: motorbikes, in blue: taxis, in dark: personal car, in yellow: trucks. 13
3.9 Architecture of YOLOv2, 24 convolutional layers followed by 2 fully connected layers, taken from [27]. 14
3.10 Architecture of YOLOv3, taken from [29]. 15
3.11 Window to chose the desired trained model. 17
3.12 Window to choose the length of the spatio-temporal image. 18
3.13 Window to choose the video to test the algorithm. 18
3.14 Report of the detected objects in the csv file with the number of detected object for each class, the detection time, the total time to run the algorithm, the number of frames per second performed by the algorithm, the length of the spatio-temporal image and the model. 18

4.1 Loss error and mAP in function of the number of iterations with YOLOv2. 19
4.2 Loss error and mAP in function of the number of iterations with tiny-YOLOv2. 20
4.3 Loss error and mAP in function of the number of iterations with YOLOv3. 20
4.4 Loss error and mAP in function of the number of iterations with tiny-YOLOv3. 21
4.5 Loss error and mAP in function of the number of iterations with YOLOv3 trained with a larger data set. 21
4.6 Comparison of bounding box for the four models. 24

List of Tables

3.1 Table presenting the different classes of object to detect, their number and the corresponding pixel value used to create the grey video. . . . 10
3.2 Table presenting the different classes to detect and their respective minimum width to be detected. . . . 12
3.3 Main parameters used to train YOLOv2. . . . 14
3.4 Main parameters used to train tiny-YOLOv2. . . . 15
3.5 Main parameters used to train YOLOv3. . . . 16
3.6 Main parameters used to train tiny-YOLOv3. . . . 16
3.7 Main parameters used to train YOLOv3 with a larger data set. . . . 17

4.1 Number of object counted for each class with each method, number total of object detected and detection time (Det. time) for a 15 minutes video. The detection time is the running time of the function *detect* of Darknet. . . . 22
4.2 Number of object counted for each class with each method, number total of object detected and detection time (Det. time) for a 15 minutes video. The detection time is the running time of the function *detect* of Darknet. . . . 23
4.3 Effectiveness compared to manual counting for each class with each method as well as their respective mean and standard deviation (std). . . . 23
4.4 Number of object counted for each class with each method, number total of objects detected and detection time (Det. time) for the 15 minutes sidewalk video and the corresponding effectiveness relative to manual counting as well as the corresponding mean and standard deviation (std). . . . 25
4.5 Number of object counted for each class with each method, number total of objects detected and detection time (Det. time) for the 30 minutes video with low traffic and the corresponding effectiveness relative to manual counting as well as the corresponding mean and standard deviation (std). . . . 26
4.6 Processing time characteristics with an NVIDIA JETSON TX2 [32]. . . . 26

Chapter 1

Introduction

For years traffic counting has been a challenge. It is the basis of managing and understanding traffic congestion. Traffic jam is a significant problem in all big cities [1] because of its effects on the economy, pollution and consequently on people health and living conditions. There are different methods for counting vehicles. The simplest one is manual counting; a person counts the number of vehicles crossing a virtual line on the road. This method is 99% accurate [2]. However, it takes a long time and cannot be doing continuously along hours or days. For this reason, more sophisticated methods have been developed such as inductive loops, magnetic sensors, acoustic detectors, piezoelectric sensors, pneumatic road tube counting, and others. However, some of them need to be installed under the road which requires a lot of work. On the contrary, video-based traffic monitoring requires cameras on the road which are often already installed in big cities, for example, surveillance cameras or traffic control cameras. Moreover, a lot of other features can be extracted from the video stream and in real time. Consequently, video analysis is the present and future of traffic monitoring. The real-time data obtained from traffic video monitoring is a necessary element of Intelligent Traffic Systems (ITS). This can afford for example adaptive traffic-light control at intersections to reduce waiting time [3]. Some countries have already embraced ITS like the USA, Japan and UK [1].

A video-based approach for traffic monitoring is an important field of research. Many different techniques have already been proposed. Some of them can extract a lot of parameters from the video stream like color, vehicle classification, plate number, vehicle speed. Others only monitored the flow of vehicles to reduce the complexity of the algorithm and to get real-time data without requiring high computational power.

One of the huge challenges of vehicle monitoring is processing time. In the article "A Low-Complexity Vision-Based System for Real-Time Traffic Monitoring" [4], the author wanted to develop a fast-performing vision-based system. To realize that, they only make an algorithm that counts the number of cars. It does not have features such as vehicle speed, vehicle classification, or even plate number detection. The performance is prioritized over the amount of information. The presence of moving vehicles is determined by an adaptive threshold. The proposed method has the following constraints: the camera is placed above the road and the angle of the camera with the road is between 45 and 90 degrees. The resolution of the camera is greater or equal to 320 x 240 pixels, it remains in the same position and the video stream is superior at 10 frames per second. The algorithm uses virtual detectors who are intended to work as inductive loop detectors. These detectors must cover one lane of the road and must be crossed in one direction and should not be obstructed by another vehicle. The motion detection is based on the comparison of the light of the pixel with a reference image. One of the problems is that a variation on the scene's illumination could lead to a false positive. In order to tackle this problem, the algorithm uses an adaptive background and a brightness compensation algorithm. To maximize the detection of a vehicle, the virtual detectors as defined as a pair of detection regions which allowed a comparison and more accurate results. Finally, this algorithm offers a low-cost computer vision system with good results but with a lot of constraints. For example, visual occlusion is not managed by the algorithm because that will require a huge time processing. Moreover, the output algorithm is only the number of vehicles.

In real time vehicle detection, it can be very interesting to know other characteristics of vehicles as vehicular volume, average speed, and class, for example, to manage traffic congestion. In the following paper: "High-efficient Detection of Traffic Parameters by Using Two Foreground Temporal-Spatial Images" [5], they try to get all these characteristics using a low complexity algorithm based on temporal-

spatial images obtained from two virtual lines in a video frame. In the proposed system, the camera is installed at a roadside and its optic axis is down towards the road. The virtual detection lines are set to be perpendicular to lane lines. Firstly, to decrease the amount of information processed, the video frames are transformed from RGB to Grey mode, then median filter algorithm is used to reduce noise. Then the algorithm is divided into three parts: foreground time space images (TSI) generation, blobs process in TSIs, parameter extraction. The algorithm was tested with Beijing Xidawang road and Langfang Aiminxi road under different weather conditions. The video frame was 640*480 at 30 fps. The results obtained for the detection ratio are between 91,10% and 96,73% depending on the weather condition. The mean speed is in the same average that other methods of comparison. The classification accuracy ratio is around 90%. Processing results are not exposed but they say that the average processing time per frame is reduced about 50% with comparable algorithms.

A similar approach was proposed in "Low Complexity Algorithm For The Extraction Of Vehicular Traffic Variables" [6]. The algorithm proposed acquires vehicular volume and mean speed processing two-pixel lines of the image and using spatio-temporal accumulation technique. The first stage of the algorithm is to create two binary spatio-temporal maps, then the contour is extracted and finally, the algorithm calculates speed and volume. Using spatio-temporal map reduces the amount of information processed by a factor 300 and the memory used for running the algorithm. Vehicle counting has very good results, around 98% and mean speed is accurate with a variance of 5%. The technique using spatio-temporal image is working; the next step could be to classify the vehicle using almost the same processing cost.

Numerous algorithms already exist for traffic monitoring, and it is difficult to find a good compromise between time processing and the desired output features. Everything depends on what the user wants to extract from the video stream. If the user wants a very fast algorithm that only counts the number of vehicles, using visual detectors is very efficient and at a low processing cost [4]. For more features and high efficiency using spatio-temporal accumulation is really processing saving and get more features like mean speed and vehicular volumes [5], [6]. To go further in traffic monitoring, getting vehicle classification could be very useful. In fact, it will give information about who uses the road. E.g. taxis, buses or trucks. Some methods already exist, as proposed in [7], but it is using an embedded system to count and classify in real time vehicles. With a video-based system, they will have no need to install the system because cameras are already available on numerous roads.

In the present work, a new approach is proposed. It uses spatio-temporal images and runs a convolutional neural network model on it to detect and classify the objects. Thus, the processing time is reduced enormously in comparison to a method where the neural network is applied to each frame of the video. However, video quality and vehicle speed influence a lot the result of the algorithm. Moreover, that algorithm only counts vehicles for each object-class but does not provide other information as for example speed.

The work presents some basics theoretical frameworks to better understand the core of the thesis in chapter 2. Then, it proposes a method to create a data set of spatio-temporal accumulation, that is used to train the network in chapter 3. Few different convolutional neural networks (CNN) are trained and challenged against manual counting and an open source algorithm [8] using a CNN to track and then count vehicles. The results of the different models are presented in chapter 4 and is followed by the conclusion in chapter 5.

Chapter 2

Theoretical framework

The theoretical framework of that work explains some basic concepts linked to the subject and details the different software and libraries used through it.

2.1 Fundamentals of traffic engineering

Traffic engineering is part of civil engineering in charge of planning and designing traffics in cities. It exists numerous traffics variables to study the state of the traffic, some of them are vehicle speed, density, time headway, volume, etc [9]. In the scope of that work the principle traffic variable monitored will be the number of vehicles on a road.

2.2 Generality about counting systems

The counting problem is the prediction of the number of objects in an image. In computer vision, the most frequent strategies for counting are the following: Line of interest (LOI) and region of interest (ROI) [10]. LOI is based on the number of objects in movement crossing a virtual line. In general, the object is detected and then using the optical flow followed and counted when it crosses the line. ROI is counting the number of objects in a specific region of interest [11]. The object can be in movement or not and that technique requires more processing time because it needs to classify the objects.

