drones

MDPI

Article

Implementation of an Edge-Computing Vision System on Reduced-Board Computers Embedded in UAVs for Intelligent Traffic Management

Sergio Bemposta Rosende [1], Sergio Ghisler [2,3], Javier Fernández-Andrés [4] and Javier Sánchez-Soriano [5,*]

1 Department of Science, Computing and Technology, Universidad Europea de Madrid, 28670 Villaviciosa de Odón, Spain; sergio.bemposta@universidadeuropea.es
2 Department of Data Science, Cathedral Software, 29590 Málaga, Spain; 21732678@live.uem.es
3 Intelligent Control Systems Research Group, Universidad Europea de Madrid, 28670 Villaviciosa de Odón, Spain
4 Department of Industrial and Aerospace Engineering, Universidad Europea de Madrid, 28670 Villaviciosa de Odón, Spain; javier.fernandez@universidadeuropea.es
5 Escuela Politécnica Superior, Universidad Francisco de Vitoria, 28223 Pozuelo de Alarcón, Spain
* Correspondence: javier.sanchez@ufv.es

Abstract: Advancements in autonomous driving have seen unprecedented improvement in recent years. This work addresses the challenge of enhancing the navigation of autonomous vehicles in complex urban environments such as intersections and roundabouts through the integration of computer vision and unmanned aerial vehicles (UAVs). UAVs, owing to their aerial perspective, offer a more effective means of detecting vehicles involved in these maneuvers. The primary objective is to develop, evaluate, and compare different computer vision models and reduced-board (and small-power) hardware for optimizing traffic management in these scenarios. A dataset was constructed using two sources, several models (YOLO 5 and 8, DETR, and EfficientDetLite) were selected and trained, four reduced-board computers were chosen (Raspberry Pi 3B+ and 4, Jetson Nano, and Google Coral), and the models were tested on these boards for edge computing in UAVs. The experiments considered training times (with the dataset and its optimized version), model metrics were obtained, inference frames per second (FPS) were measured, and energy consumption was quantified. After the experiments, it was observed that the combination that best suits our use case is the YoloV8 model with the Jetson Nano. On the other hand, a combination with much higher inference speed but lower accuracy involves the EfficientDetLite models with the Google Coral board.

Keywords: UAV; drones; computer vision; deep learning; edge computing; artificial intelligence; reduced-board hardware; energy efficiency; object detection

Citation: Bemposta Rosende, S.; Ghisler, S.; Fernández-Andrés, J.; Sánchez-Soriano, J. Implementation of an Edge-Computing Vision System on Reduced-Board Computers Embedded in UAVs for Intelligent Traffic Management. *Drones* **2023**, *7*, 682. https://doi.org/10.3390/drones7110682

Academic Editors: Hiroyuki Tomiyama, Ittetsu Taniguchi, Xiangbo Kong and Hiroki Nishikawa

Received: 26 October 2023
Revised: 16 November 2023
Accepted: 17 November 2023
Published: 20 November 2023

1. Introduction

1.1. Context

Advancements in autonomous driving have seen unparalleled improvement in recent years [1]. The growing challenge of managing urban traffic efficiently and sustainably demands innovative solutions that incorporate emerging technologies such as computer vision and unmanned aerial vehicles (UAVs) to optimize real-time traffic control and monitoring [2–4]. This section examines the state of the art in the various elements and factors that play a role in the problem, including UAVs, neural networks, edge computing, and datasets.

1.1.1. Unmanned Aerial Vehicles

There are two categories of unmanned aerial vehicles (UAVs or drones) based on the level of autonomy during operation: those that require constant piloting and autonomous ones that do not need human intervention for most of their operation [5].

The use of drones began in the early 21st century for military and security purposes [6–8]. These early drones were simple in construction, small in weight, and equipped with motors that allowed them to fly remotely and autonomously for several hours [8,9]. Subsequently, drones began to be commercialized, opening a range of possibilities [6,8,9].

Currently, drones are employed in various productive sectors, including agriculture, construction, maintenance, the environment, mining, filming, and insurance. Although a wide range of applications has already been explored, technological advancements continue to drive the development of drone use in other sectors and tasks such as emergency assistance, package transportation, and traffic management (the focus of this work), among many other potential applications [10,11].

Lastly, being such a powerful and versatile technology, security and privacy are crucial aspects of drone operation. Therefore, robust authentication protocols must be implemented to ensure safe operation within the drone network, known as the Internet of Drones (IoD) [6,12].

1.1.2. Object Recognition with UAVs

In the context of this work, UAVs are utilized for real-time image capture and object inference through artificial intelligence algorithms. These applications can range from capacity management to traffic control and communication with autonomous vehicles for their safe circulation [13,14]. Comparing models within the latter use case is the main objective of this study.

Autonomous vehicles are expected to significantly reduce the number and severity of accidents. However, intersections and roundabouts pose a challenge for these vehicles due to their complexity and variability [15–17]. Scientists and companies have proposed various solutions to address this issue, with some relying on algorithms to predict driver intentions, while others make use of control units installed at intersections to communicate the current situation to vehicles [18,19].

A recent study demonstrated that the use of drones equipped with cameras to capture driver movements at intersections is more effective than other methods of collecting such information [20]. There are also studies that show that using drones equipped with high-resolution cameras to capture driver movements at intersections is superior to other methods of recognizing this information [21].

1.1.3. Datasets

Research conducted by Milić et al. and Krajewski et al. examined the requirements for gathering a dataset of vehicle trajectories [8,22]. We can extrapolate these requirements to our goal, which is object recognition in images captured by drones. The criteria include the following:

- **Dataset size**: It is essential for the dataset to contain many images with a wide variety of labeled objects within these images;
- **Diversity of locations and time frames**: The images used to train the model should be taken at different locations and under various visibility conditions. This helps prevent overfitting, enabling the model to be effective in a variety of contexts;
- **Recognition of a wide range of objects**: When labeling the images, we should not exclude objects related to those we want to predict. For example, if we are labeling all cars, we should not exclude trucks from the dataset. We can group all objects into a category like "vehicles" or create a category for each type of object.

To meet these requirements, this work utilizes two different datasets that provide a substantial amount of data, a diverse range of scenarios, and recognition of numerous objects within each image [23,24].

1.1.4. Neural Networks for Object Detection

Within the field of object detection, there are various architectures, and some of the most popular ones today include the following:

- **YOLOv5**: This model is the fifth version in the YOLO (You Only Look Once) series and has been widely used for real-time object detection. One of the main advantages of YOLOv5 is its speed, making it ideal for real-time applications. For example, it has been used for real-time face mask detection during the COVID-19 pandemic, demonstrating its utility in real-world situations where speed is essential [25,26];
- **YOLOv8**: This is an enhanced version of YOLOv5, which has achieved even more impressive results in terms of speed and accuracy. A recent study introduced a new model called DEYO, which combines YOLO with DETR (DEtection TRansformer) to improve object detection [27,28];
- **EfficientDet**: This model is known for its balance between efficiency and performance in object detection [29].
- **DETR (DEtection TRansformer)**: This model has revolutionized the field of object detection as the first end-to-end object detector. Although its computational cost is high, it has proven to be very effective in real-time object detection [30].

