Universidade da Beira Interior
Departamento de Informática

Nº 121 - 2021: *Análise de Tráfego Automóvel a partir de Dispositivos Aéreos Não-Tripulados (UAVs)*

Elaborado por:

Henrique Leitão de Jesus

Orientador:

Professor Doutor Hugo Pedro Proença

August 5, 2022

Acknowledgements

I want to leave a thank you note to everyone who supported me at this stage.
Firstly, I would like to thank my supervisor Hugo Proença for the knowledge shared and whose help was essential to the realization of this project.
I want to thank my parents, Adélia and Víctor, and my sister Luana for supporting and motivating me on this journey.
To my friends and to everyone who helped me in some way, thank you.

Contents

List of Figures

List of Tables

Acrónimos

YOLO You Only Look Once
SSD Single Shot MultiBox Detector
CNN Convolutional Neural Network
SVM Support Vector Machine
RoI Region of Interest
RPN Region Proposal Network
IOU Intersection Over Union
NMS Non-Maximum Suppression
mAP Mean Average Precision
AP Average Precision
MOT Multi-Object Tracking
SORT Simple Online and Realtime Tracking
MTA Measurement-Track Association
HOTA Higher Order Tracking Accuracy
SGD Stochastic Gradient Descent

Chapter

1

Introduction

1.1 Problem Statement

Technological advances have allowed the emergence of revolutionary fields such as Machine Learning and Computer Vision. These fields aim to simulate human intelligence capacity. Humans are great at extracting patterns and interpreting them, making us intelligent and capable of performing complicated tasks. For this reason, we must manage to teach machines to develop this capacity so that they can help us solve problems and execute challenging tasks that exceed human capacity. In fact, this help is increasingly evident in areas such as medicine, science, and the automation of tasks such as autonomous driving. One way of interpreting the world is through vision, and for this reason, the importance of Computer Vision. This field aims to teach the machine to extract useful information from videos and images and then process it.

With the emergence of convolutional networks, analyzing and processing imagery data became more and more accessible. These mechanisms allow the creation of models capable of recognizing and locating objects and people in images. In this way, it is possible to improve surveillance or video analysis systems and make them autonomous without the need for a human to interpret these data.

Unmanned aerial vehicles with access to a camera have become a beneficial aid for several entities. Since they can fly at a certain height, they allow the capture of a vast area. For this reason, they can be used to gather information from an area and patrol a place in order to be analyzed. There are more and more vehicles circulating in urban areas, which makes it a complicated task for those who supervise traffic.

This project aims to use machine learning and computer vision mechanisms to process videos captured through unmanned aerial vehicles in order to analyze urban traffic.

1.2 Motivation and Objectives

The number of vehicles in urban environments has increased over time. In order to collect traffic information, several surveillance cameras are placed on the highways. This implies a high number of equipment needed to extract information in extensive areas. With the emergence of ML and CV fields, it is possible to create mechanisms capable of analyzing and extracting information about traffic through videos captured by drones. Thus, a single device can patrol an urban area and automatically interpret it.

Our main objective is to develop a framework capable of extracting traffic information to produce statistics through UAV videos. These statistics allow us to characterize the traffic by creating traffic density maps, counting vehicles on the road, and estimating the vehicle's speed. Several models and techniques responsible for vehicle detection, tracking, and speed estimation will be analyzed to accomplish this objective.

1.3 Project Organization

The report of this project is organized as follows:

1. **Chapter 2**: This chapter presents the models' state-of-art parts of the related works. It is divided into object detection models and multiple object tracking models. In addition, it also contains two chapters related to the metrics used in detection and tracking, respectively.

2. **Chapter 3**: Here is where we propose the method to develop this work. Here we can find a detailed explanation of each implemented module and its rationale. Also, this is where we list the materials used in this project.

3. **Chapter 4**: In this chapter, we highlight the results obtained in each implemented module and present its weaknesses.

4. **Chapter 5**: Finally, in this chapter, we conclude the project and outline the future work.

Chapter

2

State-of-Art

In this chapter, we review the state-of-art models that are part of the works related to this project. It contains two sections dedicated to object detection and tracking of multiple objects and two other sections dedicated to the metrics used respectively by these two types of models. Each section briefly explains the technologies and techniques addressed.

2.1 Object detection

Object detection is seen as one of the most fundamental problems of computer vision. This task aims to build computational models capable of locating objects in digital images as well as their classification (dog, human, car). The rapid development of deep learning technologies allowed the emergence of several models referring to this task. These models are grouped into two types: the multiple-stage and the single-stage models.

Multi-stage models prioritizes detection accuracy. In these, a first stage is responsible for extracting regions from objects, and a second stage is used to classify and improve the object's location in the image. This method works well, but on the other hand, it becomes slower as it requires performing the detection and classification process on various occasions. The most popular models are R-CNN, Fast R-CNN, and Faster R-CNN.

Single-stage models prioritizes inference speed. Typically, these methods propose several bounding boxes of different scales in the image to reach a trade-off between speed and accuracy. This technique relies on a convolutional network to recognize and locate the object in forward propagation. Thus, these types of models are usually faster than the two-stage models, but on the other hand, they have less accuracy. The most popular models are the

YOLO and the SSD.

2.1.1 R-CNN

R-CNN [1] was proposed by Ross Girshick in 2014. It combines region proposals with Convolutional Neural Networks and aims to improve the quality of candidate bounding boxes and take a deep architecture to extract high-level features. In the R-CNN model 2.1, an image is received as input(1), then the process is divided into three stages: A Region proposal generation (2), where it is used a selective search algorithm to generate about 2000 region proposals for each image. Feature extraction (3), the region proposals are cropped and resized so that a large convolutional neural network can extract a 4096 dimensional feature vector. Classification and localization (4), the feature vector is run through a collection of linear support vector machines (Support Vector Machine (SVM)), where each SVM is responsible for classifying a single object class and indicating the likelihood that the region proposal contains that class. Finally, these regions are filtered with non-maximum suppression and adjusted with bounding box regression.

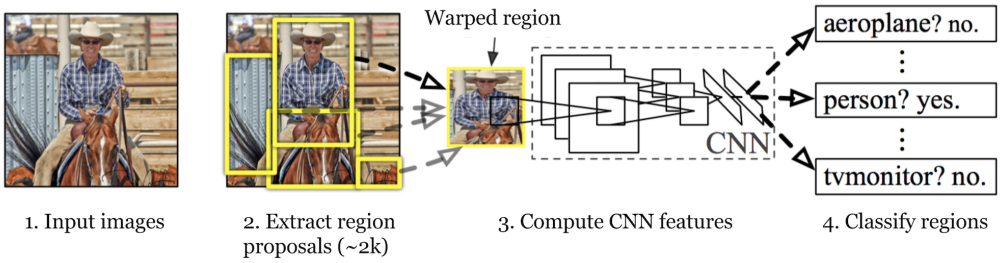


Figure 2.1: R-CNN object detection workflow (from [1]).

R-CNN drawbacks:.

- It takes much time to generate the region proposals since the model depends on the selective search algorithm.
- These regions are fed independently to the Convolutional Neural Network (CNN) for feature extraction, making it slow and impossible to run in real-time.
- Each stage is an independent component, so the model cannot be trained end-to-end.
- Processing a small training set with deep networks can be time-consuming.

2.1.2 Fast R-CNN

Fast R-CNN model [2] was later created, introducing several improvements and innovations in the training and testing speed while also increasing the accuracy rate. This model processes the whole image with convolutional and max-pooling layers to produce a convolutional feature map. Then, a Region of Interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map for each object proposal region. Each feature vector is fed into a sequence of fully connected layers and finally branched into two sibling output layers: one that applies softmax to estimate the probabilities of the K object classes plus a "background" class and another layer that outputs four real-valued numbers for each of the K object classes, these correspond to each object bounding boxes. This RoI pooling layer allows Fast R-CNN to share computation rather than doing calculations for each proposal independently, making it faster than R-CNN. Figure 2.2 illustrates the Fast R-CNN architecture.

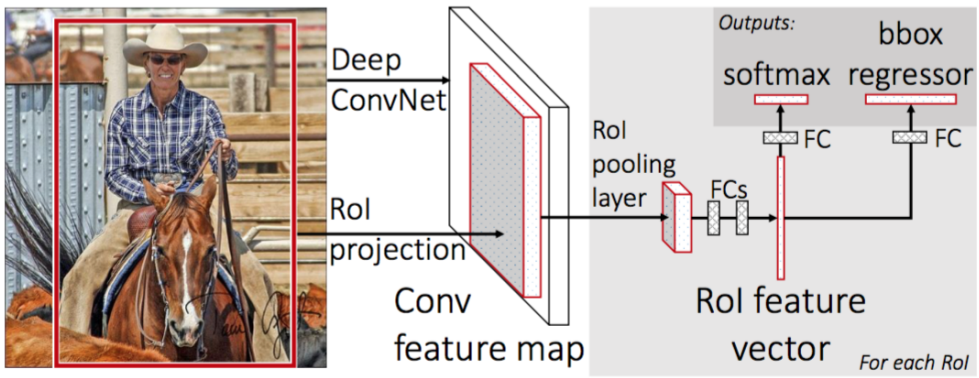


Figure 2.2: Fast R-CNN architecture (from [2]).

Fast R-CNN drawback: just like R-CNN, it also depends on the time-consuming Selective Search algorithm to generate region proposals.

2.1.3 Faster R-CNN

Faster R-CNN [3], proposed by Ren et al. in 2015, is an improved version of R-CNN and Fast R-CNN. The difference between these is that Faster R-CNN uses a Region Proposal Network (RPN) instead of using the exhaustive selective search algorithm to generate the region proposals. The RPN is a fully convolutional network that generates proposals with various scales and aspect ratios. It also implements the neural network terminology with attention this

module tells the Fast R-CNN module where to look. In this way, it is possible to calculate the features of the entire image at once and therefore does not involve repeated calculations, which significantly improves the detection speed of Faster R-CNN. This paper introduced the concept of anchor boxes rather than using pyramids of images or pyramids of filters. An anchor box is a bounding box with a specific scale and aspect ratio that will be used to regress the final predicted bounding box for each object. With multiple reference anchor boxes, a single region has multiple scales and aspect ratios, which allows using the computed convolutional features on a single-scale image. This design permits sharing features without extra cost for addressing scales. Summing up, Faster R-CNN's architecture consists of two modules: the RPN for generating region proposals faster than selective search and the Fast R-CNN for detecting objects in the proposed regions. Figure 2.3 illustrates the RPN module in Faster R-CNN.

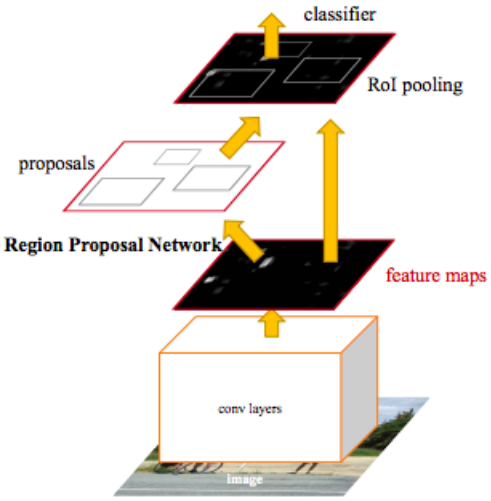


Figure 2.3: The RPN module in Faster R-CNN serves as the attention mechanism (from [3] paper).

Faster R-CNN drawbacks:

- RPN is not good at dealing with objects with huge shapes or scales.
- The multiple samples from a single image may be correlated, so it may take much time to reach the convergence.

2.1.4 YOLO

You Only Look Once (YOLO) [4], proposed by Joseph Redmon in 2016. It is a single regression-based method neural network able to classify and predict bounding boxes for detected objects in one evaluation with high performance and accuracy rate, being able to run in real-time. YOLO means You Only Look Once since it looks at the entire image simultaneously, which makes it different from R-CNNs, which look at different parts of the image separately. The main idea of YOLO is shown in Figure X. According to the paper [4], this model divides the input image into an S x S grid, and if the center of an object is inside a grid cell, that grid cell will be responsible for detecting the object. Each cell predicts B bounding boxes, with their confidence level between 0 and 1. This confidence scores are defined as $Pr(Object) * IOU_{\text{pred}}^{\text{truth}}$: $Pr(Object)$ indicates the probability of existing objects. $IOU_{\text{pred}}^{\text{truth}}$ it is a confidence that represents the Intersection Over Union (IOU) between the predicted and the ground truth box. IOU is calculated through the division between the intersection area of the predicted and ground truth box by the union of the same boxes. Each grid cell also predicts C conditional class probabilities, $Pr(Class_i|object)$. These are conditioned on the grid cell containing an object. The class-specific confidence scores for each box are achieved by multiplying the individual box confidence predictions with the conditional class probabilities as follows:

$$Pr(Object) * IOU_{\text{pred}}^{\text{truth}} * Pr(Class_i|object) = Pr(Class_i) * IOU_{\text{pred}}^{\text{truth}} \quad (2.1)$$

Each bounding box consists of 5 predictions [4]: *x, y, w, h*, and the confidence: *(x,y)* are coordinates that represent the center of the box relative to the bounds of the grid cell. *(w, h)* are the width and height predicted relative to the whole image. Thus, the predictions are encoded as an S x S x (B*5 + C) tensor.