2.3 Digital image processing

Image processing is part of computer science. Generally speaking, it is considered as any form of signal processing with an image input, such as a photo or video frames. The main application areas are the improvement of pictorial information for human interpretation, processing for transmission or data storage and representation of the world for autonomous machine [12].

2.4 Computer vision

Computer vision is the study of techniques and theories extracting useful information from visual information in order to interpret the world. It includes the acquisition by a camera, processing, analyzing and understanding of real images by a computer. In that way, traffic variable acquisition using CCTV enters the field of computer vision.

2.5 Semantic segmentation

Semantic segmentation is an image processing method assigning a label to each pixel of an image [13]. The algorithm outputs ranked list of boxes with categories labels and the probability of the detected object.

2.6 Spatio-temporal images

Spatio temporal images also called spatio-temporal accumulation are an accumulation of images through time. A single image in 2D gives only spatial information while a spatio-temporal image contains both the temporal and spatial structure of the observed phenomena [14] all contained in a three dimensions image. Spatio-temporal images are often used in photo-finish line in athletic sports as illustrated in figure 2.1. In that case, only one line is kept throughout time and a 2D image is obtained.

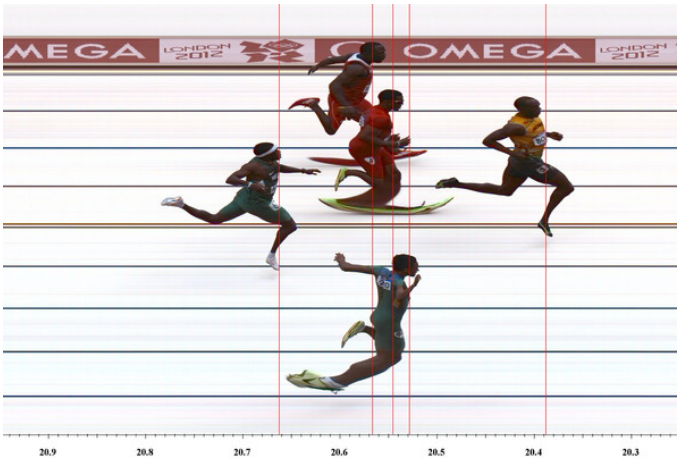


Figure 2.1: Example of spatio-temporal image used in photo-finish, image taken from [15].

2.7 OpenCV

OpenCV stands for Open Source Computer Vision Library, it is an open source computer vision and machine learning software library [16]. The library has more than 2500 optimized algorithms written natively in C++ including state-of-the-art algorithms for face recognition, camera tracking, 3D modeling in others. It has C++, Python, Java and Matlab interfaces and can be executed from Windows, Linux, Android and Mac OS. The library is used by more than 47000 people and by numerous companies, research groups and governments.

2.8 Deep learning and convolutional neural network

2.8.1 Deep learning

Deep learning is part of machine learning that is part of artificial intelligence. Artificial intelligence (AI) was born in the 1950s, the goal was to automate intellectual tasks normally performed by humans. The first AI programs were hardcoded rules crafted as for example chess programs called "Deep Blue" that beat world chess champions Garry Kasparov in 1997 [17]. Machine learning differs from old rule programming by using data and answers to learn rules that will then be used to classify, see figure 2.2. A machine learning model is trained rather than explicitly programmed. Deep learning is a specific sub-field of machine learning, it learns representation of the data in successive layers of increasingly meaningful representations [18].

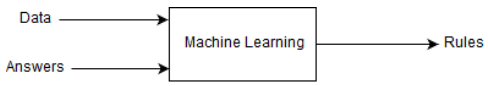


Figure 2.2: Block diagram representing the machine learning process, image taken from [18].

The word deep is used because the number of layers of the model represents the depth of it. In general, layered representation is learned throughout a model called neural networks. The learning process of a neural network is represented in figure 2.3.

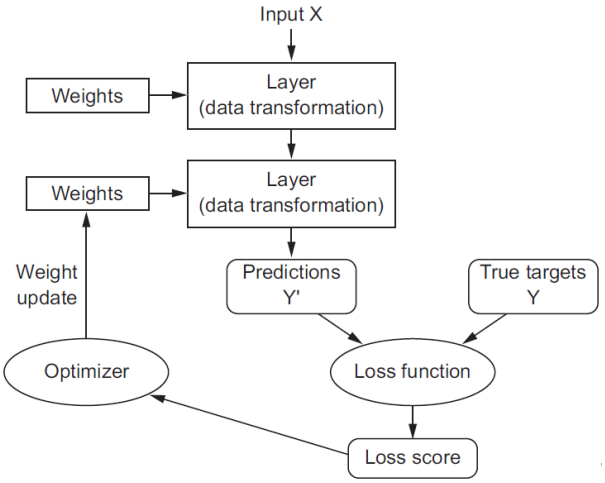


Figure 2.3: Training loop of a neural network,image taken from [18].

Initially, the weights of the network are assigned randomly. Each weight is updated throughout the process. The model prediction is compared with the true targets using the loss function. The loss function or objective function of the network computes a distance score between the targets and the outputs to obtain the loss score. Then an optimizer is used to update the weight of the network for example with the backpropagation algorithm and then the prediction is made with the new layers' weight. These actions are repeated until the loss score is reached, or the loss score converges during various iterations.

2.8.2 Convolutional neural networks

Convolutional neural networks (CNN) is a specific kind of neural network. In CNN, layers learn local patters; in the case of images, patterns of the 2D windows found in the image. The main characteristics of a convnet for a convolutional network are:

- The patterns are translation invariant. When a certain pattern is learned in a part of an image the network can recognize it anywhere. This is very efficient because the visual world is translation invariant and the network needs a fewer sample to generalize the pattern.
- They can learn spatial hierarchies of patterns. That means that the first layers will be closer from the initial representation and deeper layers more abstract.

2.9 Evaluation of supervised learning methods

2.9.1 Precision, recall and intersection over union

Precision measures how accurate is the prediction, precision is given by equation 2.1. Recall measures how good the positive is found and is given by equation 2.2. Defining the following parameter: TP for True Positive, TN for True Negative, FP for False Positive and FN for False Negative, recall and precision are defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

Intersection over Union (IoU) measures the overlap between two boundaries. It is used to measure how well the predicted bounding box overlaps with the ground truth. These three concepts are graphically presented in figure 2.4.

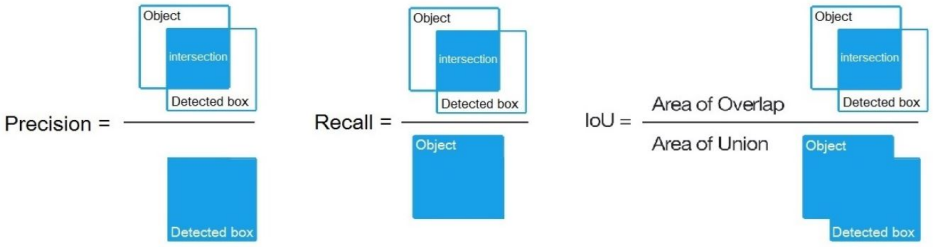


Figure 2.4: Precision, recall and IoU in the case of bounding box detection, image taken from [19].

2.9.2 Average precision

Average precision (AP) represents the area under Precision vs Recall curve, see equation 2.3.

$$AP = \int_0^1 p(r)dr \tag{2.3}$$

where $p(r)$ is the precision vs recall curve. The integral is always between 0 and 1, so the integral is inferior to one. It exists different manners to interpolate the AP. One of the most famous is the COCO mAP for mean AP, COCO is a dataset of images. In COCO mAP 101-point interpolated AP definition is used in the calculation. For COCO, AP is the average over multiple IoU.

2.10 Training parameters

2.10.1 Batch and subdivision

The batch size indicates how many images are used in one iteration of the training process to update the parameters of the neural network.

Because it can be memory-hungry to load for example 64 images in one time, subdivision allowed to decompose the batch in the number of subdivisions. Then, the GPU processes $\frac{Batch}{subdivision}$ number of images at any time and the iteration will be completed after having processed the whole batch.

2.10.2 Learning rate

The learning rate of a network controls how fast the network should learn, or more exactly the magnitude of the move of parameters in the opposite direction from the gradient per iteration [18].

2.10.3 Anchors

Anchors could be defined as default bounding boxes of the network. It is largely used in modern objects detector because predicting the width and height of a bounding box leads to unstable gradient during training [20]. These are used to predict the exact bounding box from the cell of the prediction feature map.

2.11 YOLO

YOLO for You Only Look Once is a method to do object detection, the strategy behind the code. The first version is YOLO9000, a real-time object detector that can detect up to 9000 objects [20]. The official implementation is available through Darknet. Darknet is an open source custom neural network framework written in C and CUDA, it is fast and supports CPU and GPU computation [19]. Earlier detection frameworks are looking at the images several times and with different scales to expect to find the object. YOLO takes a different approach and looks at the entire image only once and goes through the network once and detects objects. The improvement of YOLO9000 led to YOLOv2, a state-of-the-art real-time object detection algorithm.

YOLO uses a completely different approach than classical CNN as R-CNN or Fast R-CNN [21]. In fact, a simple neural network is applied to the full image, the network divides the images into regions

and realizes the prediction of bounding boxes for each region. YOLOv2 used a fully convolutional model but still train on the whole image.

YOLOv3 is using multi-scale predictions, better backbone classifier among others to improve in terms of performance and training process the previous version: YOLOv2.