These models represent significant advancements in object detection and have laid the foundation for future research and developments in this field.

1.1.5. Cloud Computing and Edge Computing for Traffic Management

Cloud computing architecture is characterized by the centralized processing of data on remote servers hosted in cloud data centers. This approach offers many advantages, including great flexibility and scalability, ease of management (as service providers handle infrastructure maintenance), and ample storage capacity. However, it also presents challenges such as data transfer latency, which is not suitable for applications requiring real-time responses, and data security concerns not only in storage but also in data transfer to/from the cloud. Additionally, cloud services can lead to significant long-term costs for continuous and extensive usage [31].

On the other hand, edge computing [32] relies on decentralized data processing on local devices, close to where data are generated. This architecture offers benefits such as the low latency that is critical for real-time applications as well as data privacy and security, which is especially important for sensitive data applications. However, it also comes with disadvantages, including limited resources—edge devices often have limited computing and storage resources compared to cloud servers—more complex management that requires detailed configuration and maintenance attention, and limited storage capacity, with edge devices typically offering minimal storage capacity.

Currently, the trend is to blend these two architectures either in different stages of the process (e.g., training and inference in AI) [33] or in a combined coexistence in the solution deployment, where data are processed locally, but the results of processing are stored in the cloud [34,35].

Indeed, the use of edge computing in drones for traffic management is a promising application and offers significant advantages compared to a centralized cloud-based approach. Below are some reasons why edge computing is beneficial in this context:

- **Low latency**: In traffic management, latency is critical. Drones need to make real-time decisions to avoid collisions and maintain efficient traffic flow. Edge computing allows drones to process data locally, significantly reducing latency compared to sending data to a distant cloud for processing;
- **Enhanced security**: By processing data locally on the UAVs themselves, dependence on internet connectivity is reduced, decreasing exposure to potential network interruptions or cyberattacks. This increases security in air traffic management;
- **Distributed scalability**: Using multiple drones equipped with edge computing allows for distributed scalability. This means that more drones can be added to address areas with dense traffic or special events without overburdening a central infrastructure;
- **Data privacy**: Air traffic management may involve the collection and transmission of sensitive data. Edge processing ensures that the data remain on the drones, improving privacy and complying with data privacy regulations;

- **Energy efficiency**: Transmitting data to the cloud and waiting for results can consume a significant amount of energy. Local processing on the drones is more energy-efficient, prolonging battery life and drone autonomy.

However, there are also challenges associated with using edge computing in drones for traffic management, such as the need for a robust network infrastructure for effective communication between drones and coordination. Additionally, managing and updating multiple edge devices can be more complex than managing a centralized system.

1.2. Research Gap

Despite the advancements in autonomous vehicle technology and computer vision systems, there are still significant challenges in the field. One of the major issues is the ability of autonomous vehicles to navigate safely and efficiently in complex urban environments such as intersections and roundabouts. These situations present dynamic and variable traffic conditions that demand fast and precise decision making, which is difficult to achieve with current solutions [15,16]. The state of the art reveals that, although several solutions have been explored, there is still a lack of effective solutions that optimally integrate computer vision with UAVs to enhance autonomous vehicle navigation. Existing computer vision algorithms and models have limitations in terms of accuracy, robustness, and efficiency. Furthermore, the selection and optimization of low-power hardware for implementing these models also present challenges.

The rapid technological evolution has facilitated the development of autonomous vehicles, promising significant benefits in terms of road safety, efficient use of transportation infrastructure, and accessibility [15,36]. However, the realization of this potential is hindered by various challenges, especially those related to autonomous navigation in complex urban environments [15,16]. Despite advancements in autonomous vehicle technology, making safe and efficient decisions in complex traffic situations like intersections and roundabouts remains a challenge [15,16].

Therefore, there is a need to develop and compare different computer vision models and low-power hardware to optimize the collaboration between UAVs and autonomous vehicles. The goal is not only to enhance the capabilities of autonomous vehicles but also to contribute to the field by providing new knowledge and solutions that could be applicable to other contexts and challenges in computer vision and autonomous vehicles.

1.3. Aim of the Study

This work arises in response to the challenges outlined in Section 1.2, aiming to enhance the navigation of autonomous vehicles through the utilization of computer vision and unmanned aerial vehicles (UAVs). The perspective offered by a UAV provides an aerial view, which can significantly enhance the perception and comprehension of the environment for autonomous vehicles [37]. Furthermore, real-time vision can provide up-to-date information on traffic conditions, enabling autonomous vehicles to make more informed decisions [14].

Another justification lies not only in the need to enhance the safety and efficiency of autonomous vehicles but also in the potential of computer vision and UAVs to achieve this. Despite prior research in this field, there is still ample room for innovation and improvement. This work focuses on the development and comparison of different computer vision models and low-power hardware to optimize the collaboration between UAVs and autonomous vehicles.

The outcomes of this research could be used by autonomous vehicle manufacturers to improve their navigation systems, by transportation authorities to enhance traffic management, and by computer-vision software developers to refine their algorithms and models. Ultimately, this work contributes to the field of study by exploring and comparing various approaches to computer vision in UAVs, providing a valuable addition to existing knowledge in this field.

2. Materials and Methods

2.1. Study Design

This work encompasses the details of a study that was designed based on several stages (see Figure 1), covering the entire process of creating a computer-vision system using low-power computers embedded in drones to provide "edge computing" capabilities. This involves everything from data or hardware selection to deployment and validation on the boards. Specifically, the following three stages are herein defined:

1. Hardware, software, and dataset selection;
2. Dataset construction and cleaning;
3. Experimentation:
    a Dataset preprocessing for training optimization;
    b Training with preprocessed/original datasets;
    c Validation of deployment results;
    d Measurement of energy consumption during deployment.