The YOLO architecture in 2.4 is divided into three components: the head, the neck and backbone. The backbone is made of 24 convolutional layers responsible to extract the features from the image. The neck consists of 2 fully connected layers responsible to predict the output probabilities and coordinates. Some convolutional layers alternate 1×1 reduction layers to reduce the features space from preceding layers. The head is the final output layer of the network, which outputs an S x S x (B*5 + C) tensor, as previously mentioned.

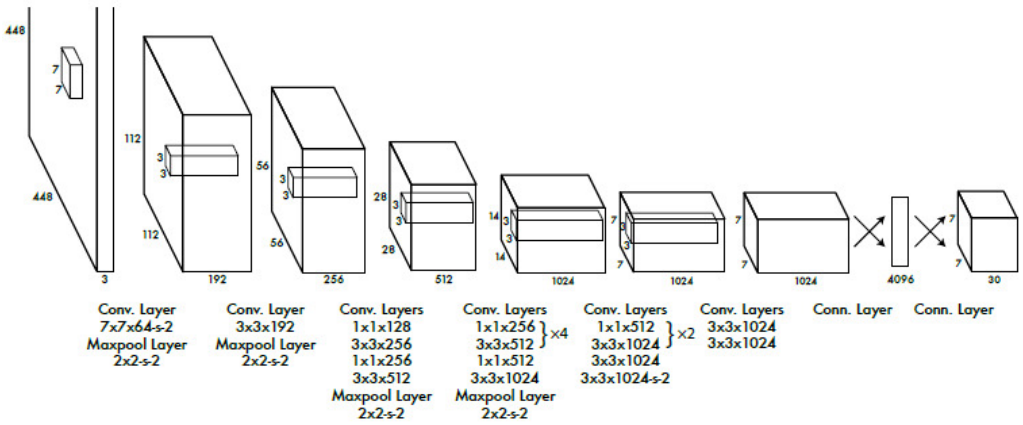


Figure 2.4: The YOLO architecture. It has 24 convolutional layers followed by 2 fully connected layers (from [4]).

The loss function optimized in the training is the following:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.2}
$$

where:

- (x_i, y_i) denotes the location of the centroid of the anchor box in grid cell i.
- (w_i, h_i) is the width and height of the anchor box.
- C_i represents the confidence score of whether there is an object or not.
- $\mathbb{1}_{i}^{obj}$ indicates the existence of objects. It is 1 if the object exists, otherwise is 0.
- $\mathbb{1}_{ij}^{obj}$ denotes that the prediction is conducted by the jth bounding box predictor and $\mathbb{1}_{ij}^{noobj}$ is its complement.

- λ_{coord} increases the weight for the loss in the boundary box coordinates and λ_{noobj} weights down the loss when detecting background.
- $p_i(c)$ is the classification loss.
- hat ($\hat{}$) variables represent the respective predicted values.

This loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability in 2.1). It also penalizes bounding box coordinate error if that predictor is responsible for the ground truth box, i.e., has the highest IOU.

In Inference, Non-Maximum Suppression (NMS) is used to fix the problem of multiple detections from the same object. NMS selects the box with the highest objectiveness score, then compares this box's IOU with other boxes, and finally removes the bounding boxes with overlap above some given threshold.

YOLO drawbacks: it imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class, this causes struggles when there are too many objects close to each other. YOLO has difficulty generalizing objects in new or unusual aspect ratios or configurations since the model learns to predict bounding boxes from data.

2.1.5 SSD

Single Shot MultiBox Detector (SSD), proposed by Liu et al. [5], aimed to solve the YOLO limitations. The model was inspired by anchors adopted in MultiBox [12], RPN, and multiscale representation, and like YOLO, it is a single neural network. As mentioned before, the most salient limitation of YOLO is the problem is the strong spatial constraints. SSD uses default anchor boxes with different aspect ratios and scaled instead of fixed grids to solve this problem. The network mixes predictions from multiple feature maps with different resolutions to handle objects of various sizes.

SSD architecture adopts an algorithm for detecting various classes of objects in an image, providing scores associated with the presence of any category of objects. This model is also suitable for running in real-time applications as it, like YOLO, re-evaluates the bounding boxes only once. The SSD architecture in 2.5 uses a VGG-16 network [13] as the backbone and replaces the last two fully connected layers with convolutional layers. Then, four more convolutional layers are responsible for predicting the offsets to default boxes with different scales and aspect ratios and their associated confidences. The network is trained with a weighted sum of localization loss and confidence loss. It uses NMS on multiscale refined bounding boxes. [14]

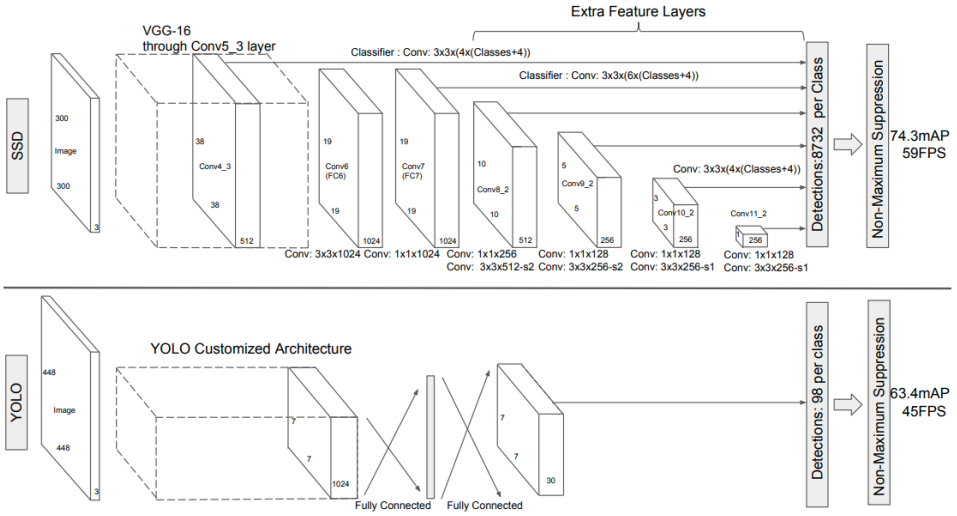


Figure 2.5: Comparison between SSD and YOLO (from [5]). The SSD model outperforms the YOLO model with more fps and Mean Average Precision (mAP).

2.1.6 YOLOv2

YOLOv2 [15] was a joint endeavor by Joseph Redmon and Ali Farhadi, and it is the second version of the YOLO, bringing improvements to the accuracy and performance. The accuracy improvements referred to in the paper are:

- **Batch normalization** - Doing this on all convolutional layers allows for regularizing the model and increasing the mAP. This technique removes the need for dropouts without overfitting.

- **High-Resolution Classifier** - The original YOLO trains the classifier network at 224 x 224 and increases the resolution to 448 for detection. This means the network has to switch to learning object detection and adjust to the new input resolution. In YOLOv2, the classification network is tuned at full 448 x 448 resolution for 10 epochs. This gives the network time to adjust its filters to work better on higher resolution input.

- **Convolutional With Anchor Boxes** - YOLO predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. The fully connected layers were removed from YOLO, and anchor boxes were used to predict bounding boxes. One pooling layer was eliminated to make the output of the network's convolutional layers higher resolution. Also, the network was shrunk to

operate on 416 input images instead of 448 x 448. Anchor boxes decreased mAP slightly from 69.5 to 69.2, but the recall improved from 81% to 88%. Even if the accuracy is slightly decreased, it increases the chances of detecting all the ground-truth objects.

- **Dimension Clusters** - Instead of choosing priors by hand, we run k-means clustering on the training set bounding boxes to automatically find reasonable priors.
- **Direct location prediction** - YOLOv2 makes predictions on the offsets of the anchor boxes.
- **Fine-Grained Features** - It turns the 26 x 26 x 512 feature map into a 13 x 13 x 2048 feature map, which can be concatenated with the original features.
- **Multi-Scale Training** - Once the fully connected layers are removed, YOLOv2 can take images of different sizes. The network is changed every few iterations instead of fixing the input image size. Since the model downsamples by a factor of 32, it pulls from the following multiples of 32: {320, 352,...,608}. Thus, the smallest option is 320 x 320, and the largest is 608 x 608.

2.1.7 YOLOv3

YOLOv3 [6] released in 2018 by Joseph Redmon and Ali Farhadi to improve YOLOv2. The main improvements referred to in the paper are:

- **Bounding Box Prediction** - YOLOv3 predicts an objectness score for each bounding box using logistic regression. Suppose the anchor overlaps a ground truth object more than the other. In that case, the corresponding objectness score should be 1, for the others with overlap more significant than a predefined threshold of 0.5 are ignored. If an anchor is not assigned, it incurs no classification and localization loss, just confidence loss on objectness.
- **Class Prediction** - YOLOv3 uses independent logistic classifiers instead of softmax. During training, it is used binary cross-entropy loss for the class predictions.
- **Predictions Across Scales** - YOLOv3 predicts boxes at 3 different scales. So the output tensor is N x N x [3*(4+1+80)] for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

- **Feature Extractor** - YOLOv3 uses the Darknet-53 2.6, which is a 53 layer feature extractor. This CNN is a hybrid approach between the YOLOv2 network (Darknet-19) and the residual network. It shows to have more 211 BFLOP/s (billion floating-point operations per second) than Darknet-19 and a slight increase in accuracy.

	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 2.6: Darknet53 architecture (from [6]).

2.1.8 YOLOv4

YOLOv4 [16] was released by Alexey Bochoknovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao in 2020 and presents several improvements to the predecessor YOLOv3 model. It increased the average precision and frames per second by 10% and 12%, respectively, compared with YOLOv3. The backbone was improved. In YOLOv4, the CSPDarknet-53 is used, which adds CSP (Cross Stage Partial) connections [17] to the previous Darknet-53. In the paper, the authors refer to two terms when talking about the improvements are **the bag of freebies** which are methods that can make the object detector receive better accuracy without increasing the inference cost, and **bag of specials** which are plugin modules and post-processing methods that only increase the in-

ference cost by a small amount but can significantly improve the accuracy of object detection. The **bag of freebies** methods are:

- **Data augmentation**: this can be categorized as photometric distortions (brightness, contrast, hue) and geometric distortions (random scaling, cropping, rotation). Doing this can lead to decent accuracy improvements. Ways of object occlusion are also referred to, such as Hide-and-Seek and Grid Mask, which select some regions on the image and replace them with zeros. Finally, the authors also proposed using MixUP and CutMix.

- **Semantic Distribution Bias**: Other methods are solving the problem that semantic distribution in the dataset may have bias. In order to solve the problem, the focal loss was proposed. This method can be applied in one-stage detectors instead of other solutions like hard negative example mining or online hard example mining that belong just to a two-stage object detector.

- **Objective Function of BBox Regression**: Instead of Mean Square Error (MSE) to directly perform regression on the bounding box, the authors refer to some functions to regress these bounding boxes. Are they: IOU loss (showed before), GIoU loss, which includes the shape and orientation of the object in addition to the coverage area, DIoU loss, which considers the distance of the center of an object and CIoU loss, which considers the overlapping area, the distance between center points, and the aspect ration.

The **bag of specials** methods are:

- **Spatial Attention Module(SAM)**: It is an attention module that only needs to play 0.1% extra calculation, and it can improve ResNet50-SE 0.5% top-1 accuracy on the ImageNet image classification task. Furthermore, it does not affect the speed inference on the GPU.

- **Mish/Swish Activation function**: According to the authors, a good activation function can make the gradient more efficiently propagated. At the same time, it will not cause too much extra computational cost. ReLU is one of the most famous activation functions since it is zero cost. Although, when differentiating this function for negative values, it turns 0. This is called dying ReLU. Both Mish and Swish functions can cause this problem. Even though they are high cost compared with ReLU, they can significantly increase the accuracy.

- **DIoU NMS**: As previously mentioned, NMS filters the multi-sample boxes in the same object, but this can be a problem if two close distinct boxes are suppressed into one. The paper suggests DIoU solve this problem. This method adds the information of the center point distance to the bounding box screening process based on NMS.

The bag of specials has also 2 techniques, the first is in the backbone which uses the mish activation and cross-stage partial connections, the second is in detection, which uses the SPP-block, the Spatial Attention Module (SAM) block, and others.

2.1.9 YOLOv5

YOLOv5 [9] was released shortly after the YOLOv4 in 2020 by Glenn Jocher using the PyTorch framework. The main difference is that YOLOv5 uses Focus Structure with CSPdarknet-53 as a backbone. The Focus layer was first introduced in YOLOv5, replacing the first three layers in the YOLOv3 algorithm. According to the official implementation on GitHub [18], the purpose of this layer is to reduce layers, reduce parameters, reduce FLOPS, reduce CUDA memory, and increase forward and backward speed while minimally impacting mAP.

2.1.10 TPH-YOLOv5

TPH-YOLOv5 [7] aims to improve object detection performance on drone-captures images. The model uses a self-attention mechanism by replacing the original prediction heads with Transformer Prediction Heads (TPH) to detect different-scale objects. This transformer encoder blocks increase the ability to extract different local information. According to the paper, this technique shows better performance on occluded objects on the VisDrone2021 dataset [19]. To find the attention region in images, the model uses a convolutional block attention model (CBAM [20]). CBAM takes a feature map, sequentially infers the attention map, and then multiplies it with the input feature map to perform adaptive feature refinement. This module will help YOLOv5 to focus on useful target objects. Figure 2.7 illustrates the CBAM and Transformer Encoder modules.