First, YOLO divides the image into a 13 x 13 grid of cells. Each cell is then responsible for predicting a number of boxes in the image. YOLO uses non-maximum suppression to eliminate bounding boxes for their confidence or because they are enclosing the same object. In fact, each bounding box has a confidence score that reflects how likely the box contains an object. To update the network during the training process, YOLO uses the sum-squared error between the predictions and the ground truth to calculate the loss. Thus, the loss function is composed of three different losses: the classification loss, the localization loss and the confidence loss. The classification loss is, if an object is detected, the squared error of the class conditional probability for each class. The localization loss measures the error in the predicted location and size of the bounding box. The confidence loss consists of measuring the objectness of the box.

Chapter 3

Development of the project

3.1 Videos from *"Secretaría Distrital de Movilidad"*

The videos used to design and test the algorithm are provided by the *"Secretaría Distrital de Movilidad de Bogota"*. It is an entity looking forward to making of Bogotá a pleasant city for his inhabitants and tourists in terms of mobility, life conditions, competitiveness, etc. There are around one hundred cameras available. These numerous cameras offer different views of the road. Videos have a frame rate of 20 images per second.

3.2 Open source algorithm for classifying and counting

There is an open source algorithm developed by the *"Secretaría Distrital de Movilidad"* that classify and counts vehicles [8]. The algorithm uses YOLOv2 [20] and is written in Python 2.7 [22]. The algorithm used a YOLOv2 model trained with around 70000 labels to detect vehicles and pedestrians for each frame of the video. Then with a system of optical flow based on the centroid of the bounding box, it counts for each class the number of objects crossing a line. In good conditions and with a well-placed camera, the algorithm reaches a precision of 97,6% [8].

Most of the program explained in the following parts is built on top of that algorithm and the results will be challenged against it.

Figure 3.1: Example of detection of the open source algorithm, with the bounding box and class for each detected object.

3.3 Creation of the dataset

The most important part of that work was to create a usable dataset to then train the neural networks. The following sections present the different stages to obtain a complete dataset.

3.3.1 Spatio-temporal images

Firstly, spatio-temporal images were created because the neural network will be trained on these spatio-temporal accumulations. The spatio-temporal accumulation was created using the Bresenham's line algorithm [23]. It is a line drawing algorithm that determines the points of an n-dimensional raster to realize a close approximation of a straight line between two points. Thanks to that algorithm, the selected pixel value is kept for each pixel forming the line and for each frame of the video to create the spatio temporal image. For example in figure 3.2a, a red line is drawn and the pixel of that line are accumulated to get the figure 3.2b.

Because YOLOv2 resize the input images to a size of 416 for training and inference, the spatio-temporal images were defined to represent 416 frames of the video. The video frame rate is of 20 frames per second (fps) so one image represents 20.8 seconds of video. Thus, the width of the spatio-temporal image (STI) will always be of 416 pixels and the high of the image is variable depending on the length of the red line drawn on the video.

(a) Example of a camera view with the red line where the spatio-temporal accumulation is performed.

(b) STI obtained for some frames of the view presented in figure 3.2a

Figure 3.2: The line of the STI in figure 3.2b and the obtained STI.

3.3.2 Labeling images

Labeling the STI was one of the most difficult part of the work. In fact, as the figure 3.2b shows it can be complicated to determine to which class an object of the STI belongs to. Also, a neural network requires a huge number of images to be trained, thus an automatic labeling algorithm was designed. The following block diagram represents the process of that last one, figure 3.3.

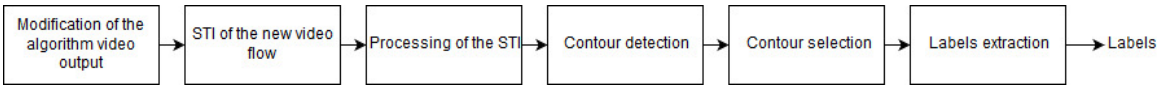


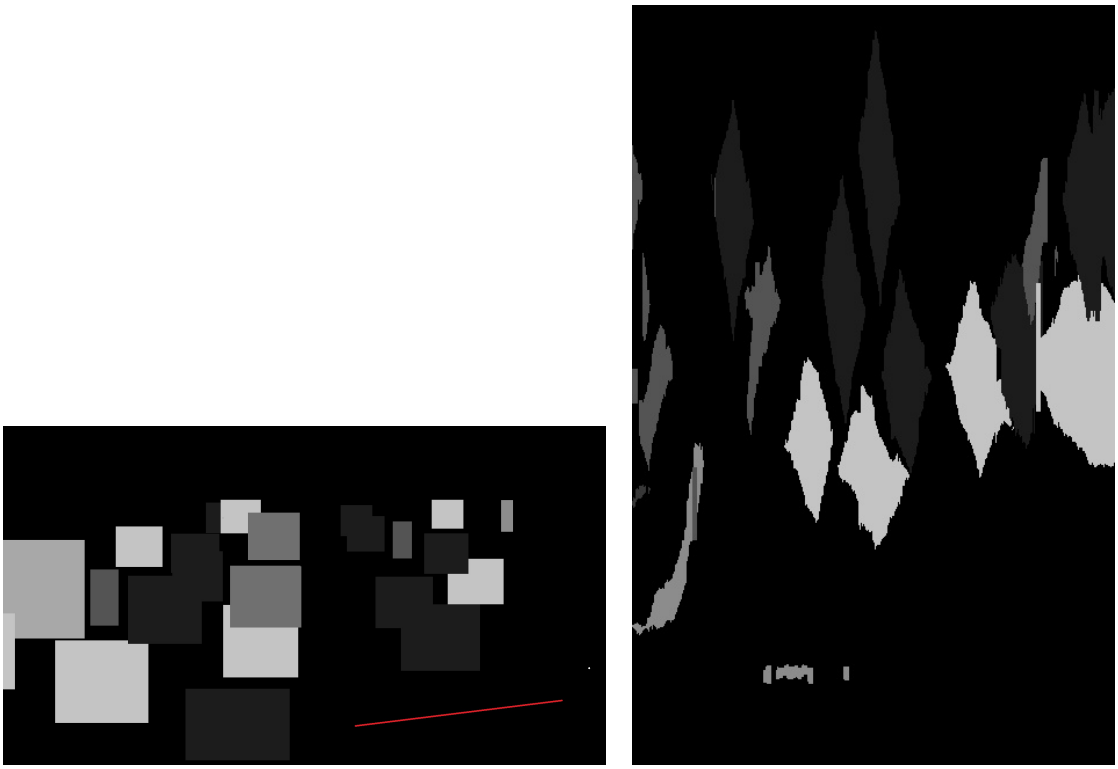
Figure 3.3: Block diagram representing the automatic labelling process.

Each block of the algorithm is explained in the following subsections.

Modification of the algorithm to create STI for labeling

The already working detection algorithm presented in section 3.2 shows in real time the bounding boxes of the detected objects, see figure 3.1. Probably in reason of the way the algorithm was trained, it detects easily big objects than small one. To prevent missing detection the line must be placed on the image where the objects are big enough to be detected by the network. Thus, the line must be placed in the foreground of the video.

In the following part labeling image means a grey scale image allowing to realize the labeling of the STI. The labeling image is also a STI, it is performed with a black background and then the bounding boxes of each object are fully filled with a specific value for each class of object as show figure 3.4a. The value are presented in table 3.1, they are calculated with a simple formula $v_{pixel} = int(cl * 255/9)$. Moreover, classes have a specific order. Classes that could be mingled by the algorithm for example buses and trucks or motorbikes and bicycle have pixel values far from each other. Thus, a STI is performed with that new video, an example of the video frame and the resulting STI are presented in figure 3.4b.



(a) Grey video with filled bounding box.

(b) STI from figure 3.4a.

Figure 3.4: The line of the STI on the modify video in figure 3.4a and the gotten STI in figure 3.4b.

Class	Particular	Bus	Motorcyclist	Minivan	Pedestrian	Truck	Taxi	Cyclist	Trucking rings
Class #	1	2	3	4	5	6	7	8	9
Pixel value	28	56	85	113	141	170	198	226	255

Table 3.1: Table presenting the different classes of object to detect, their number and the corresponding pixel value used to create the grey video.

Processing of the labeling STI and contour detection

The image presented in figure 3.4b is not precise enough to realize contour detection and then to label. It must be processed. The algorithm to process the image is presented in the following block diagram, figure 3.5.

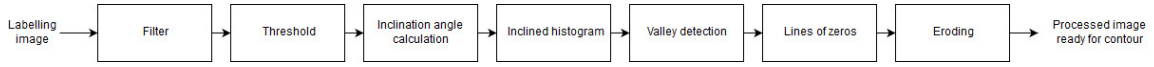
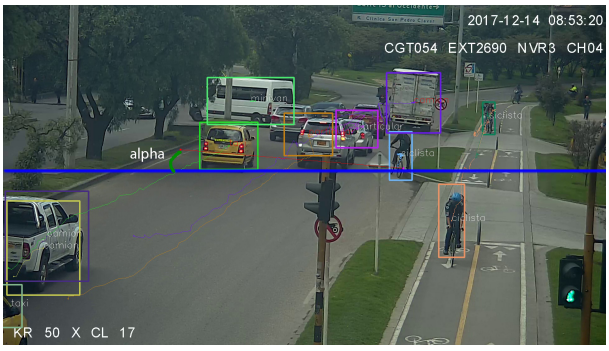


Figure 3.5: Block diagram representing the image processing of the labelling image.