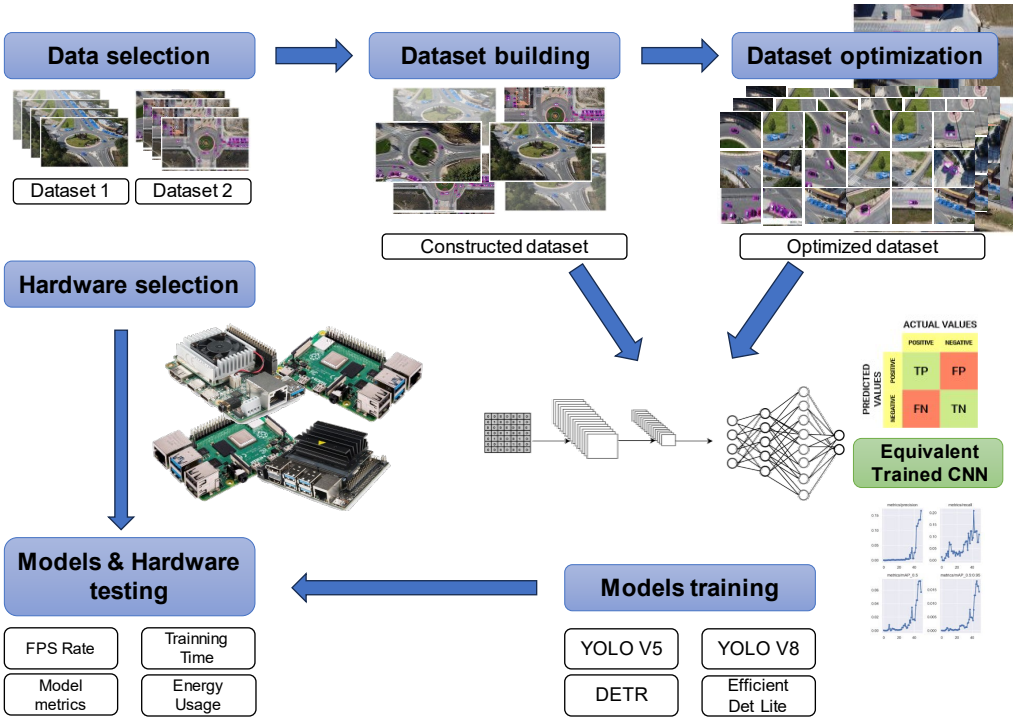


Figure 1. Experiment design, with all its phases.

2.2. Hardware, Software and Datasets

Different resources, both hardware and software, were utilized for the execution of the experiments. These resources are described in this section.

2.2.1. Reduced-Board Computers

Regarding hardware, various widely used low-power computer systems were employed, as can be observed in Table 1. In this table, in addition to technical specifications, there is a column indicating the weight in operational order (including microSD card, etc.). The amount of RAM on these single-board computers for neural network processing is of varying importance depending on the board and the network observed. Thus, for example, the Raspberry Pi 3B+ board is the oldest, and only the 1 GB (DDR2) RAM version is available. For this case, the RAM is important since the image processing is carried out in the microprocessor and the main memory, and the better these are, the better the results we can obtain, but this board is only manufactured with this configuration. The Google Coral device also comes in a 4 GB variant, which would enhance results in networks requiring

more RAM (like YOLO) but would not have as great an impact on those utilizing the TPU in processing, such as the EfficientDetLite models (see Section 3: Results and Section 4: Discussion for more details). The 1 GB version of the Google Coral device was used in our experiments, as it was the one available at the time of conducting the experiments.

Table 1. Selected reduced-board computers. Technical specifications and weight.

Board	RAM	CPU	GPU	Peso
Raspberry Pi 3B+	1 GB DDR2	64-bit @ 1.4 GHz	VideoCore IV 400 MHz	107 g
Raspberry Pi 4	4 GB DDR4	Quad-core 64-bit @ 1.8 GHz	VideoCore VI	107 g
Jetson Nano	4 GB DDR4	Quad-core MPCore processor	128 NVIDIA CUDA cores	243 g
Google Coral	1 GB DDR4	Quad Cortex-A53, Cortex-M4F	Integrated GC7000 Lite TPU coprocessor:	161 g

2.2.2. Software

Regarding software, different programming languages, development environments, and libraries were used for the development of computer vision systems:

- **VSCode** was used as an integrated development environment (IDE) for code development;
- **Anaconda** was used as a package manager and environment manager for Python;
- **Python** was the programming language used in the implementation of algorithms;
- **TensorFlow** is an open-source framework for machine learning and neural networks;
- **PyTorch** is a machine learning library also used in the implementation of algorithms;
- **RoboFlow** was utilized for dataset image management and preprocessing.

2.2.3. High-Performance Computing

In terms of high-performance computing, cloud computing resources were employed. Specifically, the cluster at the European University of Madrid (UEM) was used for machine learning model training. This cluster consists of 10 network-connected nodes, each configured with 32 cores, 256 GB of RAM, and NVIDIA RTX 3080Ti GPUs with 12 GB of RAM.

2.2.4. Datasets

Regarding data, two datasets of aerial images were used, both developed by the Intelligent Control Systems (SIC) group at UEM: (1) "Traffic Images Captured from UAVs for Use in Training Machine Vision Algorithms for Traffic Management" [23] and (2) "Roundabout Aerial Images for Vehicle Detection" [24].

2.3. Preparation of Objects and Materials

This section describes how the dataset was constructed and the approach to training and deploying models on reduced-board computers.

2.3.1. Dataset Generation

Two datasets created by the Intelligent Control Systems (SIC) Research Group at the European University of Madrid (UEM) were used and combined to create a new one. The first dataset, generated using the CVAT annotation tool [38], was documented in the article titled "Traffic Images Captured from UAVs for Use in Training Machine Vision Algorithms for Traffic Management" [23]. The second dataset, "Roundabout Aerial Images for Vehicle Detection" [24], was annotated using the PASCAL VOC XML technique, unlike the first dataset, which used YOLO annotation. The first dataset contains 15,070 images, and the second contains 15,474, resulting in a combined dataset of 30,544 images. Table 2 provides a breakdown of the objects (car and motorcycle classes) found in each of the datasets.

Table 2. Breakdown of the datasets used [23,24].

Dataset	Cars	Bikes	Total
Traffic Images Captured from UAVs for Use in Training Machine Vision Algorithms	137,602	17,726	155,328
Roundabout Aerial Images for Vehicle Detection	236,850	4899	241,749
Total	**374,452**	**22,625**	**397,077**

To unify these datasets into a common format, a Python script was created to convert PASCAL VOC XML annotations to the YOLO format. For dataset partitioning, four main strategies were explored:

1. **Random distribution:** This strategy involves dividing the dataset randomly, without considering the relationship between frames;
2. **Frame reservation by video:** In this approach, 20% of the frames from the same video were set aside for testing and validation, ensuring that temporal coherence in the training data is maintained;
3. **Selection of final frames:** This strategy involves reserving the final 20% of frames from each video, as these may contain more challenging situations for computer vision models;
4. **Frame selection per second:** This strategy, also known as subsampling, involves retaining only 1 frame out of every 24, equivalent to one frame per second, and then using random distribution for data partitioning.

This variety of approaches provides a wide range of test scenarios and enables a more robust and representative evaluation of the computer vision models being developed. After various tests and research, the decision was made to opt for the last approach (frame selection per second), as it, despite providing the fewest images for training, also exhibited the best model generalization. This resulted in 1269 images for training: 625 from the first dataset, and 644 from the second.