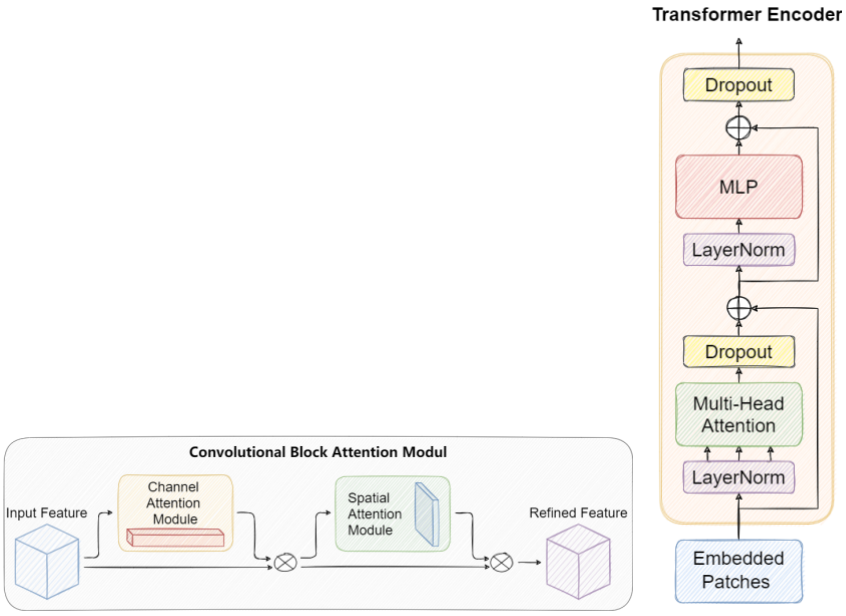


Figure 2.7: The CBAM module is on the left, and the Transformer Encoder is on the right (both images from [7]).

2.2 Object Detection Metrics

We need specific metrics to measure the accuracy of the object detectors mentioned above. These metrics allow measuring the relationship between the prediction and the ground truth of the data. The algorithms described in this section are the Intersection Over Union, Recall-Precision Curve, and Mean Average Precision.

2.2.1 Intersection Over Union

Intersection Over Union measures the ratio between the intersection of the ground truth and predicted bounding boxes area with the union of the same as seen in 2.8.

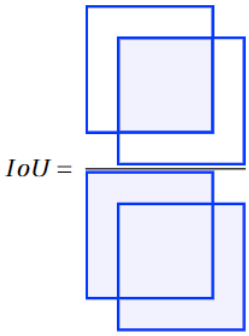


Figure 2.8: Intersection Over Union formula.

The predicted bounding box is discarded if this result does not reach a certain threshold.

2.2.2 Precision-Recall Curve

According to [21], the precision-recall curve shows the tradeoff between precision and recall for different thresholds. The precision and recall are used to measure the performance of a model to classify objects. Precision and recall are determined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

where:

- *TP* (True Positives) - When the model correctly indicates the presence of the class in the data.
- *FP* (False Positives) - When the model incorrectly indicates the presence of the class in the data.
- *FN* (False Negatives) - When the model incorrectly indicates the lack of the class in the data

High precision indicates a low rate of false positives, while a high recall means a low rate of false negatives. The greater the area under the precision-recall curve, the higher the accuracy and recall of the model. It indicates that the classifier is accurate and that most results are positive.

2.2.3 Average Precision

We can compute the Average Precision (AP) from the precision-recall curve. It summarizes the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight [21], as follows:

$$\text{AP} = \sum_{n} (R_n - R_{n-1}) P_n \tag{2.5}$$

where P_n and R_n are the precision and recall at the nth threshold. The AP is just calculated for each class.

2.2.4 Mean Average Precision

The Mean Average Precision is the mean of the AP calculated for each class in the model. This metric is state-of-art to compare the performance of object detection models. This metric is the most used to compare the performance of object detection models and can be presented as mAP_{50}, mAP_{75} and mAP_{95}. The numbers 50, 75, and 95 indicate that the value of the IOU threshold was 0.50, 0.75, and 0.95, with mAP by default being the average of these three metrics.

2.3 Multi-Object Tracking

Multi-Object Tracking (MOT) is a data association problem aiming to associate detections across a sequence of images. A MOT model is responsible for localizing and identifying multi objects. It depends on an object detector model and some algorithms to predict the localization of the object through the sequence of frames. This section will discuss Simple Online and Realtime Tracking (SORT) [22] and then DeepSort [23] since this last one is an improved version of SORT and the state-of-art MOT model.

2.3.1 SORT

Simple Online and Realtime Tracking [22] is a simple MOT model with effective algorithms consisting of four core components: detection, estimation, association, and creation and deletion of track identities.

2.3.1.1 Detection

SORT tracks bounding boxes across the sequence of images, so it needs a detector to extract these boxes like the ones referred to in 2.1

2.3.1.2 Estimation

A constant velocity linear model propagates the bounding boxes through the frames, independent of other objects and camera motion. The state of each target is represented as follows:

$$\mathbf{x} = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T \tag{2.6}$$

where u and v represent the horizontal and vertical pixel location of the center of the target, s represent the scale (area), and r the aspect ratio of the target's bounding box [22]. When the detection is associated with a target, the detection bounding box is used to update the target state. On the other hand, if the detection cannot be associated with the target, for example, occlusion, then the state is predicted using the linear velocity model. These velocity components are optimally solved with the Kalman Filter [24].

2.3.1.3 Association

The bounding boxes' locations are estimated using the estimation techniques described above. Then the predicted bounding boxes are matched with the detections ones. The module computes the cost matrix as the IOU distance between each detected bounding box and all predicted bounding boxes. The assignment is solved optimally using the Hungarian algorithm. This technique is used to associate the bounding boxes in the cost matrix.

2.3.1.4 Creation and Deletion of Track

SORT [22] uses a mechanism to manage object ids as they enter and leave images. A tracker is created if the detection overlap is smaller than a given IOU_{min} threshold. The detection bounding boxes are used to initialize the Kalman Filter state. Their velocity is set to zero, and the covariance is set to high to signify uncertainty in the state. If a tracker stops receiving updates, they are deleted after T_{Lost} frames [22].

2.3.2 DeepSort

DeepSort [23] aims to improve SORT by adding a deep association metric. SORT has critical drawbacks since it often switches identities and has difficulties dealing with occlusions. DeepSort implements some techniques to solve this problem. It uses a convolutional neural network already pre-trained for a classification problem, such as people recognition. The CNN classification layers are removed, so it outputs just the feature vector called the appearance

descriptor by authors. Then nearest neighbor queries are used in the appearance descriptor to establish the Measurement-Track Association (MTA). MTA is the process of establishing a relationship between a measurement and an existing track. Instead of using the Euclidean distance in the MTA, it is used the Mahalanobis distance. According to the authors, these improvements reduce the number of identity switches by 45%, achieving overall competitive performance at a high frame rate [22].

2.4 Multi-Object Tracking Metrics

Just as we need metrics to evaluate object detectors, we also need these to evaluate the performance of MOT models. These models are more complex to evaluate. For this reason, there are several different evaluation algorithms in order to make this evaluation fairer. The most used models for this purpose are the Multiple Object Tracking Accuracy (MOTA) [25] and the Higher Order Tracking Accuracy (HOTA) [8]. This chapter will cover HOTA as it is a MOT metrics state-of-art.

2.4.1 HOTA

Higher Order Tracking Accuracy [8, 26] is the state-of-art of the MOT metrics. While other metrics like MOTA overemphasize detection and association as seen in 2.9, HOTA measures both and combines them in a balanced way. It also measures the localization accuracy of tracking, unlike MOTA. HOTA comprises three sub-metrics: Detection, Association, and Localization. Then these three IOU scores are combined into a single score (HOTA score).

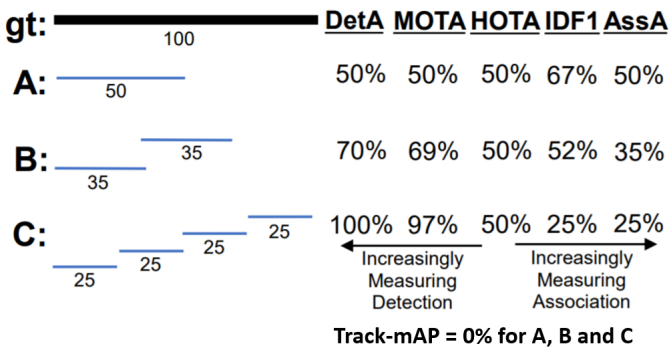


Figure 2.9: This image (from [8]) shows how other MOT metrics like MOTA and IDF1 overemphasize the detection and association. In contrast, HOTA presents a balanced evaluation.

2.4.1.1 Localization

The localization component measures the spatial alignment between one predicted detection and one ground-truth detection [26]. The metric that measures localization is Localization IOU (Loc-IOU). Since it is an IOU, it is calculated in the same way as **??**, i.e., the ratio of the intersection between the predicted and ground truth detection with its union. The Loc-IOU is then averaged over all pairs of matching predicted and ground-truth detections to determine the Localization Accuracy (LocA) [26].

2.4.1.2 Detection

The detection component measures the alignment between the set of all predicted detections and the set of all ground-truth detections [26]. The metric that measures the detection is the Detection IOU (Det-IOU). Both Det-IOU and Detection Accuracy (DetA) are calculated in the same way, i.e., using the count of True Positives (TP), False Negatives (FN), and False Positives (FP) over the whole dataset [26], as follows:

$$\text{DetA} = \text{Det-IOU} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}| + |\text{FP}|} \tag{2.7}$$

2.4.1.3 Association

The association component measures how well a tracker associates detections through a sequence of frames into the same identities (IDs). The metric that measures the association is the Association-IOU (Ass-IOU). It uses the Hungarian algorithm to match the predicted detection with the ground-truth detection. The Ass-IOU is calculated similarly as 2.7 but uses the TP, FP, and FN of class associations (respective TPA(c), FPA(c) and FNA(c)). The Association Accuracy is determined by calculating the Ass-IOU of all pairs of matching predicted, and ground-truth detections [26], as follows:

$$\text{AssA} = \frac{1}{|\text{TP}|} \sum_{c \in \text{TP}} \text{Ass-IOU(c)} = \frac{1}{|\text{TP}|} \sum_{c \in \text{TP}} \frac{|\text{TPA(c)}|}{|\text{TPA(c)}| + |\text{FNA(c)}| + |\text{FPA(c)}|} \tag{2.8}$$

2.4.1.4 HOTA score

The HOTA score is the single metric that measures the overall performance of a MOT model. To calculate this score, we need to integrate the geometric mean of the detection and the association score for each threshold value α. Then the localization accuracy is included in the final score [26], as follows:

$$\text{HOTA}_\alpha = \sqrt{\text{DetA}_\alpha \cdot \text{AssA}_\alpha}$$
$$\text{HOTA} = \int_{0<\alpha\leq 1} \text{HOTA}_\alpha \approx \frac{1}{19} \sum_{\alpha=0.05} \text{HOTA}_\alpha \tag{2.9}$$

where DetA_α and AssA_α are the Detection and Association accuracy for different α thresholds, since they both depend on the Loc-IOU threshold α values. The HOTA_α represents the HOTA score for the respective α threshold.

Chapter

3

Proposed Method

This chapter describes the proposed method to create a framework capable of automatically analyzing car traffic from Unmanned Aerial Vehicles (UAVs) videos. The proposed solution consists of three core modules. Two of these are responsible for extracting data from the video: the object detector and the MOT model. The remaining module aims to estimate the speed of vehicles. We also present a fourth module responsible for processing the data collected by the three previous modules to produce statistics and present a detailed traffic analysis. Figure 3.1 illustrates the workflow of the presented proposal.

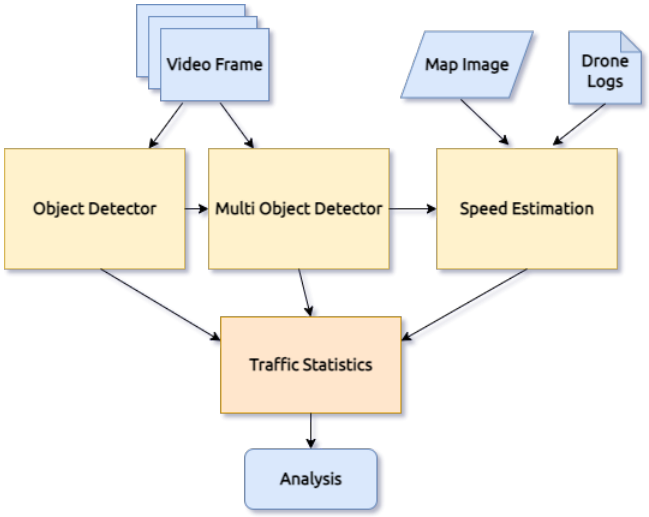


Figure 3.1: Proposed method diagram.