Firstly, a median blur with a square kernel of size 5 is applied to the labeling image with the function *"medianBlur"* of openCV [16]. Then, for each class, the same operations are repeated. Each class corresponds to a pixel value presented in table 3.1. Because of some mix of values in the labeling image, a ±5% boolean threshold is applied to keep only the pixel of the corresponding class. We noticed that when objects were very closed in the video, vehicles have almost no safety distance between them, they are forming only one block of pixels as shows figure 3.6b. In order to separate the object, a heuristic algorithm was developed. This algorithm uses the histogram of the image with an inclination depending on the line drawn on the video. A schematic representation of the angle is presented in figure 3.6a. As the image 3.6b shows between two objects of the same class there is an inclined line (the red line in figure 3.6b) where the number of pixels belonging to a class is inferior. That angle is the same as the one presented in figure 3.6a. The angle is calculated, and that same angle is used to realize an inclined histogram of the labeling image. The histogram is performed along the width of the image. The red line in figure 3.6b shows an example of the selected pixel to create the histogram.



(a) Illustration of the angle on the original image.

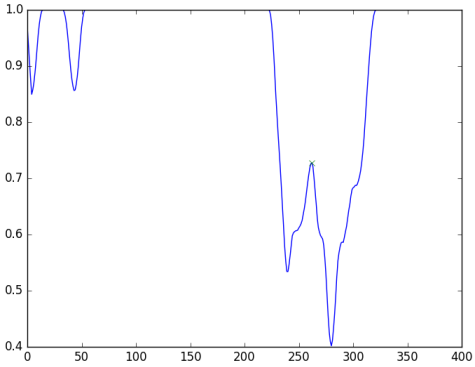
(b) STI from figure 3.6a before filtering and using the histogram method. In yellow the block that should be divided in two.

Figure 3.6: The blue line is a parallel to the x-axis in the original image and to the y-axis in the STI; the red line is the line where the spatio-temporal images are created, these lines are also marked on the STI to point out that the angle alpha is the same in both images.

When too many objects are very close to each other in the video stream, they are forming one block. But there is a moment where the number of pixels belonging to a class in the direction of the inclined line is lower. The histogram is performed with the inclination explained previously on the labeling STI. To find the valley separating two objects the histogram is firstly normalized to one and then the negation of the histogram is taken. Thus, the function *"find_peaks"* from the library [24] is used to find peaks. The attribute prominence of the function was used to detect only the peaks with a prominence superior to 0.1, also the peaks whose the value was superior to 0.97 were eliminated because they are representing

gaps between two objects of the same class. The *"find_peaks"* function returns the position of the peaks and so on of the valley in the image. The next step is to put zeros in the image. Each value of the histogram represents a line in the image. The value of the histogram valley is used to put zeros in the corresponding line of the labeling image. Then, to perform a good contour detection the image is eroded with a kernel of one of size 2x2. The image is shown in figure 3.7a is obtained. Objects previously forming a single block are now separated by a line of zeros.

(a) Taxi class after filtering, histogram suppression algorithm and eroding.



(b) Negation of the inclined histogram of the image 3.6b for taxi class. The green mark represent a valley a valley with a prominence superior to 0.1.

Figure 3.7: The yellow block of figure 3.6b is now separated in the corresponding two object-bocks thanks to the histogram suppression algorithm.

Contour extraction and labelling

Thus, contour detection is applied using the openCV function. The function returns all the points of the contour for each contour. To create the bounding box, the minimum and maximum value in x and y are extracted for each contour and added to the label of the class. Moreover, some boxes are rejected if their size or area is too small using table 3.2. The width value is equal at the number of pixels in time and is the minimum number for each class. It has been found by experimentation. The area is variable because it depends on the length of the drawn line. The corresponding minimum area is calculated with the following equation $Area_{min} = width * N_{points}/3$ with N_{points} the number of pixels of the line. Also, the area and size are strongly linked to the speed of the vehicle in the street and how big the objects appear in the video. So, depending on these two factors it may be required to change the proportional factors to prevent missing objects.

Class	Particular	Bus	Motorcyclist	Minivan	Pedestrian	Truck	Taxi	Cyclist	Trucking rings
Width	9	40	5	20	30	40	9	5	40

Table 3.2: Table presenting the different classes to detect and their respective minimum width to be detected.

Examples of the obtained labels represented by bounding boxes are presented in figure 3.8. Each class is represented by a color and the coordinate of the bounding boxes are kept in a text file that will be used to train the neural networks.

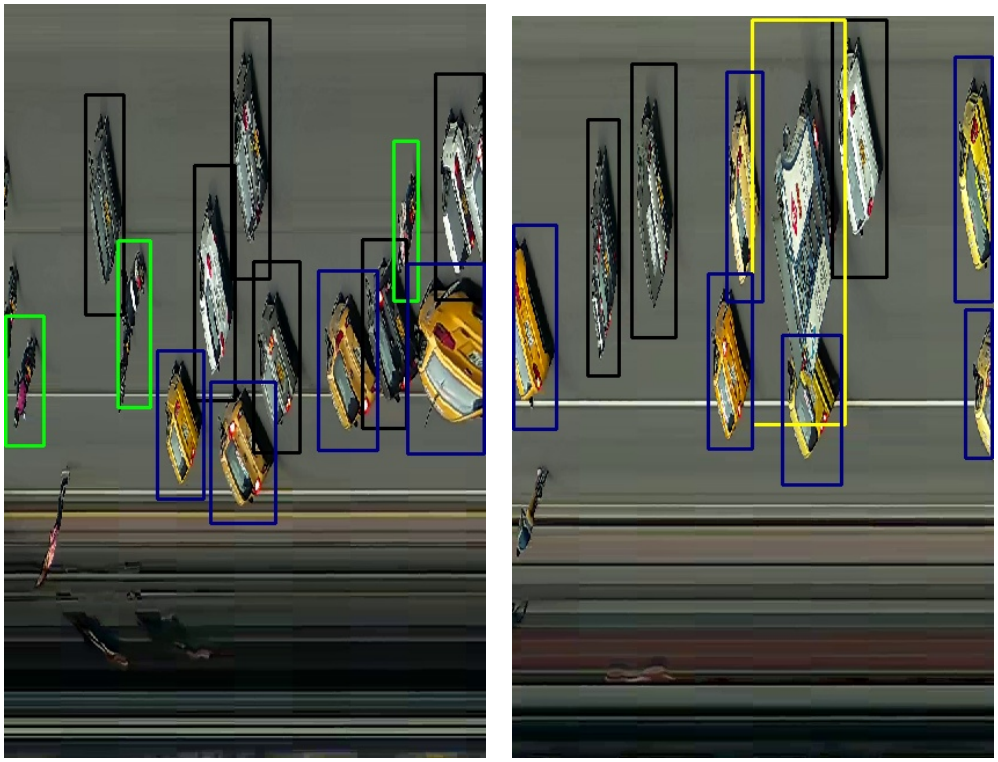


Figure 3.8: Examples of bounding boxes obtained with the automatic labeling algorithm. Each class is represented by a colour, in green: motorbikes, in blue: taxis, in dark: personal car, in yellow: trucks.

3.3.3 Transformation of the labels to YOLO input

When creating the labels, the maximum and minimum in x and y of the contour were taken to create labels with the following form: $< x_{min} >< y_{min} >< x_{max} >< y_{max} >< object - name >< - >< object - number >$.

YOLO requires labels with a different notation. The .txt of the YOLO labelling must have the following presentation: $< object - class >< x >< y >< width >< height >$ Where:

- $< object - class >$ is the number of the object class from 0 to (classes-1).
- $< x >$, $< y >$, $< width >$ and $< height >$ are float values relative to width and height of the image, it takes values in the interval [0,1]. For example, $x = |x|/image_{width}$ and $height = |height|/image_{height}$.
- $< x >$ and $< y >$ are the coordinate of the centre of the bounding box.
- $< width >$ and $< height >$ are the relative value of the width and height of the bounding box.

The labels are all transformed into that new format with a third-party open-source algorithm [25]. While the algorithm creates the training list required by YOLO as input.

3.4 Convolutional neural network

3.4.1 Presentation of the dataset

Few exploitable videos that is to say with good weather, low vehicle velocity and good camera position over the road were available. Thus, that dataset was realized with only one camera. Videos from that camera were in the number of five, each one of 15 minutes lengths. Thus, 1 hour and 15 minutes of video were used to create the dataset. To create more images, three different accumulation lines were drawn on the video to improve diversity in the dataset. In total, the data set is composed of 512 spatio-temporal images and each of them includes around 11 objects. Consequently, the number of objects in the data set is close to 5000 objects. For training, all these images were used and for the validation set, 100 images were randomly selected.

3.4.2 YOLOv2

Network

The architecture of YOLOv2 is inspired by GoogleLeNet [26]. It has 24 convolutional layers followed by 2 fully convolutional layers. Also, a 1 x 1 reduction layers are used and followed by 3 x 3 convolutional layers, as shows the figure 3.9. The output of the network is a prediction tensor of size 7 x 7 x 30. The initial layers of the network extract features from the image while fully connected layers predict the output probabilities and coordinates of the box [27].

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 3.9: Architecture of YOLOv2, 24 convolutional layers followed by 2 fully connected layers, taken from [27].

Training configuration

The principal components of the training configuration are presented in table 3.3. The last convolutional layer has 70 filters. In fact, for YOLOv2, the $filters = 5 * (n_{classes} + 5)$. Specific anchors of the data set were calculated with k-means [28]. The anchors for YOLOv2 are the following: 1.2522, 4.6962, 3.1879, 5.4931, 1.7093, 10.7377, 3.8239, 13.3217, 10.5668, 11.7837. They are representative of the anchor present in the data set for training with YOLOv2.

batch	subdivisions	learning rate	max batches
64	8	0.001	100000

Table 3.3: Main parameters used to train YOLOv2.