To facilitate data management, RoboFlow [39], a tool that allows for advanced preprocessing such as image augmentation was used. Image augmentation generally helps models learn more effectively, achieving better results [40,41]. Image augmentation techniques were chosen while preserving image coherence and included the following:

- 90° clockwise and counterclockwise rotation of the image;
- 45° clockwise and counterclockwise rotation of the image;
- 90° clockwise and counterclockwise rotation of objects within the image.

The final image number was thus increased to 3033 (1493 from the first dataset and 1540 from the second). These images are of high quality and incorporate a large number of targets (37,289 vehicles and 2253 motorcycles), making them suitable for training algorithms that can be executed on the selected hardware.

2.3.2. Model Training

The training process for artificial intelligence models was conducted using the cluster at the European University of Madrid, which provided the necessary computational resources for high-performance model training. Two frameworks were employed:

1. **TensorFlow**: An open-source framework developed by the Google Brain Team, TensorFlow is widely used in various fields that require intensive computation operations and has become a standard in the machine learning and artificial intelligence field [42]. TensorFlow was used to implement and train EfficientDet-Lite architectures (see Figure 2), which are object detection models known for their efficiency and performance in terms of speed and accuracy [25]. These models were specifically selected for their compatibility with the chosen low-power computers, including Raspberry Pi 3B+, Raspberry Pi 4, Google Coral Dev Board, and Jetson Nano;

2. **PyTorch**: Another open-source machine learning framework primarily developed by Facebook's artificial intelligence research group, PyTorch is known for its user-friendliness and flexibility, allowing for more intuitive development and easier debugging of machine learning models. PyTorch was used to train models with the YOLO and DETR architectures. YOLO is a popular real-time object detection algorithm known for its speed and accuracy. Unlike other object detection algorithms, which analyze an image in multiple regions and perform object detection in each region separately, YOLO conducts object detection in a single pass, making it particularly fast and suitable for real-time applications [43]. On the other hand, DETR is an architecture developed by Facebook AI that allows for using transformers to train object detection models [30].

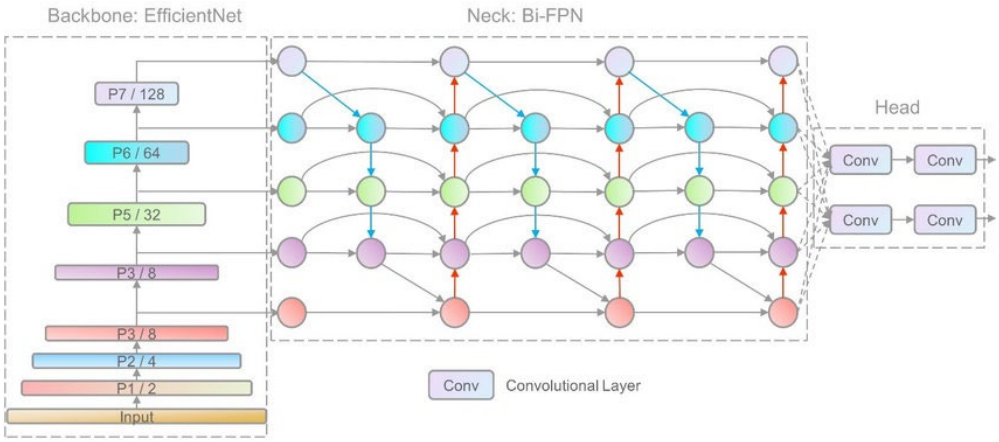


Figure 2. EfficientDet architecture [44].

2.3.3. Model Deployment on Reduced-Board Computers

The selected reduced-board computers for this study include a Raspberry Pi 3B+, a Raspberry Pi 4, a Google Coral Dev Board, and a Jetson Nano (see Figure 3). These devices, chosen for their compact size and energy efficiency, have certain hardware limitations. Therefore, it is crucial for the models developed not only to be accurate but also to have fast inference capability and perform well on the different boards.

Figure 3. Reduced-board computers used in the experiments. In the case of the Raspberry Pi 4, the variant based on a mini-PC was used, but for embedding in a UAV, the version consisting of the board alone was used.

The model deployment process included compiling the models for each of the different boards and evaluating their performance on these devices. To achieve this, a script was developed to measure the frames per second (FPS) at which the board could perform real-time video inference. Depending on the board's architecture, the model had to be consistent. For instance, to infer on the Google Coral, the models needed to be compiled to run on the tensor processing unit (TPU). For other platforms, such as Raspberry Pi, the models had to be compiled in a lightweight version compared to the original model (lite version). This phase presented a significant challenge due to the differences among the boards and the specific adjustments required for each of them.

2.4. Experiments

In this section, we describe the four experiments conducted in this work, which span from training times to equipment energy consumption, as well as model metrics and processing capabilities once the models are deployed on low-power computers.

2.4.1. Training Time

Starting with the dataset created as described in "Dataset generation" (Section 2.3.1), two training processes were performed for each model. For this purpose, two different input options were used: (1) the original dataset and (2) the same dataset optimized before training. For the second case, the training optimizer developed in the paper titled "Optimization Algorithm to Reduce Training Time for Deep Learning Computer Vision Algorithms Using Large Image Datasets with Tiny Objects" [45] was used. The use of this procedure was due to the dataset in this study fitting perfectly with the constraints and conditions of this algorithm. These constraints, in summary, are as follows [45]:

- Large images such as FullHD, 2K, 4K or even larger and with small objects or "targets" to detect considering the size of the image;
- Images taken at short intervals;
- Few objects within the image, or the objects are not evenly distributed within the image;
- There are static objects of interest in the image.

The training of the two models using both options yielded the time required for these trainings for a specific number of epochs, specifically 25 epochs.

2.4.2. Model Metrics

To evaluate the training of the models, three metrics were used: precision, recall, and mean average precision. These metrics were chosen because they are the most used within the field of object detection in images [46–48]. The definition of "precision" is the percentage of true positives obtained [48]. The formula is as follows, where TP represents true positives, and FP represents false positives:

$$P = \frac{TP}{TP + FP} \tag{1}$$

On the other hand, "recall" is a metric that analyzes the percentage of true positives compared to the number of real positives [48]. Its formula is as follows:

$$R = \frac{TP}{TP + FN} \tag{2}$$

Finally, "mean average precision" (mAP) is used to measure the average precision of detections across all classes [46,48]. It defines APi as the average precision of the nth class and N as the total number of classes evaluated [46]. The formula is as follows:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{3}$$

Figure 4 illustrates these previously explained concepts. Finally, to understand the results, it is important to explain the concept of "intersection over union" (IoU). This ratio is used to determine if a prediction is a true positive or a false positive. It is defined as the overlap between the bounding box inferred by the model and the ground truth bounding box, divided by their union (see Figure 5). An IoU of 0.5 was used, which is one of the standards [46,48].