We also present the technologies and tools we use to implement the pro-

posed method. Thus, we divide this chapter into five sections: Object Detection module, Multi-Object Tracking module, Speed Estimation module, Traffic Statistics, and Technologies and Tools. In each one, we describe the implementation as well as the rationale for it.

3.1 Object Detection Model

In order to be able to detect objects in the several frames of the drone video we use an object detector. We implemented three different detectors to compare their performance towards the problem. The detectors chosen were the YOLOv5 [9], the TPH-YOLOv5 [7], and the Faster R-CNN [3]. The rationale for this choice involved comparative research of other works related to this problem. The paper [11] collects several detector results used in related works and sets them up in a table to provide a comparative analysis. This analysis was essential for choosing the detectors used in this work. Table 3.1 was taken from [11] paper and shows the comparison between different models:

Reference	Dataset	Model	Analysis
Li et al., 2021 [27]	Remote sensing images collected from GF-1 and GF-2 sattelites.	Faster R-CNN YOLOv3 SSD	YOLOv3 has higher accuracy and performance than SSD and Faster R-CNN models.
Zhao et al., 2019 [28]	Google Earth and DOTA	SSD Faster R-CNN YOLOv3	YOLOv3 has higher accuracy and performance than Faster R-CNN and SSD.
Bochkovski et al., [29]	MS COCO dataset	YOLOv3 YOLOv4	YOLOv4 has higher accuracy and performance than YOLOv3
Ge et al., [30]	MS COCO dataset	YOLOv3 YOLOv4 YOLOv5	YOLOv5 has higher accuracy than YOLOv3 and YOLOv3 has higher performance than YOLOv4

Table 3.1: Comparison between different object detectors (from [11]).

This table shows that YOLOv3 has better accuracy and performance than SSD and Faster R-CNN. On the other hand, YOLOv5 has a higher mAP compared to YOLOv3. YOLOv5 features several settings that allow for a tradeoff between accuracy and performance. These settings are minor changes in the model's backbone and head and the tuning of hyperparameters. Figure 3.2 was taken from [9] and compares each configuration's inference time and accuracy.

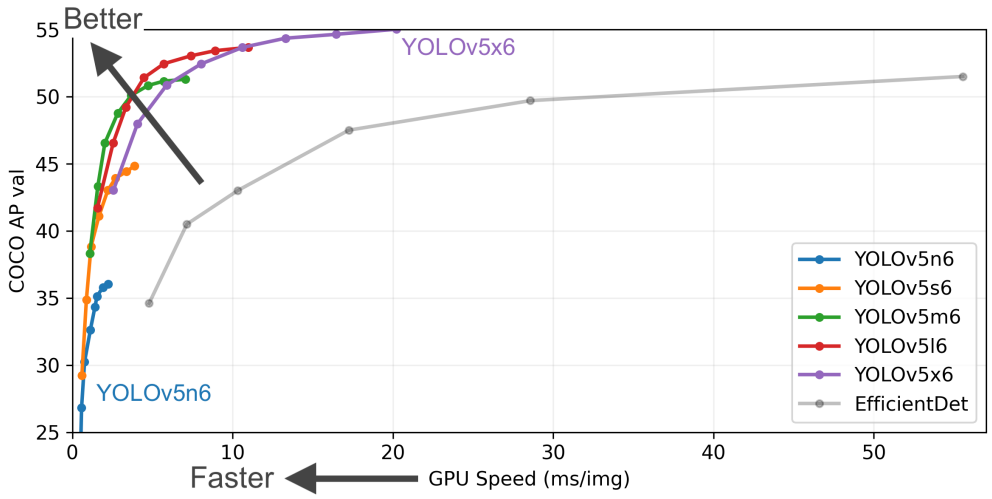


Figure 3.2: Comparison between YOLOv5 configs (from [9]).

The TPH-YOLOv5 was mentioned in the state-of-art 2 as a model with specific mechanisms for detecting objects in drone videos. Therefore, we decided to use two single-stage and one multi-stage detector models in this work. For the single-stage models, we used YOLOv5 with l configuration since this offers a good tradeoff between accuracy and performance and TPH-YOLOv5 because it is a specific detector for this problem. Thus we can make a comparison between these two models. We decided to use Faster R-CNN for the multi-stage model as it is state-of-art for models of this type.

3.2 Multi-Object Tracking Model

In order to locate the objects while preserving their identity through the different frames, we used a MOT Model. As mentioned, we chose three different object detectors to compare their performance. However, in the case of the MOT model, we decided to choose just one. Therefore, we decided to choose DeepSort [23] because it is a state-of-art MOT model and a tracking-by-detection model. The latter means that the objects do not need to be entered manually to be tracked. Instead, they are chosen automatically through a detector. Another reason for this choice is that DeepSort is an online tracking model, i.e., we can use it on a sequence of frames directly without having to use a batch of images. This also means that we can visualize the predicted output in real-time, which is one of the decided requirements for this work. According to the article [31], which analyzes the performance of several MOT models, we

conclude that DeepSort presents a good tradeoff between accuracy and performance.

The metric used to evaluate the DeepSort's predicted results was HOTA. As mentioned in 2, MOT model metrics are a complicated task. For that reason, we decided to choose HOTA because it does not over-emphasize the detection and association measurements and is divided into sub-metrics that help us to understand in a more detailed way the performance of the MOT model. In this way, we use the MOT standard format to represent the prediction results of our model. This representation is given by: frame's id, object's id, x, y, w, h, object confidence, and object class where x and y are the top centers bounding box coordinates, and w and h are respectively the width and height of the bounding box.

3.3 Speed Estimation

As mentioned before, tracking allows us to know the position of vehicles through the sequence of video frames. In this way, it is possible to estimate the speed of vehicles. The speed formula is given by:

$$speed = \frac{d}{t}$$

where d is the distance traveled and t is the time elapsed. The time elapsed can be determined directly since we know how many frames per second (fps) the video has. The fps can be obtained with *get(cv2.CAP_PROP_FPS)*, a Python's OpenCV library method. So, to calculate the time interval between frames, we determine:

$$\text{seconds per frame} = \frac{1}{\text{fps}}$$

To calculate the vehicle's displacement, we have to register the successive positions of the vehicle through the video frames. In order to facilitate this, we have summarized the object into a single point. This point is determined by calculating the centroid of the bounding boxes as follows:

$$centroid = (\frac{(x_1 + x_2)}{2}, \frac{(y_1 + y_2)}{2})$$

where x_1 and y_1 are the top-left coordinates of the bounding box and x_2 and y_2 are the right bottom coordinates of the bounding box.

Since we want to determine the speed in the International System units (m/s), we have to convert the pixel coordinates into geographic coordinates (latitude and longitude) to measure the distance in meters. It is impossible

to perform this conversion with just the video data since it does not contain information about the camera's position. However, obtaining the drone coordinates through the flight logs is possible. The proposed method to solve this problem involves georeferencing the video frames. The georeferencing of an image is a technique that allows mapping the coordinates in pixels to geographic coordinates. This technique has to be done manually and is similar to the Image Registration technique widely used in computer vision when it is intended to transform different images into a single coordinate system. For this reason, it is practically impossible to perform this technique manually since doing it in each video frame would become a very labor-intensive process. However, this would be the ideal solution.

The proposed solution to automate this process was to use only one georeferenced image, this being the map of the place where the drone recorded the video. Through the logs, we could determine the drone's coordinates on this map in each frame. It is possible to extract the heading and the height too. This data makes it possible to reproduce the drone's footprint on the map accurately. The drone's footprint is the area the drone's camera covers on the ground. Figure 3.3 illustrates the footprint calculation.

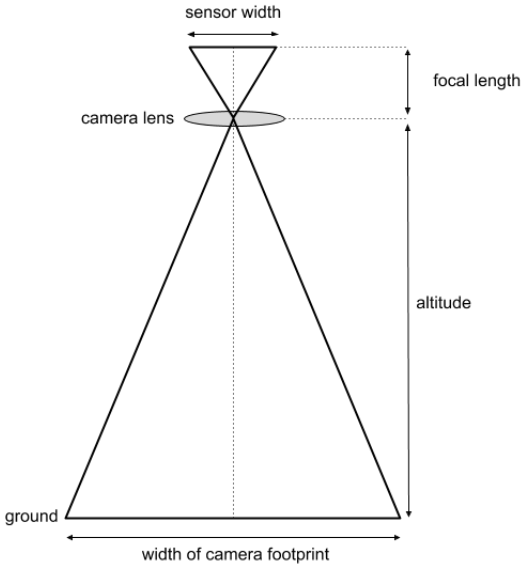


Figure 3.3: Drone's footprint illustration (from [10]).

The sensor width and focal length were obtained a priori through camera specifications. These specifications were taken from this source: `https://docs.google.com/spreadsheets/d/1w5PXRstTtZ0xMOewYdNrtzpvyVXHISPNwa65XuqcXEI/`

edit#gid=0 Therefore, for the drone used in this work, DJI Phantom 4 Pro v2, the camera specs are shown in 3.2.

Sensor width (mm)	13.2
Focal length (mm)	8.8
Image width	5477
Image Height	3612

Table 3.2: DJI Phantom 4 Pro v2 camera specs.

In order to calculate the footprint width and height, the Ground Sample Distance (GSD) ratio must be determined. This ratio relates to distance (cm) and pixels, so a GSD of value 1 means that 1 centimeter in the ground corresponds to 1 pixel in the image. The GSD formula is given by:

$$\text{GSD} = \frac{\text{sensor width} * \text{altitude} * 100}{\text{focal length} * \text{image width}} \tag{3.1}$$

The drone videos had to be recorded with the camera facing downwards, with a pitch of $-90°$. In this way, we can assume that the ground is two-dimensional, and there is no need to account for ground elevation. Once we know the drone heading, we can rotate the footprint rectangle to match the video frame correctly. Figure 3.4 is an example of the drone's footprint on the map, as well as the corresponding frame.

(a)

(b)

Figure 3.4: (a) Drone footprint. (b) Video frame.

In this way, it is possible to map the coordinates of the objects to the georeferenced map. For this, we used the map cropped footprint as an intermediate to convert the pixel coordinates to latitude and longitude. Figure 3.5 illustrates the schematic of the proposed approach:

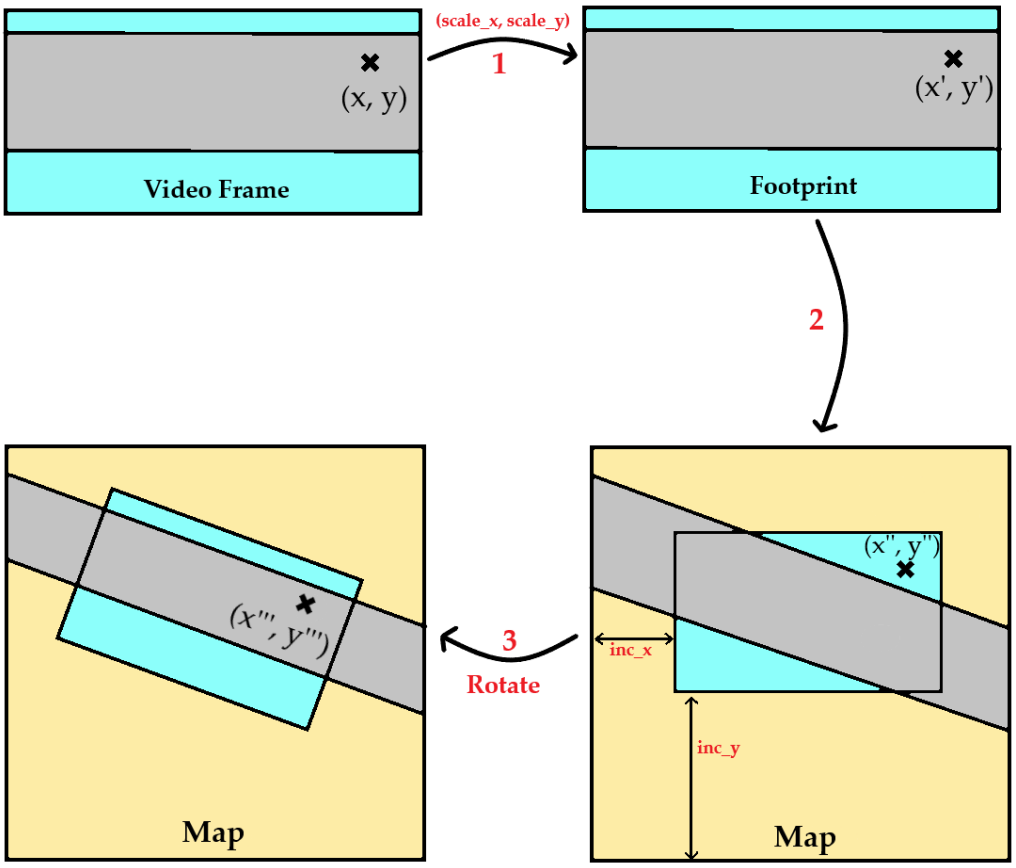


Figure 3.5: Schematic about converting a point of a frame to a footprint.