3.4.3 Tiny-YOLOv2

Network

Fast YOLO or tiny-YOLOv2 uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, training and testing parameters are the same between YOLOv2 and Fast YOLO [27].

Training configuration

Because the network is lighter, the number of subdivisions were decreased to 2. Moreover, as training with 100000 was a bit useless because the mAP started converging earlier, the number of iterations was reduced to 50000. Thus, the training time was cut down by a factor 2.

batch	subdivisions	learning rate	max batches
64	2	0.001	50000

Table 3.4: Main parameters used to train tiny-YOLOv2.

3.4.4 YOLOv3

Network

Since YOLOv3 is an improvement of YOLOv2, it has some hybrid part of the YOLOv2 network, but it is also significantly larger. In fact, it has 53 convolutional layers and uses successive 3 x 3 and 1 x 1 convolutional layers and add some short-cuts. Thus, the network is bigger but remains very efficient [29]. The architecture is presented in figure 3.10.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.10: Architecture of YOLOv3, taken from [29].

Training configuration

The principal components of the training configuration are presented in table 3.5. Also, each of the last convolutional layers has 42 filters because there are 9 classes. In fact, for YOLOv3, the $filters = 3*(n_{classes}+5)$. Specific anchors of the data set were calculated with k-means. The anchors for YOLOv3 are the following: 22,74, 18,155, 35,117, 32,255, 64,132, 60,306, 134,161, 124,352, 301,269.

batch	subdivisions	learning rate	max batches
64	16	0.001	50000

Table 3.5: Main parameters used to train YOLOv3.

The number of subdivisions is higher than for YOLOv2 (see table 3.3) because the network has more convolutional layers in its architecture and so requires more RAM of GPU to be loaded. Consequently, the number of images loaded is lower.

3.4.5 Tiny-YOLOv3

Network

Tiny-YOLOv3 is a light model of YOLOv3. In fact, it has only 3 yolo layers whereas the full model as 5 yolo layers. Thus, it is faster but still performed good detection.

Training configuration

The principal components of the training configuration are presented in table 3.6. Anchors and filters are the same as for YOLOv3.

batch	subdivisions	learning rate	max batches
64	8	0.001	50000

Table 3.6: Main parameters used to train tiny-YOLOv3.

The number of subdivisions is lower than for YOLOv3 (table 3.5) because the network has less convolutional layers in its architecture.

3.4.6 YOLOv3 with a larger data set

That part was added in a second time. Firstly, the results were tested with only one camera view because few videos available was adequate for the labeling algorithm. As results were good the idea to train with more videos was launched. That section presents the new data set developed and named larger data set. It has been noticed with the results of the one camera view that the best model was YOLOv3 and the size of the network had actually a small impact on the running time. Consequently, it has been decided to test that larger data set only with YOLOv3.

Data set

That data set was built with the previous data set from section 3.4.1 and other videos were added. The other videos used are not presenting the same quality as one of the other data set. Indeed, objects appear smaller or their speed is higher and then make it harder to automatically label the images. Four camera views were added which represent around 3021 minutes of video. In total, the training list is composed of 7242 spatio-temporal images and the validation list of 1500 images randomly selected from all the images. Here on the contrary to the other training, the validation set is completely different from the training set because more images were available.

Network and training configuration

The network used is YOLOv3. The principal components of the training configuration are presented in table 3.7. Anchors are different than previous training with YOLOv3 because the bounding boxes of the data set are different. The anchors for that data set are: 9,92, 20,60, 9,266, 20,143, 42,98, 33,298, 82,127, 209,127, 148,338. The number of filters is the same as for YOLOv3. The number of batches was increased to 100000 because of the size of the training set.

batch	subdivisions	learning rate	max batches
64	16	0.001	100000

Table 3.7: Main parameters used to train YOLOv3 with a larger data set.

3.5 Counting objects

To count objects in the video stream, the spatio-temporal images of size 416 pixels are realized continuously. Each time a STI is available the convolutional neural network is used to detect the objects on the STI. The network returns the detected objects with a confidence score superior to 0.1. The number of objects by class is incrementing each time by the number of objects detected in that class until the end of the video. Indeed, the network runs 416 fewer times than with a traditional counting system using the detection algorithm on the video stream. Also, it is possible to increase the length of the STI in order to decrease the number of times the detection process is used.

3.6 Presentation of the graphic interface

To test the algorithm, the folder to use is the folder "YOLO-inference" and inside the python script to run is in the folder "python". Then run the script "inference" with python. Then, the user is asked to choose the desired model as in figure 3.11; the model is loaded. After, the user can choose the length of the spatio-temporal image, figure 3.12. Then, a new window is automatically opened to select the video, figure 3.13. Some instructions are printed in the terminal to guide the user to draw the line on the video window. After selecting the points of the line, the algorithm starts and each spatio-temporal image is shown with the bounding box detected by the model. At the end of the video, the number of objects for each class is printed as well as the detection time and the length of the spatio-temporal images used. At the same time, a .csv file is created with the same previously cited information and save at the video path, s shows figure 3.14. The graphic interface was created with the library EasyGUI [30].

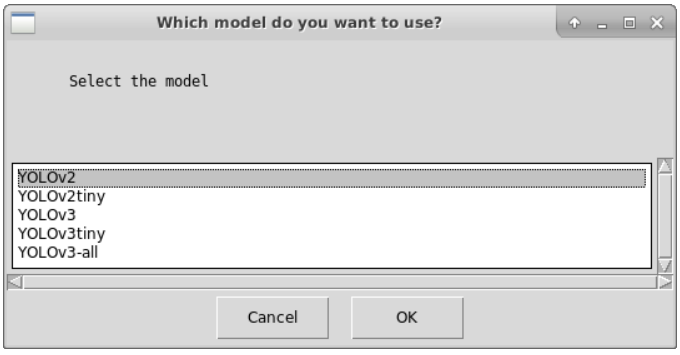


Figure 3.11: Window to chose the desired trained model.

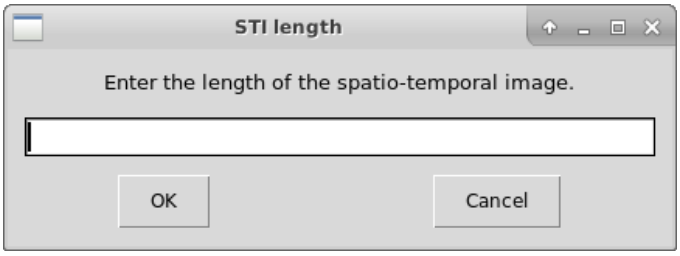


Figure 3.12: Window to choose the length of the spatio-temporal image.

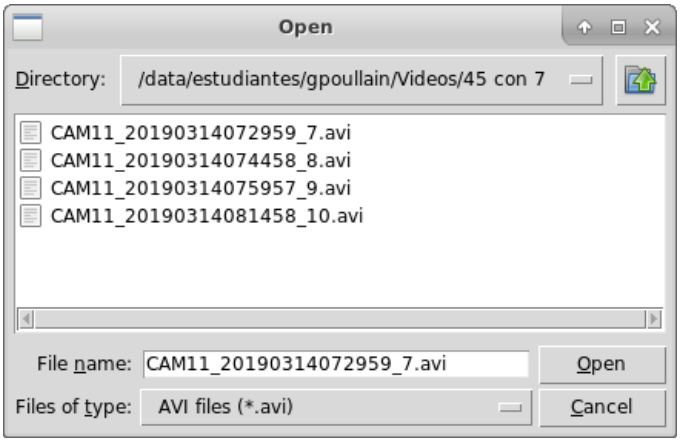


Figure 3.13: Window to choose the video to test the algorithm.

	A	B	C
1	Label	number	
2			
3	particular	183	
4	bus	41	
5	motorcyclist	183	
6	minivan	9	
7	pedestrian	6	
8	truck	6	
9	taxi	183	
10	cyclist	14	
11	tractomula	0	
12			
13	detection time =	0.946465015411s	
14	total time =	256.6010952s	
15	frames/s =	69.9529360389s	
16	SPI length =	600	
17	model :	YOLOv2	
18			
19			

Figure 3.14: Report of the detected objects in the csv file with the number of detected object for each class, the detection time, the total time to run the algorithm, the number of frames per second performed by the algorithm, the length of the spatio-temporal image and the model.

Chapter 4

Results

4.1 Training process

That section presents the results of the training process with the parameters explained in section 3.4.

YOLOv2

The network starts converging after 20000 iterations to the maximum mAP, while the loss error continues to decrease. After 100000 iterations, the mAP reaches 88% and the loss error 0.42 as shows the training graph of figure 4.1.

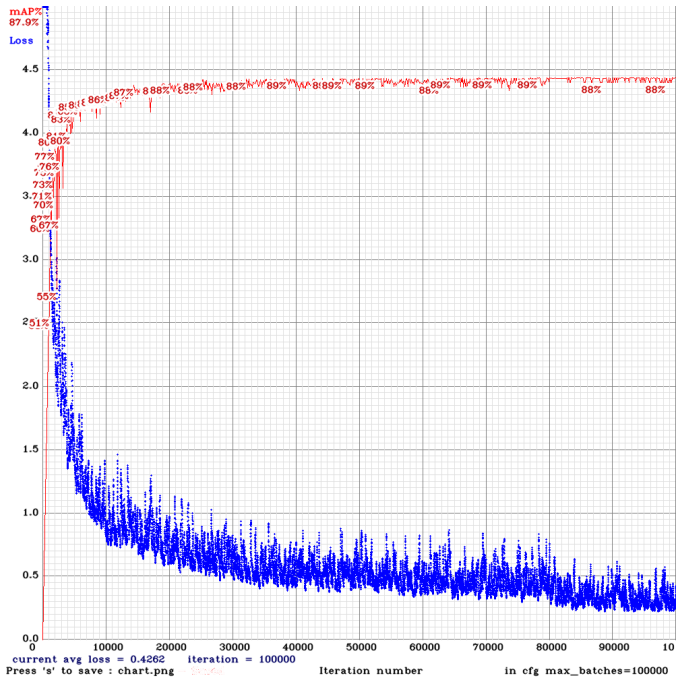


Figure 4.1: Loss error and mAP in function of the number of iterations with YOLOv2.