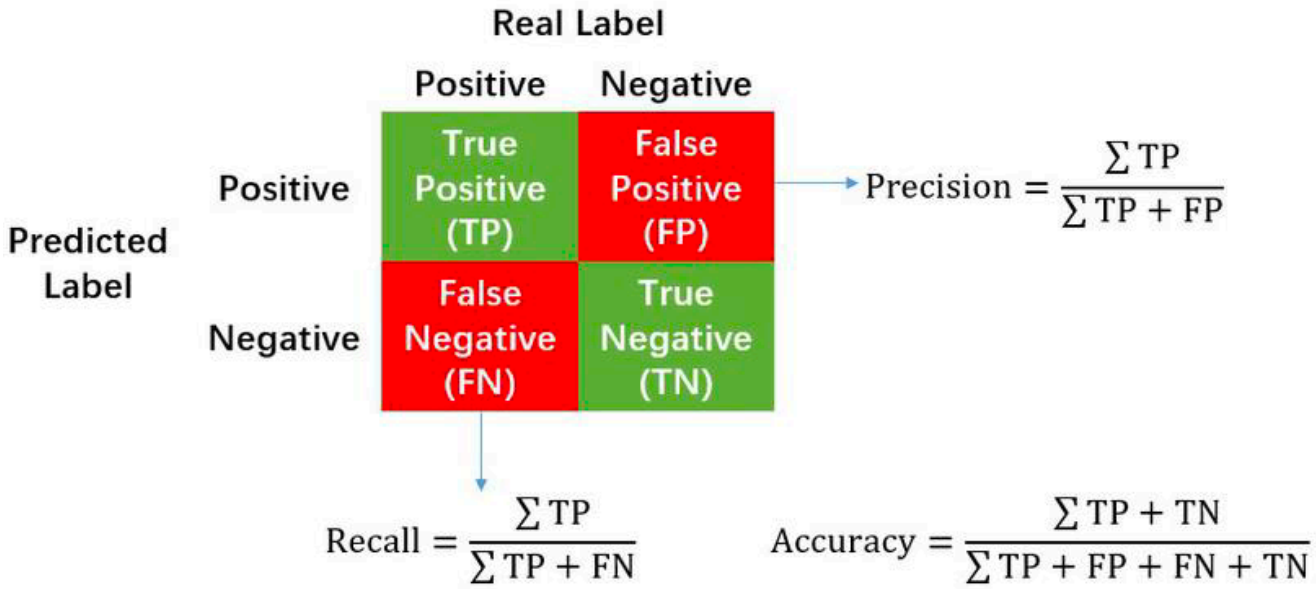


Figure 4. Visual explanation of the metrics [46].

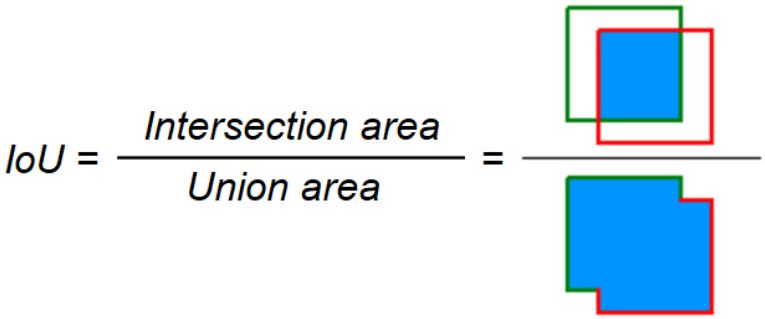


Figure 5. Intersection over union (IoU) equation [48].

2.4.3. Deployment Metrics

Once the models were deployed on the different single-board computers, their performance on these devices was assessed. The metric used for this assessment was "frames per second" (FPS). This metric indicates how many images per second the board can process when applying the model. The higher this number, the closer our product is to real-time inference. In the case of the Google Coral, models with architectures that are not compatible with this board were not tested.

2.4.4. Power Consumption

Once the models were deployed on the different single-board computers, their power consumption was measured. The metric used was watts (W), which indicates how much power each board consumes when running the model. The higher this number, the more it affects the UAV's autonomy, as it consumes part of the energy needed to keep it in flight.

3. Results

In this section, the results of the four experiments described in "Experiments" (Section 2.4) are presented.

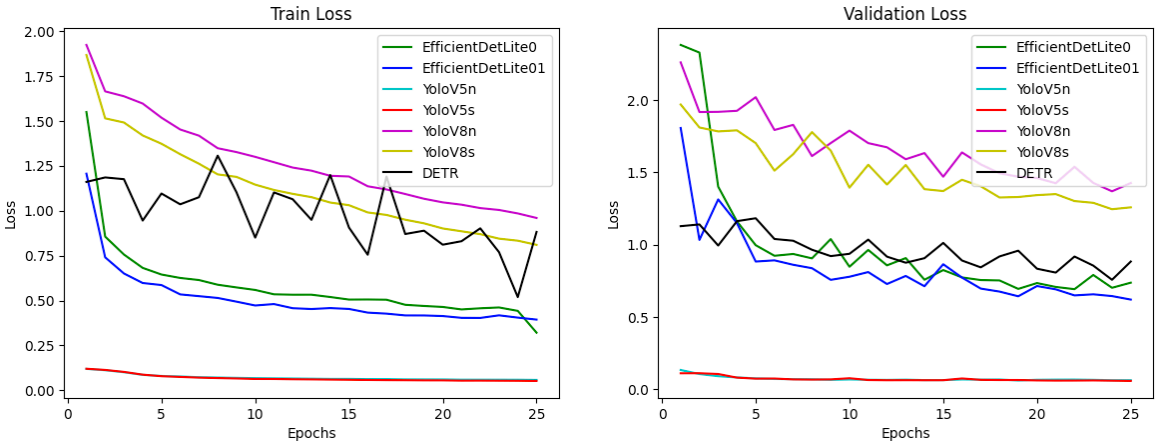
3.1. Training Times

Table 3 displays the training times of the different models mentioned that were obtained for each model with and without optimized data.

Table 3. Training times with and without data optimization.

Model	Epochs	Time without Optimization	Time with Optimization	Time Saving
YoloV5n	20	4 h 44 m 40 s	47 m 35 s	16.72%
YoloV5s	20	7 h 18 m 50 s	1 h 5 m 25 s	14.91%
YoloV8n	20	5 h 3 m 20 s	1 h 17 m 20 s	25.49%
YoloV8s	20	7 h 45 m 45 s	1 h 37 m 45 s	20.99%
DETR	20	19 h 45 m	23 h 35 m	**119.41%**
EfficientDetLite0	20	2 h 7 m 35 s	2 h 44 m	**128.54%**
EfficientDetLite1	20	2 h 41 m 45 s	3 h 48 m 35 s	**141.32%**

Figure 6 shows the evolution of metrics throughout the different epochs. It is shown in both Figure 6a training and Figure 6b validation.