The proposed algorithm to convert the coordinate system consists of four steps:

- **Scale the frame points to cropped footprint (1)** - The scale values that convert a framing point to the footprint are determined as follows:

$$\text{scale}_x = \frac{\text{frame width}}{\text{footprint width}}$$
$$\text{scale}_y = \frac{\text{frame height}}{\text{footprint height}}$$

Then, the conversion is given by:

$$\text{point}' = (\frac{\text{point}_x}{\text{scale}_x}, \frac{\text{point}_y}{\text{scale}_y})$$

where point_x and point_y are the point coordinates.

- **Scale the cropped footprint points to map coordinates (2)** - To perform this conversion, we need to add a value to each coordinate of each point coordinate. This process will place the points in the right location on the map. These increment values are given by:

$$\text{inc}_x = \frac{\text{drone x coordinate pixel on the map}}{\text{footprint's width in pixels}}$$
$$\text{inc}_y = \frac{\text{drone y coordinate pixel on the map}}{\text{footprint's height in pixels}}$$

  So for each point, we do:

$$\text{point}'' = (\text{point}'_x + \text{inc}_x, \text{point}'_y + \text{inc}_y)$$

- **Rotate the points (3)** - To match the points with the drone heading, we have to rotate them. For this, we apply the rotation matrix to each point according to the heading angle value of the drone:

$$\text{point}''' = \begin{bmatrix} \cos(\text{heading}) & -\sin(\text{heading}) \\ \sin(\text{heading}) & \cos(\text{heading}) \end{bmatrix} \cdot \begin{bmatrix} \text{point}''_x \\ \text{point}''_y \end{bmatrix}$$

- **Convert pixels into coordinates** - Finally, we take advantage of the fact that the map image is georeferenced to convert pixels into latitude and longitude coordinates.

With this, we can estimate the distance in meters a given vehicle performed between frames. For this, we used the Haversine formula as follows:

$$\begin{aligned} Distance(m) = 3963000 \times \arccos[&\sin(Lat_A) \times \sin(Lat_B) + \\ &\cos(Lat_A) \times \cos(Lat_B) \times \cos(Lon_B - Lon_A)] \end{aligned}$$

where Lat_A and Lon_A are the latitude and longitude values of point A and Lat_B and Lon_B of the point B. Finally, we calculate the speed by applying the formula described above 3.1.

3.4 Traffic Statistics

In fact, the object detection, tracking, and speed estimation modules allow us to extract information about the urban environment in which the drone flew. It is possible to extract statistics that help us understand how many vehicles circulate on a given road and the type of this. Once we can estimate the speed, it is possible to deduce the average speed of a given road. Finally, we

can analyze the traffic density and create a heatmap that illustrates this analysis. Therefore, in this section, we discuss each approach implemented for extracting traffic statistics. These are Heatmap, Vehicle counting, and Speed Statistics.

3.4.1 Vehicle counting

In order to know how many cars enter a lane or leave it, we implement a vehicle counting technique. To do this, we start by creating one or more pairs of points in the map image. These pairs are needed to create line segments. This process is made with the help of Python's OpenCV [32, 33] library. As we did in 3.3, we decided to summarize the vehicles to the centroid of their respective bounding box. During program execution, these centroids are stored with the respective vehicle id in memory. In order to count whether an object has passed through any entered line segment, we check if this line intersects with the line segment created, respectively, with the first and last points of the object's centroid stored in memory. Figure 3.6 shows an example of the described algorithm.

Figure 3.6: The black circle shows the intersection. Inside this circle, the red line is the line segment between the first object's centroid coordinate (black cross) and the last object's centroid coordinate stored in memory. The green line is the entered segment line.

Whenever an object intersects the introduced line, the counter increments, keeping information about the quantity and category of the objects that passed

through the line.

3.4.2 Heatmap

A heatmap represents the distribution and density of data through a map with color variations. We decided to implement this mechanism to obtain a visualization of the traffic density, that is, the concentration of objects detected by our model on the ground. For this, we use a hash table as a structure to store this information. In this table, the key corresponds to the object's centroid point, while the table value corresponds to the number of times this point is repeated through the frames. In this way, if one or several vehicles remain stationary in a sequence of frames, we can emphasize their concentration on the road. In order to represent this concentration in a heatmap, we make a Kernel Density Estimation (KDE). This method applies a kernel function to each point in our data and finally sums up all these functions, in order to highlight the regions with the highest density.

The kernel function we used is the Gaussian available in Python's scipy [32, 34] library. After applying this algorithm to our data, we represent the result in a density map. We apply warmer colors in densest areas and cooler colors in less-dense areas. This technique results in a heatmap that illustrates the concentration of vehicles on the road. Figure 3.7 shows an output heatmap example.

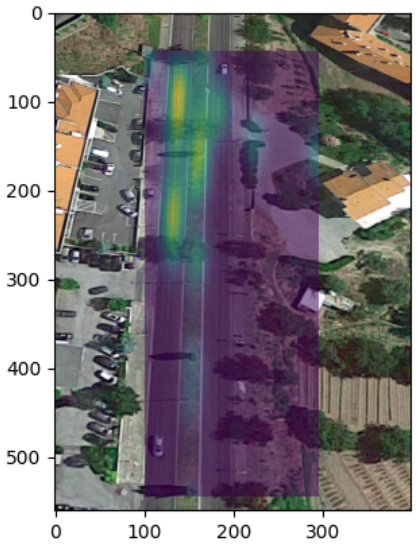


Figure 3.7: Example of the output heatmap. The hotter color (yellow) represents the high traffic density on the road.

3.4.3 Speed Statistics

With the Speed Estimation module, we can calculate some statistics by considering the speed of vehicles. We present three types of possible statistics: average speed by vehicle class, average speed by given id, and the average speed for a specified region.

These first two can be calculated directly by introducing the vehicle category or id as program input.The output will be the speeds these objects performed throughout the video. For the last mentioned statistic, it is necessary to specify the desired region with two points. From there, we create a rectangle. Then, we store the instantaneous velocity of the objects inside this rectangle into a list. For each frame, if possible, we calculate the average of this list and store this value in another list. Finally, we average this last one to obtain the region's average speed.

3.5 Technologies and Tools

3.5.1 Programming Language and Libraries

We decided to build this project using the Python [32] programming language. This choice is because it is a top language for machine learning and computer vision since it has complete libraries and frameworks that help us develop this project. The main libraries used in this work were Pytorch [35] (machine learning framework), OpenCV [33] (Computer Vision and Machine Learning library), Numpy [36] and Scipy [34] (libraries for linear algebra), and Matplotlib [37] (library with tools to build plots).

3.5.2 Packages

The models mentioned above were implemented from their official GitHub repository. Table 3.3 shows the implemented models and their respective link to the GitHub repository:

Model	GitHub repository link
YOLOv5 [9]	`https://github.com/ultralytics/yolov5`
TPH-YOLOv5 [7]	`https://github.com/cv516Buaa/tph-yolov5`
Faster R-CNN [3] visdrone	`https://github.com/oulutan/Drone_FasterRCNN`
DeepSort [23]	`https://github.com/nwojke/deep_sort`

Table 3.3: GitHub repositories and respective link.

3.5.3 Data Annotation

In order to annotate the video frames captured by the drone, we use the CVAT [38]. This tool allows to select a bounding box on objects and assign them a label and an ID. It uses an optional interpolation algorithm that allows skipping a few frames and keeping track of objects. Thus it facilitates the annotation of big data such as video frames. Finally, CVAT allows returning the results in MOT format.

3.5.4 Devices and Logs

The drone used for this project was the DJI Phantom 4 Pro V2.0. Information and specifications for this can be viewed at `https://www.dji.com/en/phantom-4-pro-v2/specs`. The drone's flight logs cannot be viewed directly as the file format is binary. For that, it was necessary a tool to parse these logs. This tool is online and can be found at `https://airdata.com/`.

3.5.5 Georeferencing Tools

The map images were taken from Google Earth Pro. It provides access to a vast database with satellite images. However, this platform does not provide information about the georeferencing of images, i.e., a GeoTIFF file. In fact, it is possible to draw a path on the image and export it in a format (KML file) that stores the coordinates of this path. Then it is possible to manually georeference the images using QGIS [39], a platform that offers tools for geographic purposes. In this way, the images of the maps used in this project are georeferenced and stored in GeoTIFF format.

Chapter

4

Experiments and Results

In this chapter, we present the results obtained for each of the three modules used. Then for each of these, we discuss the results, pointing out the advantages, disadvantages, and a qualitative analysis. Finally, we make a general assessment of the framework's performance in extracting statistics.

4.1 Dataset

The dataset used for this project is the VisDrone [19] dataset. VisDrone is part of a challenge that aims to make competition between different object detectors capable of identifying objects through drone videos. The dataset contains 10208 images captured by a drone at different locations and heights. All the images are annotated with the bounding boxes and the object classes (pedestrian, person, car, van, bus, truck, motor, bicycle, awning-tricycle, and tricycle). Table 4.1 shows the distribution of the data present in the object detection dataset:

Training	6471
Testing	3190
Validation	548
Total	10209

Table 4.1: Visdrone dataset data distribution.

The dataset used for training only falls into the object detection category, although there is also a dataset for Multi-Object Tracking. As mentioned in this chapter 3.2, DeepSort [23] is part of tracking-by-detection models, i.e., it

depends only on the detector to identify objects. However, we use the MOT dataset for testing the model.

4.1.1 Data Variability Factors

There are factors in dataset images that can influence the detection system. These factors consider the number of vehicles in an image, size, occlusion, and luminosity. We present sample images from Visdrone's object detection dataset for each of these.

Figures 4.1 show examples of images with the presence of vehicle clusters.

Figure 4.1: Vehicle clusters images from Visdrone detection dataset.

Images where vehicles are too concentrated, can be challenging to detect and classify. Figure 4.2 shows examples of dataset images with traffic jams where these factors are visible.

Figure 4.2: Traffic jam images from Visdrone detection dataset.

This dataset contains images taken from different perspectives and heights. There are images in which the camera was higher and vertical, as seen in figure 4.3 in the image on the left. However, there are also images where the camera was lower and horizontal, as shown in figure 4.3 on the right.

Figure 4.3: Images with different perspectives and heights from Visdrone detection dataset.

Images with low or high luminosity can be challenging to detect. These examples can be seen in figure 4.4, wherein the left image, the traffic is exposed to high illumination, and in the right, it is exposed to low illumination.

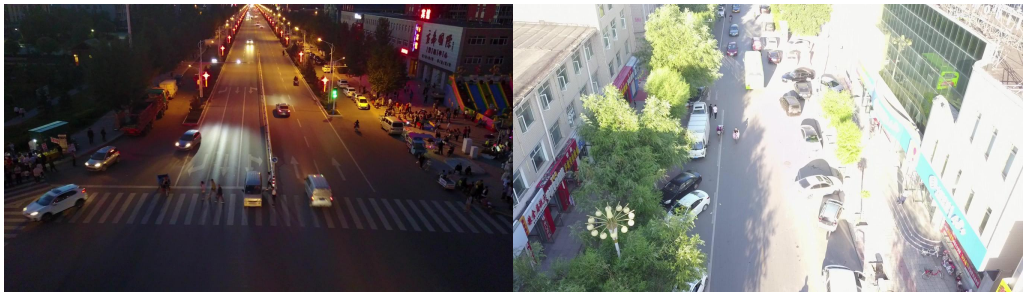


Figure 4.4: Images with different illumination from Visdrone detection dataset.

There are images in the dataset in which the vehicles have tiny dimensions, which can be a challenge for detection, as seen in Figure 4.5.

Figure 4.5: Images with tiny objects from Visdrone detection dataset.

Finally, the partial occlusion of an object is a factor that makes its detection difficult. This dataset presents images of vehicles partially hidden in

other objects such as trees, walls, or other vehicles. Figure 4.6 shows an example where the occlusion factor in vehicles is notable.

Figure 4.6: Image with occlusion from Visdrone dataset.

4.2 Object Detection

4.2.1 Preprocessing and Training

In the three models used, the images are preprocessed before forwarding. The techniques used by the three are the resizing and normalization of the images. In the case of YOLOs, only multiple resolutions of the max stride are supported, which by default is 32. Finally, all images are normalized so that values lie between 0 and 1, i.e., are divided by 255, which is the maximum value of the channels RGB.

VisDrone training and test datasets were used to train and test the three models. However, only YOLOv5 [9] was trained by us using the recommended settings and hyperparameters for the mentioned dataset. The weights provided in the official GitHub repository were used for the other two models. In the YOLOv5 training, we used a model already pre-trained in the COCO Dataset available in the official repository of GitHub [40]. The training settings used for YOLOv5 can be seen in table 4.2.

Epochs	150
Batch Size	15
Image Size	Stochastic Gradient Descent (SGD)
Number of classes	10

Table 4.2: YOLOv5 training settings.

The hyperparameters used in training are presented in table 4.3.

learning rate	0.01
SGD momentum	0.937
Optimizer weight decay	0.0005
Warmup epochs	3.0
Warmup momentum	0.8
Warmup learning rate	0.1
Box loss gain	0.05
Class loss gain	0.5
Class BCELoss positive weight	1.0
IOU training threshold	0.20
Objectness loss gain	1.0
Objectness BCELoss positive weight	1.0

Table 4.3: YOLOv5 training hyperparameters.

In addition, data augmentation is performed on the data before training. Table 4.4 presents these parameters respectively.