Tiny-YOLOv2

The network is slower to converge than YOLOv2 and as a larger variance. However, after 50000 iterations the mAP and the loss error seem to have converged. They are reaching respectively 84% and 0.71 as shows figure 4.2.

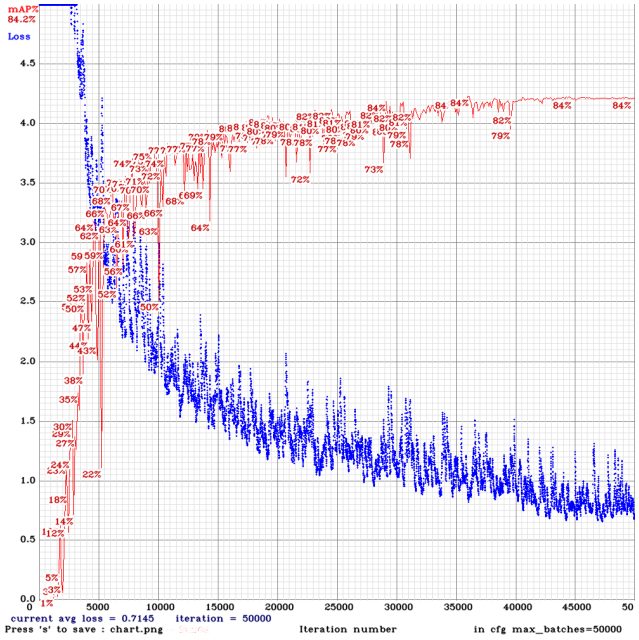


Figure 4.2: Loss error and mAP in function of the number of iterations with tiny-YOLOv2.

YOLOv3

The network was trained with the previously cited parameters. The obtain average loss is of 1,4 and the mAP reaches 89% as shows figure 4.3. The mAP of YOLOv3 is 1% percent higher than the one of YOLOv2, but for the error, the difference is bigger. For YOLOv2, the error converges to 0.4 while only 1.4 for YOLOv3. That difference might come from one of the losses used to calculate the loss error: the classification loss, the localization loss and the confidence loss.

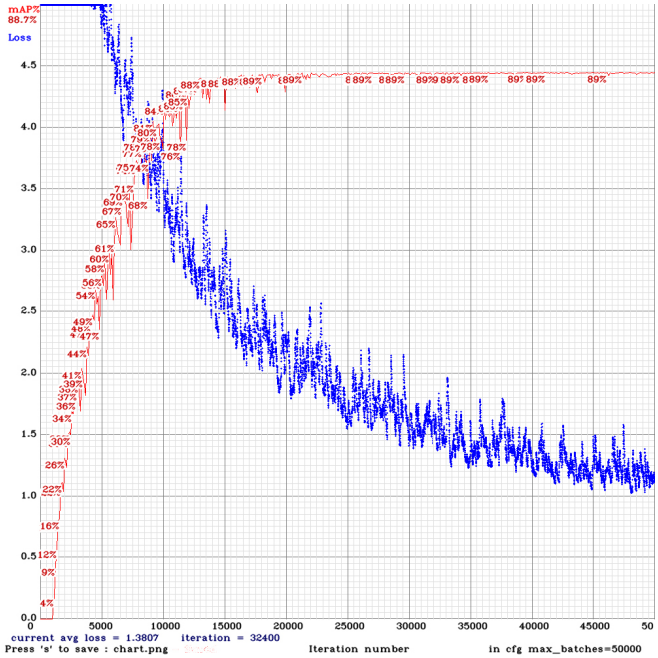


Figure 4.3: Loss error and mAP in function of the number of iterations with YOLOv3.

Tiny-YOLOv3

The logs of the training are presented in figure 4.4. The mAp converges relatively fast but with larger variance. It reaches 88%, so almost the same mAP as YOLOv3. However, the loss error is higher and has also a larger variance. It converges to 2.4.

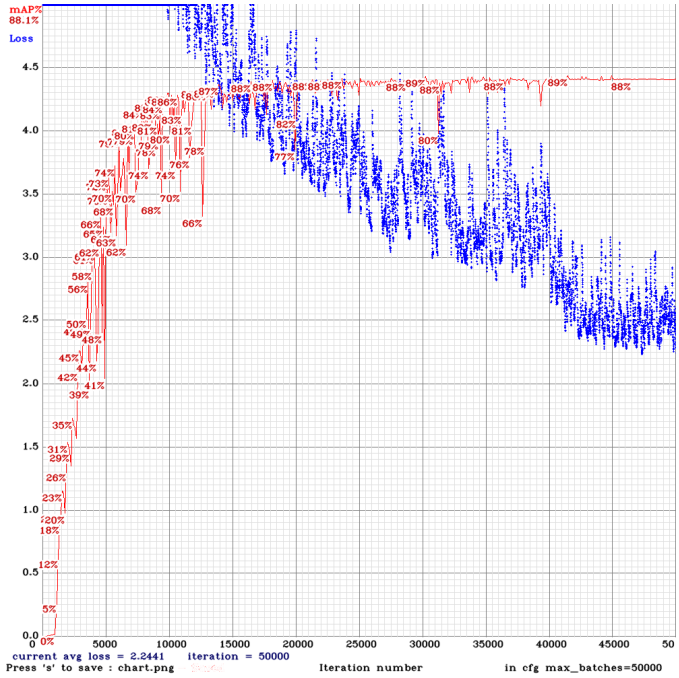


Figure 4.4: Loss error and mAP in function of the number of iterations with tiny-YOLOv3.

YOLOv3 with a larger data set

The logs of the training are presented in figure 4.5. The mAp reaches 99%. The loss error converges until around 1.2. As these results are extremely high, the training and validation were mixed, and the network was trained with the last weight of that model until 130000 iterations in total. The logs of the training were actually very similar, so they are not presented here. These abnormal results may be due to some images where there are no objects and to the approximative labeling algorithm.

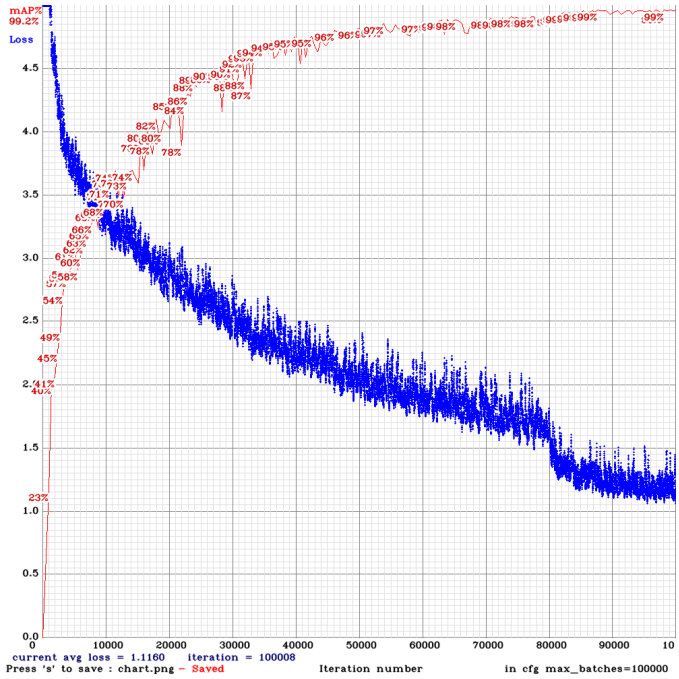


Figure 4.5: Loss error and mAP in function of the number of iterations with YOLOv3 trained with a larger data set.

4.2 Counting process

That section presents the results of each of the models presented in the development part. They are challenged against manual counting and the open source algorithm presented in section 3.2, denoted SDMA for "*Secretaria de Movilidad Algorithm*" in the following part. The same line was drawn for each test.

4.2.1 Results with the main camera used for training

The test was processed on a fifteen minutes video that was used for training. However, a different line was drawn to prevent over-fitting detection. The number of objects for each class is presented in table 4.1. The line total is the number of detected objects whatever the class. The detection time was also calculated. It represents the time the processor is running the function *detect* of Darknet to extract the object from the image. It means that time is started before the function *detect* and stopped in the next line. It does not include the time used to decode the video and run the algorithm. The total detection time is the sum of the time running the function. The test was performed on the computer "cratos" of the university. Thus, a graphic processor: GeForce GTX1080 TI with 11 GB GDDR5X dedicated was used [31]. As the processor is powerful, inference time is relatively low.

Model	YOLOv2	tiny-YOLOv2	YOLOv3	YOLOv3-big	tiny-YOLOv3	SDMA	Manual
Particular	170	206	165	163	172	207	212
Bus	41	51	41	39	37	26	37
Motorcyclist	182	213	171	158	175	204	265
Minivan	9	12	9	9	20	8	9
Pedestrian	4	7	3	3	4	4	0
Truck	6	6	5	6	20	5	5
Taxi	163	176	153	153	160	181	195
Cyclist	14	29	13	13	16	3	11
Trucking rigs	0	0	0	0	0	0	0
Total	589	700	560	544	604	637	794
Det. time (s)	1.3	0.89	1.5	2.1	0.71	490	1800

Table 4.1: Number of object counted for each class with each method, number total of object detected and detection time (Det. time) for a 15 minutes video. The detection time is the running time of the function *detect* of Darknet.