(**a**) (**b**)

Figure 6. Metrics evolution over the 25 epochs during (**a**) training and (**b**) validation.

3.2. Model Metrics

Table 4 shows the selected metrics for each of the models.

Table 4. Metrics of the models (accuracy, recall and mAP).

Model	Precision	Recall	mAP50	mAP50-95
YoloV5n	72.9%	19.4%	16.2%	3.3%
YoloV5s	46.3%	31.4%	26.9%	6.5%
YoloV8n	83.3%	72.6%	80%	44.9%
YoloV8s	84.8%	70.1%	77.4%	44.0%
DETR	-	28.6%	55.5%	21.2%
EfficientDetLite0	-	19.3%	27.4%	7.6%
EfficientDetLite1	-	23.5%	36.8%	10.5%

3.3. Deployment Metrics

Table 5 provides a detailed overview of the results obtained after deploying the models on the different single-board computers. Specifically, it presents the average FPS obtained on each device compared to the UEM cluster as the reference equipment.

Table 5. Average FPS achieved on different single-board computers by model.

Model	Cluster UEM (FPS)	Raspberry Pi 3B+ (FPS)	Raspberry Pi 4 (FPS)	Jetson Nano (FPS)	Google Coral (FPS)
YoloV5n	130.44	0.46	1.3	14.7	-
YoloV5s	114.72	0.19	0.73	4.8	-
YoloV8n	75.08	0.27	0.76	6.2	-
YoloV8s	72.24	0.09	0.44	3.3	-
DETR	12.26	0.01	0.05	0.03	-
EfficientDetLite0	9.08	1.14	2.92	2.04	6.7
EfficientDetLite1	4.7	0.58	1.63	1.14	5.4

3.4. Energy Consumption

Table 6 provides a breakdown of the results obtained for power consumption.

Table 6. Power consumption of the reduced-board computers.

Reduced-Board Computer	Idle			Execution		
	Voltage	Current	Power	Voltage	Current	Power
Raspberry Pi 3B+	5.2 V	0.45 A	2.34 W	5.2 V	0.79 A	4.1 W
Raspberry Pi 4	5.4 V	0.35 A	1.89 W	5.4 V	0.66 A	3.56 W
Jetson Nano	5.4 V	0.78 A	4.2 W	5.4 V	1.9 A	10.2 W
Google Coral	5.2 V	0.95 A	4.94 W	5.2 V	1.2 A	6.24 W

4. Discussion

Given the results obtained, determining the best hardware–software solution for implementing onboard processing in UAVs for intelligent traffic management is challenging. It heavily depends on specific needs and budget constraints. In this chapter, we describe some characteristics of the best combinations found from a hardware perspective. It is important to note that the results obtained from cluster execution are not relevant since there is a clear disparity in computational processing power. This high-performance cluster was used for the training of the different models, a task impossible to perform with the single-board computers. In addition, the advantage of using this equipment compared to a conventional PC is that it greatly reduces the time required for this phase. Given its high capacity, it was also used to measure the FPS in deployment of the different models. This helped us to establish a baseline in the comparison with the performance of the rest of the boards.

The metrics obtained in the cloud only served as an index of the correct functioning of the networks and were not used for direct comparison. With that in mind, we can evaluate the solutions as follows:

- **Raspberry Pi:** This is one of the least-power-consuming solutions as well as being the lightest hardware; it can run neural networks of various types, although its FPS is one of the slowest. Raspberry Pi would be the best choice for PyTorch -type networks, such as Yolo (YoloV5n, YoloV5s, YoloV8n, and YoloV8s), where power consumption and weight are critical, for example, to be employed in UAVs. If, in addition, we can work with small images such as those for classification (and not recognition), the Raspberry Pi is the best choice;
- **Jetson Nano:** it is the most powerful option in PyTorch -like network processing but outperforms the Raspberry Pi in power consumption and weight. The number of FPS is considerably higher, which makes it a better choice if (1) processing is a key factor,

such as for object recognition, which needs in an image instead of classification, as it not only processes faster but also performs better with larger images, and if (2) power consumption and weight are not a critical factor, such as for images in static cameras (gantries or surveillance cameras);

- **Google Coral:** This hardware is one of the most powerful in processing capacity, with a slightly higher power consumption than the rest of the boards and with a weight between that of the two previous boards. However, this board has an important limitation: it has no GPU but TPUs (tensor processing unit), which makes it very inefficient for PyTorch-type networks but extremely efficient for TensorFlow networks such as Effi-cientDetLite0 or Effi-cientDetLite1 networks. The FPS difference is 500% faster compared to its competitors, which makes it the most suitable board when processing time is critical, and its weight makes it a good choice for onboard UAVs, while its power consumption greatly limits its working autonomy.

Another crucial factor to consider is the inference results. In this aspect, it is relatively straightforward to draw conclusions about the best options. On one hand, YOLO-based models such as YOLOv5n, YOLOv5s, YOLOv8n, and YOLOv8s demonstrate a trend towards higher precision and mAP50 compared to other architectures. YOLOv8n and YOLOv8s, in particular, stand out with precisions of 83.3% and 84.8%, respectively, and mAP50 values of 80.0% and 77.4%. However, there is some variability in recall, indicating that these models might struggle to detect all relevant objects in certain circumstances.

In contrast, the DETR model shows a recall of 28.6% and an mAP50 of 55.5%, which, while lower than the mentioned YOLO versions, still represents respectable performance. The EfficientDetLite0 and EfficientDetLite1 models have lower recall values and mAP50 scores of 27.4% and 36.8%, respectively.

However, it is important to interpret these data correctly. Remember that the images being processing are of HD size (1920 $\times$ 1080 px) and the objects to recognize range from medium-sized (trucks = 200 $\times$ 200 px) to small-sized (pedestrians = 30 $\times$ 30 px). For YOLO-based networks, the results are accurate, with precision around 84%. In contrast, the DETR network, although it has a lower recall rate mainly due to not detecting many objects, is influenced by the image manipulation before being processed by the network. The image is resized to 640 $\times$ 640 pixels, which is too small for some targets (pedestrians and motorcycles) that need to be detected. This resizing causes these objects to be too small for the network to detect, while larger targets (cars, trucks, etc.) remain recognizable.

On the other hand, TensorFlow-based networks (EfficientDetLite0 and EfficientDetLite1) face a similar issue but to a greater extent due to the network's architecture. These models cannot process images larger than 640 $\times$ 640 on Google Coral due to memory limitations. Consequently, their effectiveness in inference is significantly lower because the network was trained with original-sized images.