Image HSV-Hue augmentation	0.015
Image HSV-Saturation augmentation	0.7
Image HSV-Value augmentation	0.4
Image translation	0.1
Image scale	0.5
Image flip left-right	0.5 (probability)
Image mosaic	1.0 (probability)

Table 4.4: Data augmentation parameters.

The machine's specifications used for training are Nvidia RTX 2080 Ti (GPU), i5-8600K (CPU), and 8 Gb RAM. The results were generated automatically using the Wandb [41] software, shown in figure 4.7.

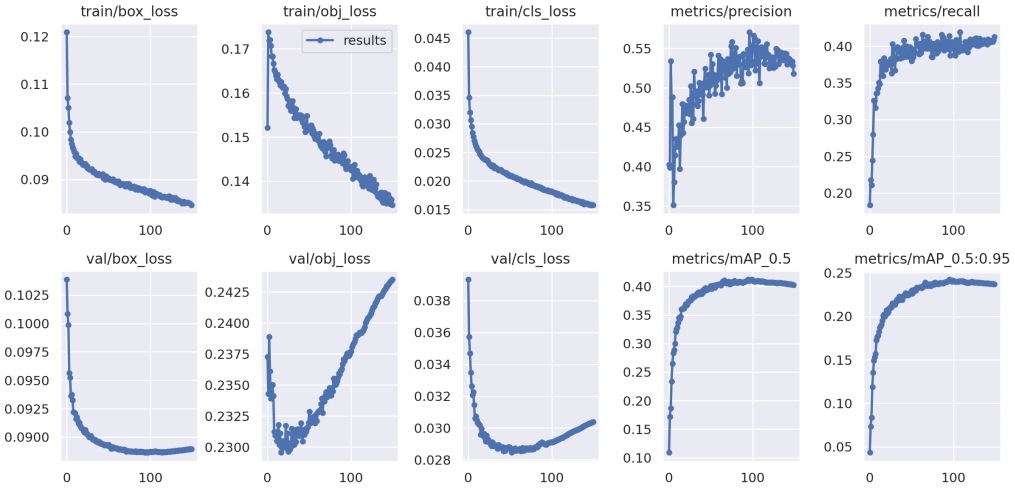
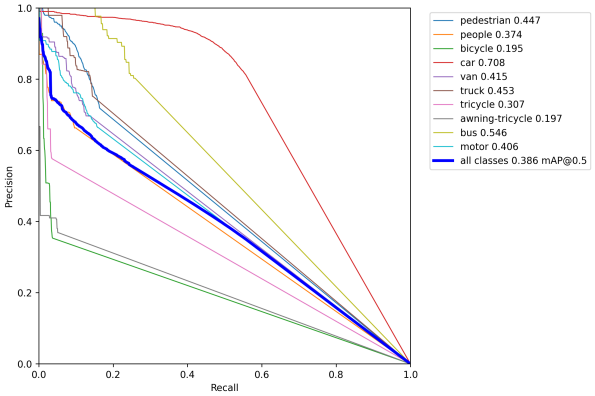


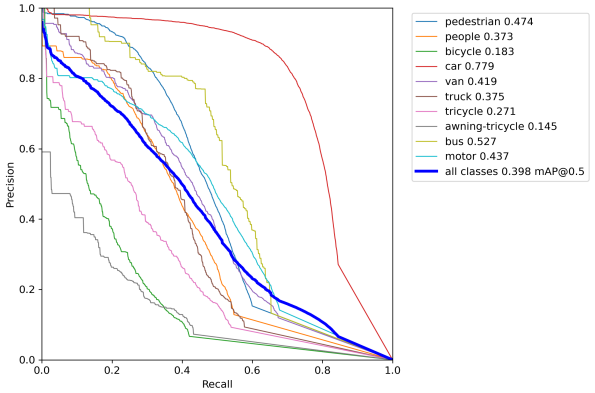
Figure 4.7: YOLOv5 training results.In the first three top and bottom plots, the x-axis corresponds to the number of epochs while the y-axis corresponds to the loss. In the last two bottom and top plots, the x-axis corresponds to the number of epochs while the y-axis corresponds to the metric value.

4.2.2 Results

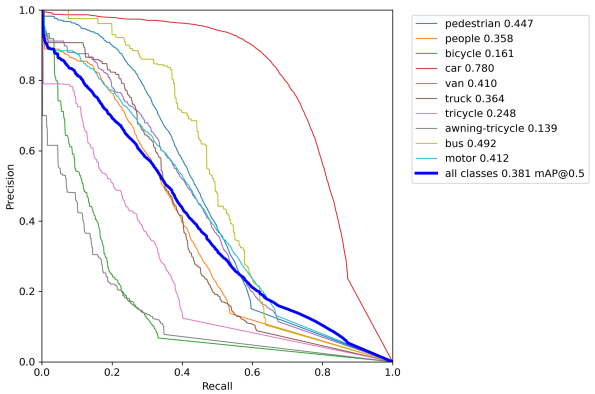
We used the Visdrone detection test dataset for each of the three models to evaluate the object detection results. We made three evaluations for each model, changing only the image resolution. Then we can compare the results obtained and conclude the best tradeoff between accuracy and inference time. For both YOLO models, we used a batch size of 4, a confidence threshold of 0.001, and an IOU threshold of 0.60. These thresholds are chosen because they are the values used as standard in this type of evaluation. For Faster R-CNN, we use a batch size of 1, a confidence threshold of 0.70, and an IOU threshold of 0.5. These are the threshold values recommended by the official implementation of this model. The resolutions chosen to compare the performance of the models are 640, 1280, and 1920. The figures 4.8, 4.9 and 4.10 presents the Precision-Recall curve of each model's results for each resolution.



(a) Faster R-CNN

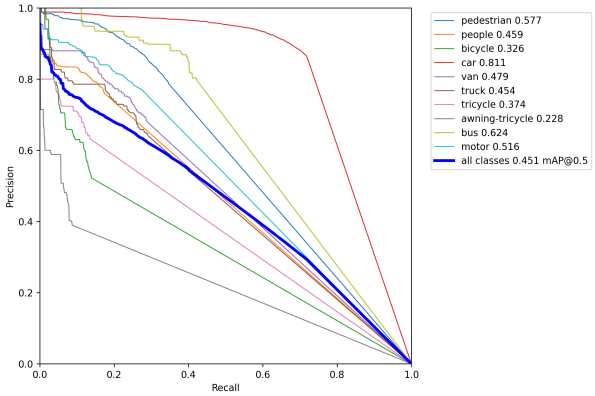


(b) YOLOv5

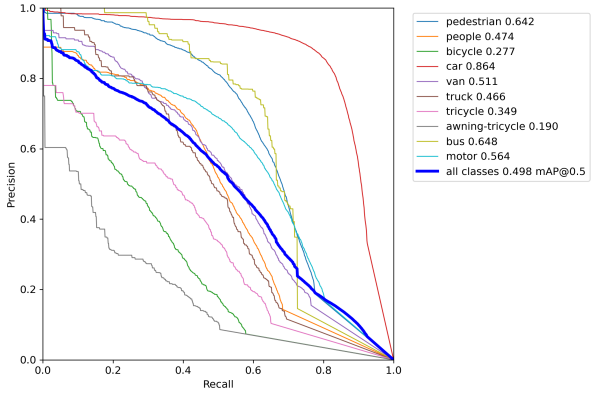


(c) TPH-YOLOv5

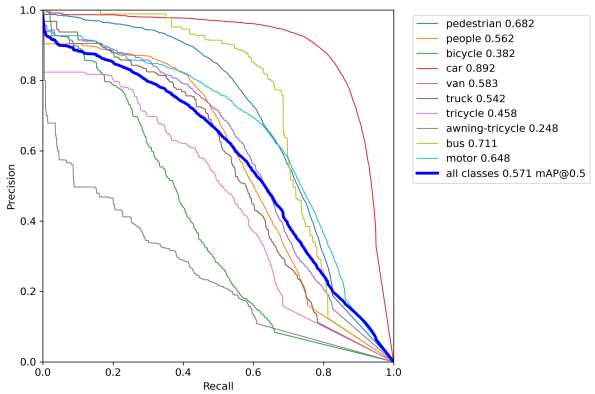
Figure 4.8: Results of each model with resolution at 640.



(a) Faster R-CNN

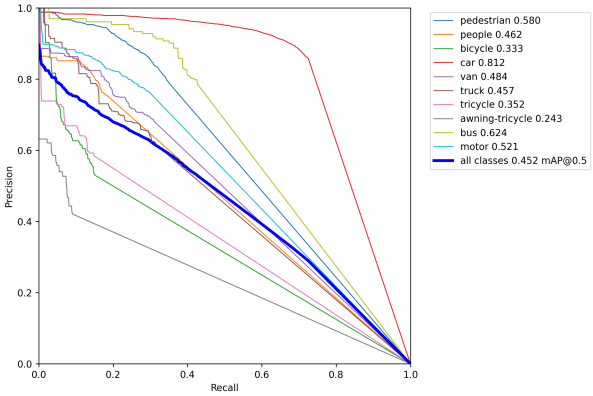


(b) YOLOv5

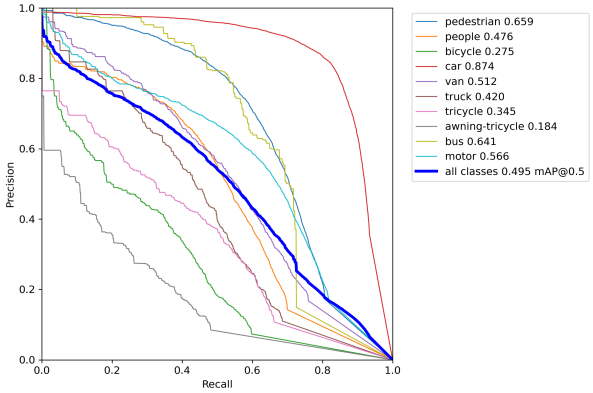


(c) TPH-YOLOv5

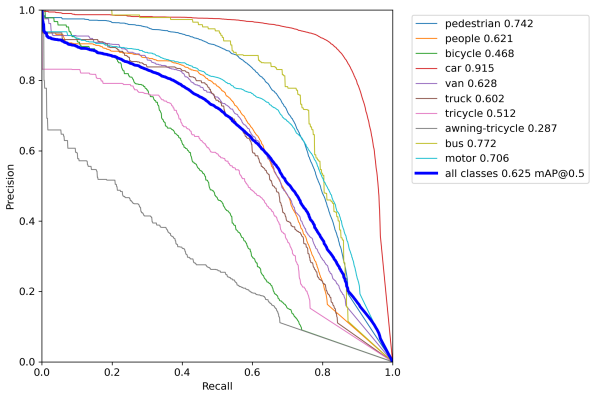
Figure 4.9: Results of each model with resolution at 1280.



(a) Faster R-CNN



(b) YOLOv5



(c) TPH-YOLOv5

Figure 4.10: Results of each model with resolution at 1920.

Table 4.5 presents the average inference time of the images of each model for each resolution.

	Faster R-CNN	YOLOv5	TPH-YOLOv5
640	145.01 (ms)	47.2 (ms)	93.0 (ms)
1280	269.61 (ms)	91.1 (ms)	264.3 (ms)
1920	280.90 (ms)	155.8 (ms)	583.4 (ms)

Table 4.5: Model's inference times.

We did some tests in order to evaluate the detector's performance according to the factors mentioned in 4.1.1. For this, the TPH-YOLOv5 was used with IOU and confidence threshold at 0.5 and resolution at 1920.

Figure 4.11 presents the result of a scenario with vehicle clusters having obtained 0.61 mAP. As can be seen, most vehicles are successfully detected even in some occlusion zones (red circles).

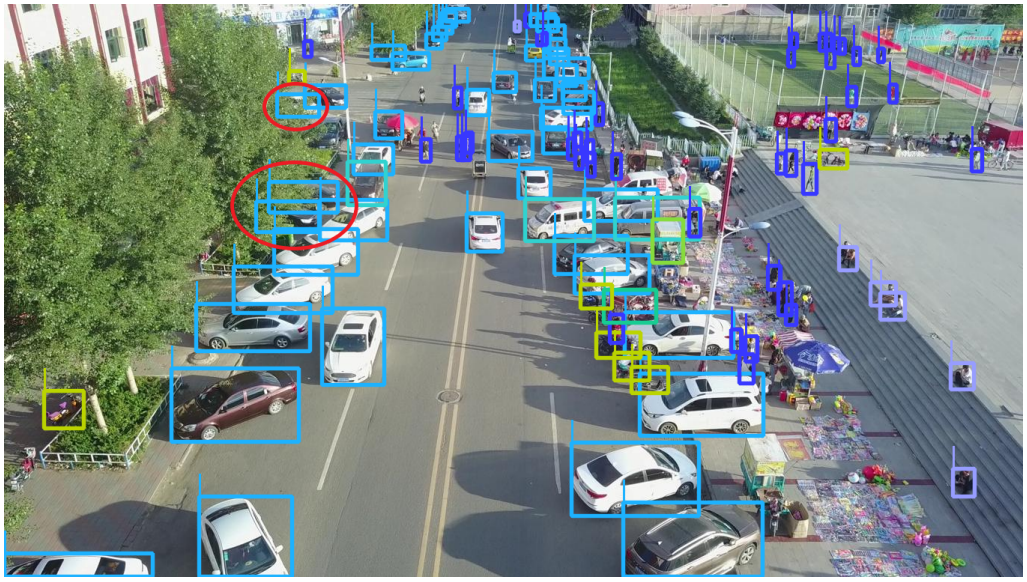


Figure 4.11: Vehicle cluster prediction results.