The fastest model for detection time is tiny-YOLOv3, followed by tiny-YOLOv2 and then YOLOv2 and YOLOv3 that runs 2 times slower. Times are given with only two significant numbers because detection time is variable. With a fast look at the table 4.1, it is easy to think that the best model is tiny-YOLOv2 and tiny-YOLOv3 because they have more total detection. However, that number of detections is due to a lot of false positives, (e.g. it detects objects that should not be detected). For example, tiny-YOLOv2 detects almost 3 times more cyclists than in reality and tiny-YOLOv3 detects 2 times more. To tackle the problem the detection threshold should be increased for these two models, but then leads to a lower number of detections. Another remark about the table is that all the models are detecting pedestrians. In fact, no pedestrian is crossing the line, however, they are crossing the road perpendicularly to the street and part of their body is on the line, then the models are detecting there is a pedestrian on the image and are counting it.

To prevent false positives with the tiny model, the threshold was changed. For tiny-YOLOv2 and tiny-YOLOv3 thresholds of respectively 0.5 and 0.2 were used. Then, the table 4.2 is obtained.

The new results are now more coherent in terms of detection for the two fast models. To better understand the results, the table 4.3 presents the effectiveness compared to manual counting for each

Model	YOLOv2	YOLOv2-tiny	YOLOv3	YOLOv3-big	YOLOv3-tiny	SDMA	Manual
Particular	170	159	165	163	168	207	212
Bus	41	38	41	39	37	26	37
Motorcyclist	182	170	171	158	160	204	265
Minivan	9	8	9	9	8	8	9
Pedestrian	4	6	3	3	3	4	0
Truck	6	4	5	6	5	5	5
Taxi	163	151	153	153	155	181	195
Cyclist	14	17	13	13	14	3	11
Trucking rigs	0	0	0	0	0	0	0
Total	589	553	560	544	550	637	794
Time (s)	1.3	0.89	1.5	2.1	0.71	490	1800

Table 4.2: Number of object counted for each class with each method, number total of object detected and detection time (Det. time) for a 15 minutes video. The detection time is the running time of the function *detect* of Darknet.

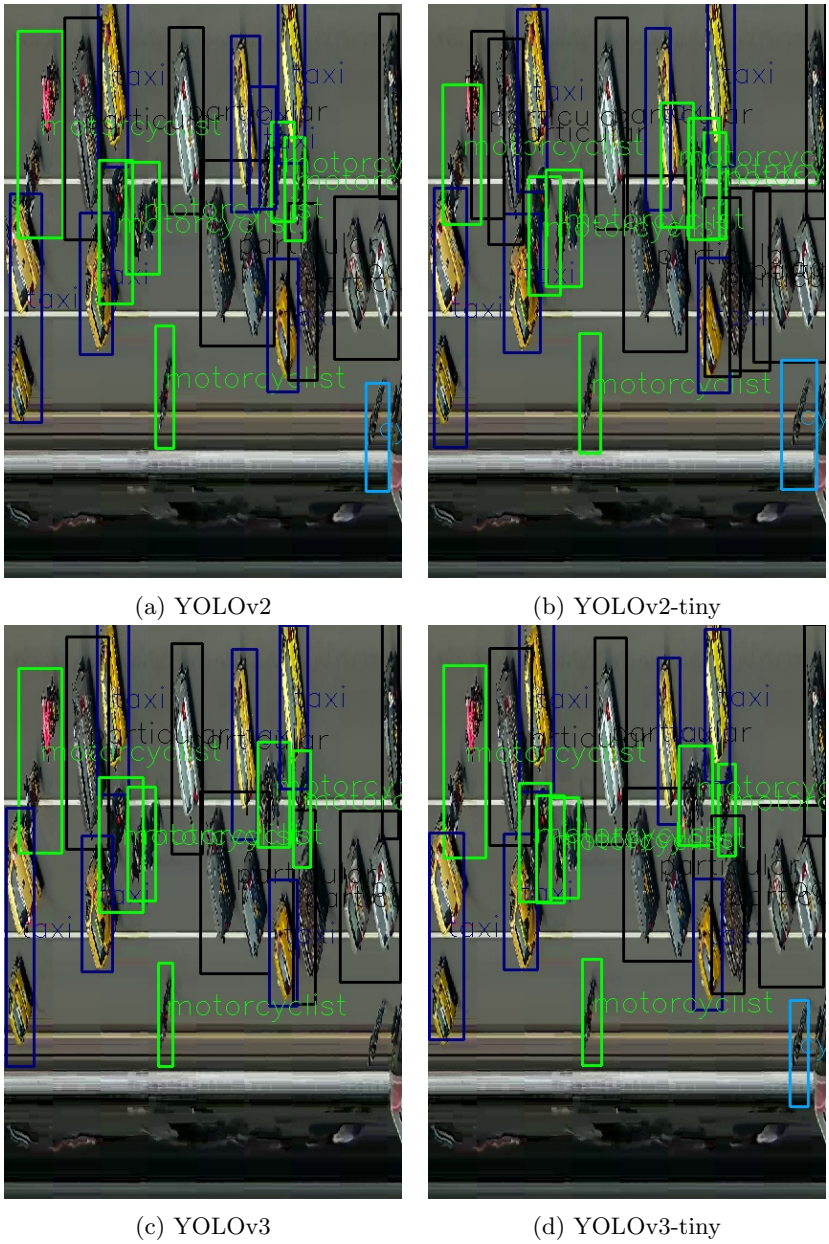
class. The mean was calculated as the sum of error per class divided by the number of classes without taking into account pedestrian and trucking rigs because these two classes are not representative. The same thing was done to calculate the standard deviation (std).

Model	YOLOv2	YOLOv2-tiny	YOLOv3	YOLOv3-big	YOLOv3-tiny	SDMA
Particular	80.19	75	77.83	76.89	79.25	97.64
Bus	89.19	97.3	89.19	94.59	100	70.27
Motorcyclist	68.68	64.15	64.53	59.62	60.38	76.98
Minivan	100	88.89	100	100	88.89	88.89
Pedestrian	-	-	-	-	-	-
Truck	80	80	100	80	100	100
Taxi	83.59	77.44	78.46	78.46	79.49	92.82
Cyclist	72.73	45.55	81.82	81.81	72.73	27.27
Trucking rigs	-	-	-	-	-	-
Total	74.18	69.65	70.53	68.51	69.7	80.23
Mean	79.07	71.38	81.97	81.62	80.12	79.13
std	10.4	16.88	12.84	13.07	14.47	25.28

Table 4.3: Effectiveness compared to manual counting for each class with each method as well as their respective mean and standard deviation (std).

The best models are YOLOv2 and YOLOv3. YOLOv2 is getting better results but some false positive detections were noticed in the images that could explain why it performs better than YOLOv3. They have a relative error of around 20% for the most relevant class that are taxis and particular vehicles. That result is far from the OS algorithm which gets less than 10% error for these two classes. However, the SDM algorithm has a higher error for classes as cyclist and motorcyclist. That is due to a difficult configuration of the testing line for that algorithm. In fact, the line was drawn only for two lanes out of

three. Then as it uses the optical flow, some objects were crossing the line but not the centroid of the bounding box and so these objects were not counted. Another reason is the occlusion of these objects by other bigger. For the spatio-temporal model, the main reason for the low number of detections comes from the data set. Indeed, the labeling algorithm is not perfect, and some objects are not separated by the histogram method and others are approximately labeled. Examples of detection are presented in figure 4.6. The YOLOv3 model train with a larger data set is not obtaining the same results as YOLOv3 trained only with one camera. In particular, the detection of the motorcyclist is extremely lower while for the other classes results are similar. The difference can be explained by the data set that has different views for all the class and then decreases the confidence score of the box. However, at the same time, many more false positives were noticed partly due to some shapes that are representing an object in some videos but that is not with that camera view.



(a) YOLOv2 (b) YOLOv2-tiny

(c) YOLOv3 (d) YOLOv3-tiny

Figure 4.6: Comparison of bounding box for the four models.

For the little models, some false positives are still noticed. In general, for that spatio-temporal accumulation (figure 4.6) all the models are detecting the same objects. As previously explained, the problems of counting are well shown in these images. Models are doing bounding boxes that include two taxis or two particular vehicles and then it leads to an underestimation of the number of objects for the class.

4.2.2 Results of the model trained with a larger data set

The YOLO model trained with different views was also tested on two other camera views. The first view is presented in figure 4.7a is a sidewalk with a bike path. Only pedestrians and cyclists are passing in that 15 minutes video. The second one is presented in figure 4.7b. That video is 32 minutes long and has the specificity to have low traffic.

(a) View of the sidewalk camera.

(b) View of the camera with low traffic.

The results of the algorithm are presented in table 4.4 for the sideways view and in table 4.5 for the low traffic volume video. The model is challenged against the *Secretaria de Movilidad* algorithm (SDMA) and manual counting. The effectiveness is relative to manual counting.

	Number of detection			Effectiveness %	
Model	YOLOv3	OS algo	Manual	YOLOv3	SDMA
Pedestrian	17	26	27	62.96	96.3
Cyclist	4	25	27	14.81	92.6
Total	21	51	54	38.89	94.44
Det. time (s)	1.5	811	1800	-	-
Mean	-	-	-	38.89	94.44
std	-	-	-	24.07	1.85

Table 4.4: Number of object counted for each class with each method, number total of objects detected and detection time (Det. time) for the 15 minutes sidewalk video and the corresponding effectiveness relative to manual counting as well as the corresponding mean and standard deviation (std).