It is worth noting that this article focuses on the observation and surveillance of traffic from UAVs equipped with high-resolution cameras capturing objects at a certain distance. Therefore, the images that need to be inferred are similar to the images analyzed with the different boards. As a result, we can determine that Raspberry Pi or Jetson Nano with YOLO could process these images without the need for prior preprocessing and at high resolution. However, this would lead to a decrease in the frames per second (FPS) proportional to the input image size.

On the other hand, Google Coral with EfficientDetLite networks requires a preprocessing step involving tiling or subdividing the input images without reducing their size. This method allows for leveraging the high FPS of Google Coral, but processing a high-resolution image would involve breaking it into smaller images such as 3 $\times$ 3 or 4 $\times$ 4 sub-images. As a result, the calculated FPS would need to be adjusted accordingly to account for the processing of the original high-resolution images.

To consolidate part of the data obtained in the experiment, we created a new table that represents power consumption per inferred frame (Table 7). It is essential to generate a more comprehensive table that also considers the energy consumption of a UAV due to the

added hardware weight. However, this calculation is highly dependent on the type and configuration of the UAV and falls outside the scope of this experiment.

Table 7. Relationship between power (watts) consumed by the reduced-board computers per FPS inferred for the differently constructed models. The results are displayed with a color gradient from green (best) to red (worst).

Model	Raspberry Pi 3B+	Raspberry Pi 4	Jetson Nano	Google Coral
YoloV5n	8.93	2.74	**0.70**	-
YoloV5s	21.62	4.88	2.14	-
YoloV8n	15.21	4.69	1.65	-
YoloV8s	45.64	8.10	3.11	-
DETR	**410.80**	71.28	342.00	-
EfficientDetLite0	3.60	1.22	5.03	0.93
EfficientDetLite1	7.08	2.19	9.00	1.16

From this comparison of watts-FPS, it can be seen that the best board/model combination is Jetson Nano with Yolo, followed by Google Coral with EfficientDetLite. In contrast, the DETR configuration with Raspberry Pi 3B+ is the worst by a significant margin. In fact, DETR consistently yields the worst results overall.

The amount of RAM (random access memory) in a computer can indeed affect both computation time and power consumption, although the impact may vary based on the specific tasks and applications being run.

If we focus the analysis on the case described in this article, namely the processing time and energy consumption, RAM is an influential factor in all cases. However, in some instances, it has a lesser influence compared to others. Considering the case of the Intel Coral board running EfficientDetLite-type networks, the amount of memory needed is sufficient to support the operating system and applications supporting the neural network (communications, image capture, disk storage, etc.). This is because the image processing by the neural network is entirely carried out on the TPU, so the efficiency in image processing is summarized by the efficiency of this component. In this case, all Intel Coral boards have the same TPU regardless of the RAM they incorporate.

The same situation applies to the Jetson Nano board running PyTorch-type networks (like YOLO), as these networks run entirely on the board's GPU. Once again, all Jetson Nano cards incorporate the same GPU. In contrast, boards without specific GPUs or TPUs, such as Raspberry Pi 3 and 4, process information in the main microprocessor and memory. For these boards, it is a very relevant factor since these hardware resources are not only allocated to image processing by the neural network but are also shared with the operating system and other tasks of running applications. For these cards, RAM will significantly affect their efficiency, especially at the lower limit, as insufficient RAM will cause the operating system to utilize virtual memory, which involves storing data on the slower storage. Accessing data from virtual memory is much slower than accessing it from RAM, leading to increased computation time. On the contrary, an excess of unused RAM would result in higher energy consumption, but this consumption is negligible compared to the rest of the energy expenditures. Therefore, this factor is not considered in the analysis.

5. Conclusions

Various lines of work were conducted in this study, including a review of the state of the art, model training, and their implementation, among others. All of these efforts serve a common goal: to implement object recognition models on single-board computers and determine which combination of board and model is the most suitable for the specific use case of traffic management assistance, particularly for autonomous vehicles.

As demonstrated in the preceding sections, numerous models and boards were tested, each with its unique characteristics. The YOLO models, especially version 8, exhibited strong performance in terms of accuracy and acceptable detection speed. However, their

implementation was restricted on certain boards due to limitations inherent to their architectures, such as the Google Coral, which achieved the best processing speed but also presented the most limitations.

On the other hand, the EfficientDetLite models were successfully deployed on the Google Coral. While their model metrics were inferior to other models, they demonstrated superior inference capability thanks to their compatibility with a TPU version of the mentioned board.

These results underline the importance of considering both the performance metrics of object detection models and the limitations of the hardware during model selection for a computer vision project. Striking a balance between model performance and hardware compatibility is crucial for the successful execution of such projects.

Based on these results, two optimal model–board combinations were selected for the project's objectives. On one hand, a combination with better accuracy but lower inference speed includes the YoloV8n model with the Raspberry Pi 4. On the other hand, the combination with much higher inference speed but lower accuracy involves the EfficientDetLite1 model with the Google Coral board.

Author Contributions: Conceptualization, J.S.-S., S.G. and S.B.R.; methodology, J.S.-S. and S.G.; software, S.G. and S.B.R.; validation, J.S.-S., S.G. and S.B.R.; formal analysis, J.S.-S. and S.G.; investigation J.S.-S., S.G. and S.B.R.; resources, J.S.-S., S.G. and S.B.R.; data curation, S.G.; writing—original draft preparation, J.S.-S., S.G. and S.B.R.; writing—review and editing, J.S.-S., S.G., S.B.R. and J.F.-A.; visualization, S.G. and S.B.R.; supervision, J.S.-S.; project administration, J.S.-S.; funding acquisition, J.F.-A. All authors have read and agreed to the published version of the manuscript.

Funding: This work is part of the I+D+i projects with reference PID2019-104793RB-C32, PIDC2021-121517-C33, PDC2022-133684-C33, funded by MCIN/AEI/10.13039/501100011033.

Data Availability Statement: The data presented in this study are openly available in https://zenodo.org/record/5776219 (accessed on 1 November 2023) with doi: https://doi.org/10.3390/data7050053 [23] and https://zenodo.org/records/6407460 (accessed on 1 November 2023) with doi: https://doi.org/10.3390/data7040047 [24].

Acknowledgments: The authors would like to thank the Universidad Francisco de Vitoria and the Universidad Europea de Madrid for their support. In addition, special thanks are due to the Universidad Europea de Madrid for allowing us to use their high-capacity cluster for the training of the machine learning models.