Figure 4.12 illustrates the results obtained in a traffic jam scenario with 0.65 mAP. We can see that most of the closest vehicles are detected. However, objects further away from the camera are not well detected.

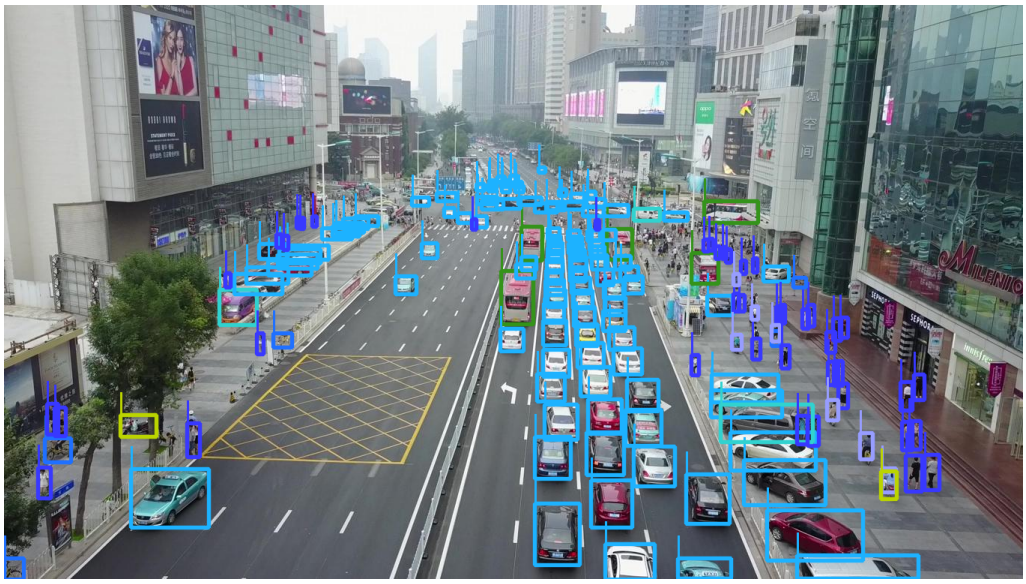


Figure 4.12: Traffic jam prediction results.

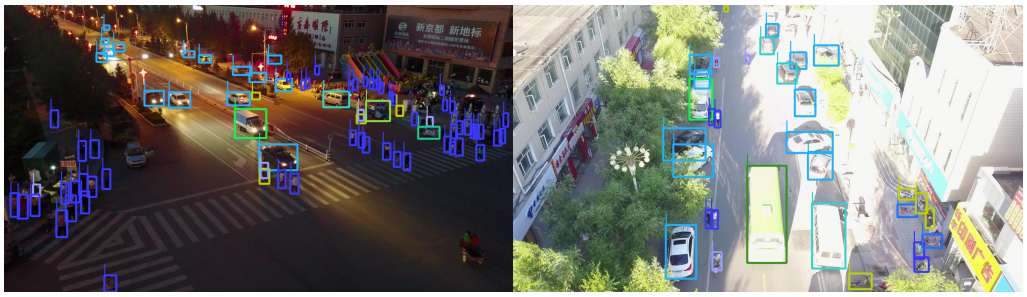
Figure 4.13 (a) illustrates an example of an image captured in a vertical perspective and at a high altitude, having obtained 0.51 mAP. From this perspective, we can see that most vehicles were detected with some class switching. However, most people were not appropriately detected, perhaps because they are not so well visible at this altitude. On the other hand, when the camera is very close to the vehicles, they are not detected, as seen in the figure 4.13 (b), having obtained 0 mAP.

(a) (b)

Figure 4.13: Images with different perspectives. (a) High altitude and vertical perspective prediction results.

In low luminosity scenarios, the detector performs well, having obtained 0.80 mAP in image (a) of figure 4.14. Most vehicles on the road were detected

correctly, perhaps because of the lights. On the left of this image, we can see that a car was not correctly detected as it is in a darker area. When the scenario's luminosity is very high, the detector presents some difficulty, having been unable to detect three vehicles exposed to much light, as seen in figure 4.14 (b).



(a) (b)

Figure 4.14: Images with different light scenarios. (a) Low luminosity. (b) High luminosity.

As mentioned in some of the results above, the detector presents difficulties when the vehicles have tiny dimensions, as shown in figure 4.15, having obtained only 0.33 mAP.

Figure 4.15: Tiny objects prediction results.

Finally, in partial occlusion of the vehicles, the detector performs well,

having managed to detect most of the possible objects in figure 4.16, having obtained 0.78 mAP.

Figure 4.16: Parcial occlusion prediction results.

4.3 Multi-Object Tracking

To test the DeepSort together with the object detectors, we used the Visdrone multi-object tracking dataset. This dataset contains 33 sequences in a total of 12968 frames annotated in MOT format. As we did in 4.2, we decided to test this model in 3 different resolutions: 640, 1280, and 1920. The thresholds used in the three detectors were 0.5 for both the IOU and the confidence. Tables 4.6, 4.7, and 4.8 show the results obtained for each mentioned resolution. The results represent the average scores obtained in all classes and are in HOTA metrics.

	HOTA	DetA	AssA	LocA
Faster R-CNN	24.42	23.18	27.70	87.93
YOLOv5	27.17	26.77	31.81	88.07
TPH-YOLOv5	32.41	30.66	37.15	90.88

Table 4.6: Average HOTA scores with frames resolution at 640.

	HOTA	DetA	AssA	LocA
Faster R-CNN	30.08	29.65	32.89	88.15
YOLOv5	31.50	30.99	35.54	88.13
TPH-YOLOv5	35.99	35.48	40.96	90.20

Table 4.7: Average HOTA scores with frames resolution at 1280.

	HOTA	DetA	AssA	LocA
Faster R-CNN	31.12	30.37	34.33	88.58
YOLOv5	32.53	30.93	36.80	89.06
TPH-YOLOv5	37.10	36.06	41.55	90.04

Table 4.8: Average HOTA scores with frames resolution at 1920.

4.4 Speed Estimation

Since there is no dataset with these specifications, we decided to collect our data with the drone and therefore have more control over this data. For this testing, we focused only on a specific car, which we had access to the ground truth speed values. All footage was recorded with the drone camera facing the ground ($pitch = -90°$) and the ground truth with a smartphone camera pointing to the vehicle's odometer. Since this odometer is analog, extracting these values through the video may contain human precision errors. However, this error is minimal and is ignored. In this testing, we used four videos from two scenarios: the drone is stationary, and the drone is in motion. In order to reduce the data noise, we used the median filter. This filter calculates the mean of all values under the kernel area k then the central values are replaced with the median value. Figures 4.17, 4.18, 4.19, and 4.20 present each video's raw and smoothed speed estimation results with the ground truth. Speed is represented on the y-axis in kilometers per hour and time on the x-axis in seconds.

- **The drone is stationary**: In Figure 4.17, the car remained at a constant speed of about 25 km/h between second 12 and 54.

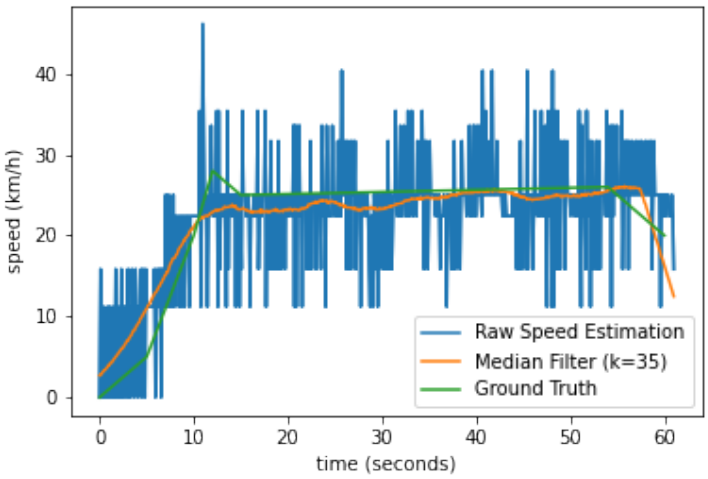


Figure 4.17: Speed estimation with stationary drone.

In Figure 4.18, the car remains stationary for 33 seconds and then speeds up to 30 km/h.

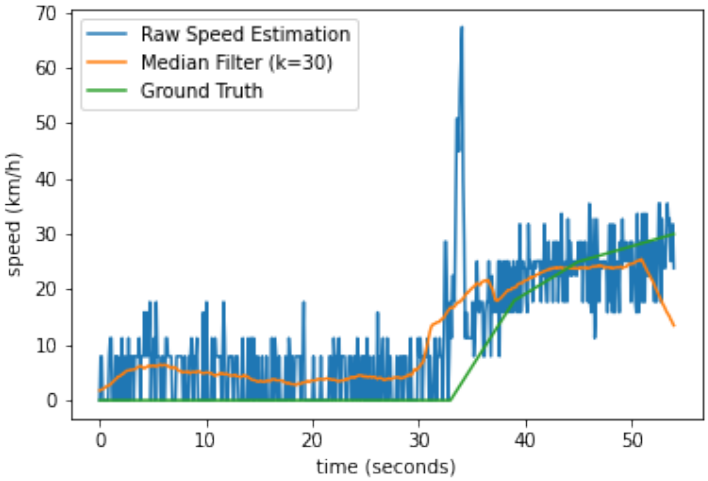


Figure 4.18: Speed estimation with stationary drone while the car remains stationary and then speeds up.

- **The drone is in motion:** In figure 4.19, both the car and the drone were in motion. The car remained at a constant speed of about 30 km/h between second 20 and 140.

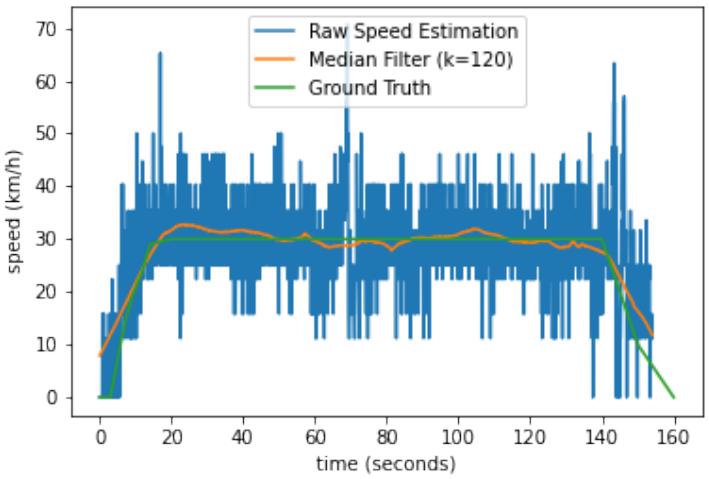


Figure 4.19: Speed estimation with both car and drone in motion.

In figure 4.20, the drone is in motion and the car remains stationary.

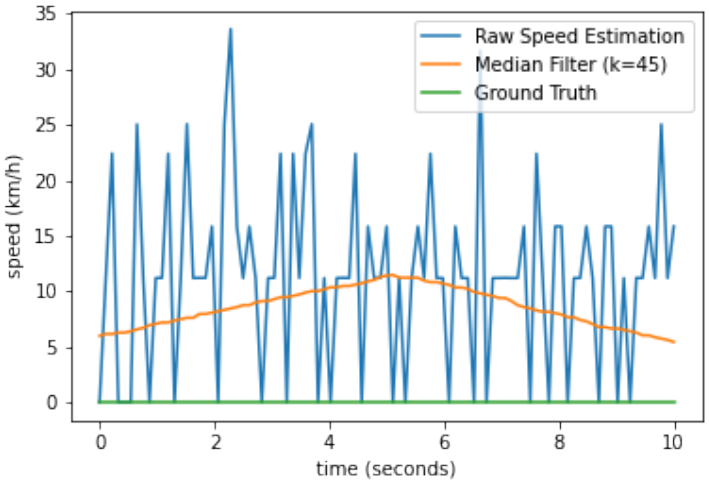


Figure 4.20: Speed estimation with drone in motion and stationary car.

4.5 Discussion

This section discusses the results in sections 4.2, 4.3, and 4.4. We describe the advantages and drawbacks of each implementation, as well as a qualitative

analysis. Finally, we discuss our framework's performance in extracting traffic statistics.

4.5.1 Object Detection

According to the results obtained in 4.2.2, we can conclude that the TPH-YOLOv5 [7] outperforms the other models in all tested resolutions. However, its inference time is much higher compared to YOLOv5 [9]. Faster R-CNN [3] always presents lower accuracies in all resolutions, and, in addition, it presents high inference times compared to YOLOv5. Therefore, it seems evident that to obtain good accuracy, we must use TPH-YOLOv5 with a resolution of 1920. However, to obtain a good tradeoff between accuracy and inference time, we must use YOLOv5 with a resolution of 1280 since this model's inference time difference between 1920 and 1280 is quite significant compared to accuracy.