	Number of detection			Effectiveness %	
Model	YOLOv3	OS algo	Manual	YOLOv3	SDMA
Particular	15	15	15	100	100
Bus	12	1	13	92.31	7.69
Motorcyclist	33	30	36	91.67	83.33
Truck	1	0	1	100	-
Taxi	7	10	10	70	100
Cyclist	4	4	5	80	80
Total	72	60	80	90	75
Time (s)	1.6	962	3600	-	-
Mean	-	-	-	89.14	74.34
std	-	-	-	10.82	34.27

Table 4.5: Number of object counted for each class with each method, number total of objects detected and detection time (Det. time) for the 30 minutes video with low traffic and the corresponding effectiveness relative to manual counting as well as the corresponding mean and standard deviation (std).

The position of the camera for the sideways cameras can explain the very bad results obtains by the model using spatio-temporal images, in fact, that position leads to a very small shape in the spatio-temporal image. While the open-source algorithm still performs well.

On the other hand, for the video with low traffic the model performs very well even better than the open source algorithm. It reaches the average detection effectiveness of 90%. That is clearly the best results obtained by a spatio-temporal model in that work. This demonstrates the importance of the video quality to performs detection on spatio-temporal accumulation.

4.3 Complementary time test

In order to get the more relevant execution time of the algorithm, the model was tested on a NVIDIA Jetson TX2 [32]. That system is an embedded AI computing device. It has 8GB of memory and so it is closer to the kind of computer that could be used for traffic monitoring. All the spatio-temporal models were tested as well as the SDM algorithm on a 15 minutes video used to train all models. As in the last section, the detection time is the time the processor runs the function detect of Darknet. The total time, on the other hand, is the total time used to process the whole 15 minutes video, it includes de-codification of the video as well as the saving of the spatio-temporal images and the counting process. To sum up, it is the time since the algorithm starts until it returns the results. The frame per second characteristic is the number of frames processed by the algorithm per second. It is given by dividing the number of frames of the video (here 17860 frames) by the total time.

Model	YOLOv2	YOLOv2-tiny	YOLOv3	YOLOv3-tiny	YOLOv3-big	SDMA
Detection time (s)	13.98	8.079	23.82	7.025	23.22	5195
Total time (s)	443.6	446	457.2	442	457	12147
Frames/s	40.46	40.25	39.26	40.61	39.27	1.478

Table 4.6: Processing time characteristics with an NVIDIA JETSON TX2 [32].

Even with that less powerful computer, the spatio-temporal models are performing detection two times faster than the video. They are outperforming the SDM algorithm by a factor 27. The difference in processing time between these two methods is even bigger with that less powerful computer than with *"cratos"* where the factor was around 6.

Chapter 5

Conclusions and future works

That work proposes to count objects from 9 classes crossing a virtual line in the road. It runs convolutional neural networks on spatio-temporal images to detect, count, and classify objects present in the spatio-temporal accumulation. The best model train for a specific road with right conditions: vehicle speed lower than 50km/h, excellent visibility, and camera position over the road reaches effectiveness of around 75%. If the model is trained as well with other cameras views, the percentage tends to decrease, and the number of false positive detection increases. However, that model still provides 70% effectiveness and even reach 90% effectiveness with low traffic volume, which demonstrates that with a better data set a single model for several camera views could work.

The considerable advantage brought by this method is the gain of processing time. The developed algorithm was able to run faster than the video-frame per second (e.g., the overall time for a 15 minutes video is around 4 minutes with the computer *"cratos"* while the open source algorithm is slower than the video). The processing time could even be improved a little by stopping to plot and save the processed images. Moreover, because the detection time is shorter than the running process of the algorithm (around 1% of the total), a large neural network as YOLOv3 can be used for detection without considerably increasing processing time.

The main drawback of the work is the quality of the data set. The automatic labeling algorithm developed throughout the project allows for fast labeling of a vast number of images. However, the bounding box created for the labeling can be approximate and may encompass few objects of the same class in the same labeling box. Also, the quality of the output of that algorithm is significantly dependent on the vehicle speed and the frame rate of the video. With a higher frame rate than the one available in the videos of the *"Secretaría Distrital de Movilidad"* objects would appear bigger (even at high speed) and more separated which could improve the performance of the algorithm.

Even if the developed labeling algorithm is not perfect, it is still helpful in the process of creating a manually labeled data set of spatio-temporal images for the detection of these nine object-classes. As this work shows, the training of a convolutional neural network such as YOLO on spatio-temporal images makes sense and could provide a high-speed algorithm for traffic monitoring. However, the main task is to create a better data set, accurate and big enough to support many camera views and vehicle velocities. The developed labeling algorithm could help to label that data set because it is often tough for a human to directly label a spatio-temporal accumulation, especially at high speed of the objects. In fact, objects appear very small, and it is for example, almost impossible for a human to differentiate a bike and a motorcycle in a spatio-temporal image. In conclusion, developing that accurate data set could probably provide a better than 90% effectiveness algorithm for counting vehicles and with a processing time six times inferior compared to a detection algorithm using an optical flow or similar to count vehicles.

Bibliography

[1] S. P. Biswas, P. Roy, N. Patra, A. Mukherjee, and N. Dey, "Intelligent traffic monitoring system," in *Proceedings of the Second International Conference on Computer and Communication Technologies*. Springer, 2016, pp. 535–545.

[2] W. S. Ltd, "Sega ends production of d reamcast," 2018, [Accessed: Aug. 14, 2018]. [Online]. Available: http://www.windmill.co.uk/vehicle-sensing.html

[3] N. V. Hung, N. H. Dung, T. M. Hoang, N. T. Dzung *et al.*, "A traffic monitoring system for a mixed traffic flow via road estimation and analysis," in *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*. IEEE, 2016, pp. 375–378.

[4] J. I. Engel, J. Martin, and R. Barco, "A low-complexity vision-based system for real-time traffic monitoring," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1279–1288, 2017.

[5] J. Ren, L. Xin, Y. Chen, and D. Yang, "High-efficient detection of traffic parameters by using two foreground temporal-spatial images," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 1965–1970.

[6] C. Francisco and F. Alejandro, "Low complexity algorithm for the extraction of vehicular traffic variables," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2011, pp. 1021–1026.

[7] W. Balid, H. Tafish, and H. H. Refai, "Intelligent vehicle counting and classification sensor for real-time traffic surveillance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1784–1794, 2018.

[8] A. Forero and F. Calderon, "Vehicle and pedestrian video-tracking with classification based on deep convolutional neural networks," *2019 XXII Symposium on Signal Processing, Images and Artificial Vision (STSIVA), Bucaramanga*, 2019.

[9] R. G. Sigua, *Fundamentals of Traffic Engineering*. UP Press, 2008.

[10] M. K. Kocamaz, J. Gong, and B. R. Pires, "Vision-based counting of pedestrians and cyclists," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016, pp. 1–8.

[11] V. Lempitsky and A. Zisserman, "Learning to count objects in images," in *Advances in Neural Information Processing Systems*, 2010.

[12] R. C. Gonzalez, R. E. Woods *et al.*, "Digital image processing [m]," *Publishing house of electronics industry*, vol. 141, no. 7, 2002.

[13] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous detection and segmentation," in *European Conference on Computer Vision*. Springer, 2014, pp. 297–312.

[14] B. Jähne, *Spatio-temporal image processing: theory and scientific applications*. Springer Science & Business Media, 1993, vol. 751.

[15] Handout, "Olympics day 11 - athletics," 2012, [Accessed: Apr. 7, 2019]. [Online]. Available: http://www.zimbio.com/pictures/XSgerwyo19N/Olympics+Day+11+Athletics/VvVFNXLmq-L

[16] G. Bradski, "The opencv library," *Dr Dobb's J. Software Tools*, vol. 25, pp. 120–125, 2000.

[17] D. G. Raymond Keene, *Man Versus Machine: Kasparov Versus Deep Blue.*

[18] F. Chollet, *Deep learning with python.* Manning Publications Co., 2017.

[19] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[20] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[21] R. Girshick, "Fast r-cnn," in *International Conference on Computer Vision (ICCV)*, 2015.

[22] G. van Rossum and J. de Boer, "Interactively testing remote servers using the python programming language," *CWi Quarterly*, vol. 4, no. 4, pp. 283–303, 1991.

[23] t. f. e. Wikipedia, "Bresenham's line algorithm," 2015, [Accessed: May. 7, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

[24] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001, [Accessed: Apr. 2, 2019]. [Online]. Available: http://www.scipy.org/

[25] G. Ning, "cpunet/scripts/convert.py," 2015, [Accessed: May. 10, 2019]. [Online]. Available: https://github.com/Guanghan/cpuNet/blob/master/scripts/convert.py

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions, corr abs/1409.4842," *URL http://arxiv.org/abs/1409.4842*, 2014.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[28] D. J. MacKay and D. J. Mac Kay, *Information theory, inference and learning algorithms.* Cambridge university press, 2003.

[29] R. Joseph and F. Ali, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[30] A. Zawadzki, H. Jens, R. Lugg, and S. Ferg, "Easygui," 2014, [Accessed: May. 20, 2019]. [Online]. Available: http://easygui.sourceforge.net/

[31] NVIDIA, "Geforce gtx 1080 ti," 2019, [Accessed: May 20, 2019]. [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/

[32] ——, "Jetson tx2," 2019, [Accessed: May 27, 2019]. [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-tx2

Appendix

Python code of the *Trabajo de grado*

https://github.com/GuillaumePou/Trabajo_de_grado