In images containing small objects, YOLOv5 and Faster R-CNN struggle in identifying these compared to TPH-YOLOv5. Figure 4.21 illustrates an example where the latter can confidently identify smaller objects.

Figure 4.21: Example of small objects detection by TPH-YOLOv5 (inside red cirles).

We faced two negative details in the three models. Although these are not observed very regularly we must mention them. One of them is the inconsistency in assigning classes to objects. It is more noticeable in the video

when the models are not consistent in assigning certain classes in consecutive frames, i.e., in one frame, it classifies as a car and in the next as a truck. The other detail is an inconsistency in detection. Sometimes, the object is correctly detected in the current frame, but in the next frame, it is not. It is interesting to note this because the content between frames does not differ too much in a video.

4.5.2 Multi-Object Tracking

Since DeepSort depends on an object detector, the results verified in 4.3 are similar to what was mentioned in 4.2. The TPH-YOLOv5 outperforms the other models in all resolutions. From what we have observed, when the drone is stationary, or there is no sudden change in the camera's perspective, this model can correctly assign the IDs in most cases. Even with the drone in motion, it performs quite reasonably. However, when there are sudden changes in perspective, it often swapes the IDs. The model also presents some association difficulties in the presence of occlusions that last more than 15 frames.

4.5.3 Speed Estimation

Estimating the object's speed from the video is challenging, especially with the drone in motion. However, this module can present values very close to the ground truth, as seen in the Figures 4.17 and 4.19, whether the drone is in motion or stationary. This performance is essentially due to the matching strategy between frame and footprint. This technique allows mapping the objects to the georeferenced map, facilitating the calculation of speed in the International System unit (km/h). However, in all samples, there is noise, which means that we have to increase the median's filter k value to see similarities with the ground truth. This noise is due to several factors. One of them is that the centroid changes its position even with the car stationary due to the prediction of the bounding box since its position can vary significantly through frames. These small changes are enough for the module to wrongly assume that the car is moving and estimate a false speed, as seen in figure 4.20 and the first 30 seconds of figure 4.18. Another detail is that this module depends a lot on the tracking model, i.e., if it wrongly swaps the ID of an object, we may not be able to get its complete or correct speed estimation.

Two crucial factors to make this prediction closer to reality are the rigorous georeferencing of the map image and the correct footprint calculation. For this first one, we recommend using an image with high resolution, i.e., 3840 x 2160 or higher, and using many matching points. The footprint is crucial for this prediction, and the fact that we assume that this and the drone frame cor-

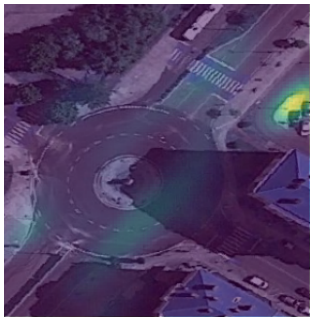
respond in totality can contribute to the imprecision of estimation and noise. We verified that the log of the height recorded from the drone sometimes deviates from the ground truth value. In order to obtain this value correctly, we calculate the difference between the log height above the sea and the value of the height above the sea referring to the ground. This second value is not present in the logs and cannot be obtained directly, having to be manually extracted from tools such as Google Earth.

4.5.4 Traffic Statistics

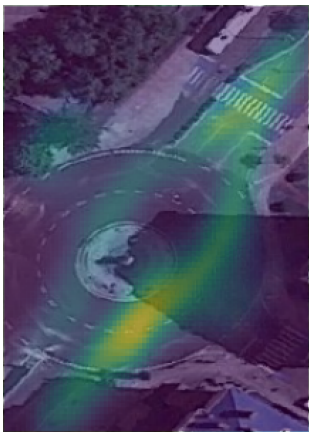
Overall, these three mentioned modules allow extracting useful traffic statistics. The Speed Estimation module depends on the MOT model, which in turn depends on the Object Detection Model. That is why we have given so much importance to choosing a robust detector. Consequently, our framework performs well when extracting traffic information.

We recorded a video with the drone stationary above a roundabout to test the performance of counting vehicles on the road. Then we selected the desired routes and compared our framework's results with the ground truth. The latter's values were 7 vehicles, of which 6 cars and 1 motorcycle. Our framework counted 6 vehicles, of which 5 cars and 1 van. It could not detect the motorcycle and confused the car with a van when crossing the line. This test generalizes the overall performance, i.e., the framework manages to count almost the entirety of the vehicles, and sometimes it confuses some classes.

The heatmap generated by the framework makes a good representation of the traffic density. In scenarios where the drone captures parked cars, it results in a heatmap with higher density zones in these locations. Since these vehicles are not on the road, they must be ignored. To solve this, we use a binary mask to filter what is the road and what is not. The ideal method would be to apply this mask directly to the video frames, implying that the drone is stationary and the camera does not rotate. Therefore, we decided to apply it directly to the map image, although for this, it must be robustly georeferenced. Figure 4.22 illustrates the difference between the result of a heatmap before and after the filter on the map.



(a) Heatmap without mask.



(b) Heatmap with mask.

Figure 4.22: Difference between heatmap with and without mask.

Chapter

5

Conclusions and Further Work

5.1 Conclusions

The main objective of this work was the development of a framework capable of analyzing car traffic through unmanned aerial devices. In order to accomplish this objective, it was required to understand: 1) How to detect and track vehicles in video frames. 2) How to estimate the object's speed from videos. 3) Methods to generate statistics using data extracted from traffic.

A detailed study was carried out on state-of-art to implement the models required for this work. This study explored Object Detection models, Multi-Object Tracking models, and their respective metrics.

We present a proposed method to automatically analyze car traffic through a video captured by a drone. The method was based on developing three modules responsible for extracting information through video frames. The two main modules are the object detector and multi-object tracking models. The third module works together with the other to estimate the speed of objects in km/h. Finally, we made a fourth module responsible for processing the statistics. We implemented a method of counting vehicles and predicting their category on a given road, as well as the traffic characterization. Other statistics included creating a heatmap that illustrates the car traffic density on the map and the output of the speed estimates of the detected cars.

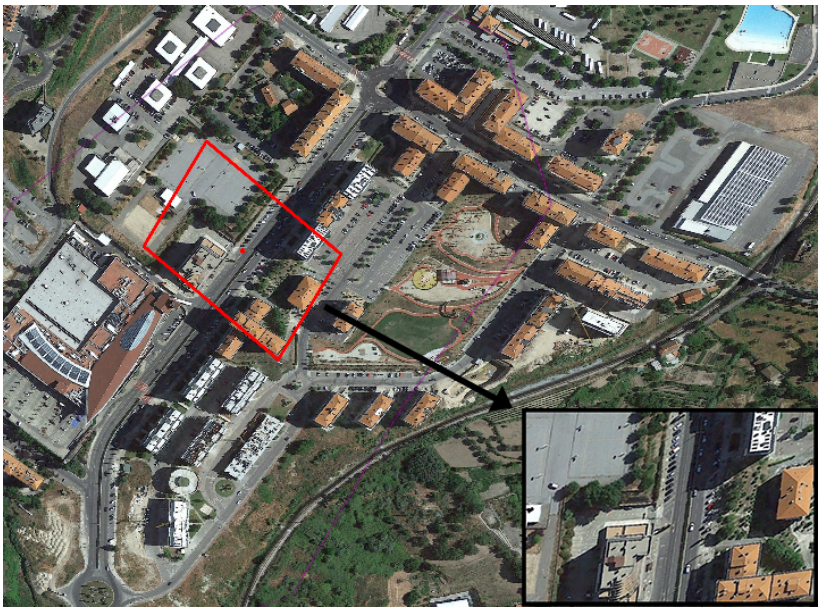
After implementing each proposed module, we performed the respective tests and discussed the results. From these, we conclude that TPH-YOLOv5 [7] outperforms the other object detection models, allowing, together with DeepSort [23], a good performance in object tracking. The speed estimation also presented results close to the ground truth ones, although with some noise. This discussion is presented in more detail in section 4.5, as well as a subjec-

tive analysis for each module.

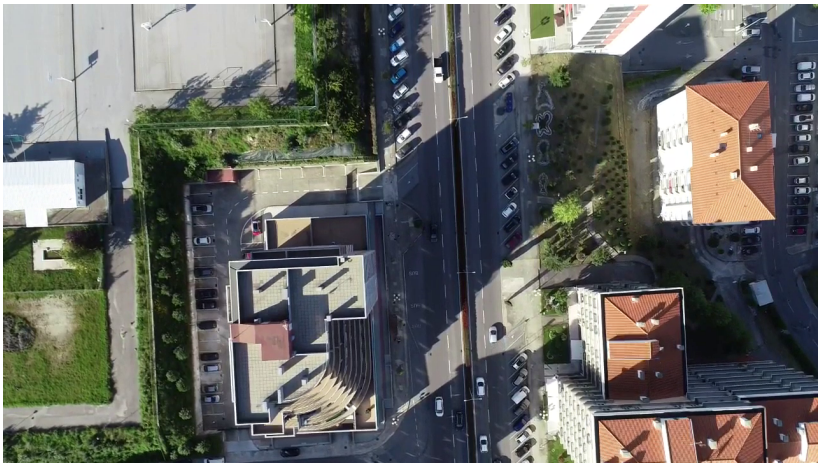
In conclusion, this work allowed the creation of a framework capable of analyzing automobile traffic automatically through unmanned aerial vehicles. We consider that, in general, all objectives were fully achieved.

5.2 Further Work

In order to improve the present work, some improvements could be made in future work. It may happen that the map image was slightly distorted after georeferencing. This anomaly implies that the footprint and the video frame are poorly matched. Consequently, it may disturb the accuracy of the speed calculation and the mapping of vehicles from the frame to the map, implying in the count and heatmap statistics. Figure 5.1 illustrates an example where the footprint distortion is perceptible and consequently a poor correspondence with the video frame.



(a)



(b)

Figure 5.1: Example of notable distortion in map image. (a) The red square is the footprint on the map, and the black arrow points to the cropped and rotated footprint. (b) Corresponding drone frame.

In fact, some technologies such as [42] allow doing Image Registration automatically to match two images. This technique could be used to match each frame of the video to the respective footprint of the drone in order to correct this map image deformations. Implementing a Google Earth API could replace the need to do manual georeferencing. In order to expand the framework's dynamics, the footprint calculation could be improved to support videos

with a pitch different from -90º.

Bibliography

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[2] Ross Girshick. Fast r-cnn, 2015.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[7] Xingkui Zhu, Shuchang Lyu, Xu Wang, and Qi Zhao. Tph-yolov5: Improved yolov5 based on transformer prediction head for object detection on drone-captured scenarios, 2021.

[8] Jonathon Luiten, Aljos
a Os
ep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. HOTA: A higher order metric for evaluating multi-object tracking. *International Journal of Computer Vision*, 129(2):548–578, oct 2020.

[9] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu , changyu98, Mingyu Wang, Naman

Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements, October 2020.

[10] Lesson 1: Measuring distance using a drone photo.

[11] Upesh Nepal and Hossein Eslamiat. Comparing yolov3, yolov4 and yolov5 for autonomous landing spot detection in faulty uavs. *Sensors*, 22(2), 2022.

[12] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks, 2013.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[14] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review, 2018.

[15] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[16] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[17] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[18] Ultralytics. Yolov5 focus() layer · discussion 3181 · ultralytics/yolov5.

[19] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Heng Fan, Qinghua Hu, and Haibin Ling. Detection and tracking meet drones challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[20] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018.

[21] Precision-recall.

[22] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2016.

[23] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 2017.

[24] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[25] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008, 01 2008.

[26] Jonathon Luiten. How to evaluate tracking with the hota metrics, Mar 2021.

[27] Min Li, Zhijie Zhang, Liping Lei, Xiaofan Wang, and Xudong Guo. Agricultural greenhouses detection in high-resolution satellite images based on convolutional neural networks: Comparison of faster r-cnn, yolo v3 and ssd. *Sensors (Basel, Switzerland)*, 20, 2020.

[28] Kun Zhao and Xiaoxi Ren. Small aircraft detection in remote sensing images based on yolov3. *IOP Conference Series: Materials Science and Engineering*, 533:012056, 05 2019.

[29] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[30] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021, 2021.

[31] Yesul Park, L. Minh Dang, Sujin Lee, Dongil Han, and Hyeonjoon Moon. Multiple object tracking in deep learning approaches: A survey. *Electronics*, 10(19), 2021.

[32] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[33] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[34] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0

Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[35] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[36] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[37] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[38] Boris Sekachev, Nikita Manovich, Maxim Zhiltsov, Andrey Zhavoronkov, Dmitry Kalinin, Ben Hoff, TOsmanov, Dmitry Kruchinin, Artyom Zankevich, DmitriySidnev, Maksim Markelov, Johannes222, Mathis Chenuet, a andre, telenachos, Aleksandr Melnikov, Jijoong Kim, Liron Ilouz, Nikita Glazov, Priya4607, Rush Tehrani, Seungwon Jeong, Vladimir Skubriev, Sebastian Yonekura, vugia truong, zliang7, lizhming, and Tritin Truong. opencv/cvat: v1.1.0, August 2020.

[39] QGIS Development Team. *QGIS Geographic Information System*. QGIS Association, 2022.

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[41] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[42] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks, 2019.