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

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
CNN	Convolutional Neural Networks
COVID-19	COronaVIrus Disease 2019
CUDA	Compute Unified Device Architecture
CVAT	Computer Vision Annotation Tool
DDR	Double Data Rate
DETR	DEtection TRansformer
FP	False Positive
PFS	Frames Per Second
GPU	Graphics Processing Unit
HD	High Definition
IDE	Integrated Development Environment
IoD	Internet of Drones

IoU	Intersection over Union
mAP	mean Average Precision
microSD	micro Secure Digital
PASCAL	Pattern Analysis, Statistical modeling, and Computational Learning
PC	Personal Computer
RAM	Random Access Memory
TP	True Positive
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicle
VOC	Visual Object Classes
XML	Extensible Markup Language
YOLO	You Only Look Once

References

1. Pettersson, I.; Karlsson, I.C.M. Setting the stage for autonomous cars: A pilot study of future autonomous driving experiences. *IET Intell. Transp. Syst.* **2015**, *9*, 694–701. [CrossRef]
2. Yildiz, M.; Bilgiç, B.; Kale, U.; Rohács, D. Experimental Investigation of Communication Performance of Drones Used for Autonomous Car Track Tests. *Sustainability* **2021**, *13*, 5602. [CrossRef]
3. Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, J.; Liu, X.; Pietikäinen, M. Deep Learning for Generic Object Detection: A Survey. *Int. J. Comput. Vis.* **2020**, *128*, 261–318. [CrossRef]
4. Menouar, H.; Guvenc, I.; Akkaya, K.; Uluagac, A.S.; Kadri, A.; Tuncer, A. UAV-enabled intelligent transportation systems for the smart city: Applications and challenges. *IEEE Commun. Mag.* **2017**, *55*, 22–28. [CrossRef]
5. Ahmed, F.; Jenihhin, M. A Survey on UAV Computing Platforms: A Hardware Reliability Perspective. *Sensors* **2022**, *22*, 6286. [CrossRef] [PubMed]
6. Johnston, R.; Hodgkinson, D. *Aviation Law and Drones Unmanned Aircraft and the Future of Aviation*; Routledge: London, UK, 2018.
7. Merkert, R.; Bushell, J. Managing the drone revolution: A systematic literature review into the current use of airborne drones and future strategic directions for their effective control. *J. Air Transp. Manag.* **2020**, *89*, 101929. [CrossRef] [PubMed]
8. Milic, A.; Ranđelović, A.; Radovanović, M. Use of Drons in Operations in The Urban Environment. Available online: https://www.researchgate.net/profile/Marko-Radovanovic-2/publication/336589680_Use_of_drones_in_operations_in_the_urban_environment/links/60d2751845851566d5839b29/Use-of-drones-in-operations-in-the-urban-environment.pdf (accessed on 12 September 2023).
9. Vaigandla, K.K.; Thatipamula, S.; Karne, R.K. Investigation on Unmanned Aerial Vehicle (UAV): An Overview. *IRO J. Sustain. Wirel. Syst.* **2022**, *4*, 130–148. [CrossRef]
10. Plan Estratégico para el Desarrollo del Sector Civil de los Drones en España 2018–2021 | Ministerio de Transportes, Movilidad y Agenda Urbana. Available online: https://www.mitma.gob.es/el-ministerio/planes-estrategicos/drones-espania-2018-2021 (accessed on 11 June 2023).
11. Lee, H.S.; Shin, B.S.; Thomasson, J.A.; Wang, T.; Zhang, Z.; Han, X. Development of Multiple UAV Collaborative Driving Systems for Improving Field Phenotyping. *Sensors* **2022**, *22*, 1423. [CrossRef] [PubMed]
12. Alsharif, H.; Khan, M.A.; Michailidis, E.T.; Vouyioukas, D. A Review on Software-Based and Hardware-Based Authentication Mechanisms for the Internet of Drones. *Drones* **2022**, *6*, 41. [CrossRef]
13. Wang, X.; Cheng, P.; Liu, X.; Uzochukwu, B. Fast and Accurate, Convolutional Neural Network Based Approach for Object Detection from UAV. In Proceedings of the IECON 2018—44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, 21–23 October 2018; pp. 3171–3175. [CrossRef]
14. Kyrkou, C.; Plastiras, G.; Theocharides, T.; Venieris, S.I.; Bouganis, C.-S. DroNet: Efficient Convolutional Neural Network Detector for Real-Time UAV Applications. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018.
15. Sánchez-Soriano, J.; De-Las-Heras, G.; Puertas, E.; Fernández-Andrés, J. Sistema Avanzado de Ayuda a la Conducción (ADAS) en rotondas/glorietas usando imágenes aéreas y técnicas de Inteligencia Artificial para la mejora de la seguridad vial. *Logos Guard. Civ. Rev. Cient. Cent. Univ. Guard. Civ.* **2023**, *1*, 241–270. Available online: https://revistacugc.es/article/view/5708 (accessed on 28 June 2023).
16. Cuenca, L.G.; Sanchez-Soriano, J.; Puertas, E.; Andrés, J.F.; Aliane, N. Machine Learning Techniques for Undertaking Roundabouts in Autonomous Driving. *Sensors* **2019**, *19*, 2386. [CrossRef] [PubMed]
17. Tang, H.; Post, J.; Kourtellis, A.; Porter, B.; Zhang, Y. Comparison of Object Detection Algorithms Using Video and Thermal Images Collected from a UAS Platform: An Application of Drones in Traffic Management. *arXiv* **2021**, arXiv:2109.13185.
18. Tobias, L.; Ducournau, A.; Rousseau, F.; Mercier, G.; Fablet, R. Convolutional Neural Networks for object recognition on mobile devices: A case study. In Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, Mexico, 4–8 December 2016; pp. 3530–3535. [CrossRef]

19. Akram, R.N.; Markantonakis, K.; Mayes, K.; Habachi, O.; Sauveron, D.; Steyven, A.; Chaumette, S. Security, privacy and safety evaluation of dynamic and static fleets of drones. In Proceedings of the AIAA/IEEE Digital Avionics Systems Conference, St. Petersburg, FL, USA, 17–21 September 2017. [CrossRef]
20. Peng, H.; Razi, A.; Afghah, F.; Ashdown, J. A Unified Framework for Joint Mobility Prediction and Object Profiling of Drones in UAV Networks. *J. Commun. Netw.* **2018**, *20*, 434–442. [CrossRef]
21. Bock, J.; Krajewski, R.; Moers, T.; Runde, S.; Vater, L.; Eckstein, L. The inD Dataset: A Drone Dataset of Naturalistic Road User Trajectories at German Intersections. In Proceedings of the IEEE Intelligent Vehicles Symposium, Las Vegas, NV, USA, 19 October–13 November 2020; pp. 1929–1934. [CrossRef]
22. Krajewski, R.; Bock, J.; Kloeker, L.; Eckstein, L. The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems. In Proceedings of the IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC, Maui, HI, USA, 4–7 November 2018; pp. 2118–2125. [CrossRef]
23. Ghisler, S.; Rosende, S.B.; Fernández-Andrés, J.; Sánchez-Soriano, J. Dataset: Traffic Images Captured from UAVs for Use in Training Machine Vision Algorithms for Traffic Management. *Data* **2022**, *7*, 53. [CrossRef]
24. Puertas, E.; De-Las-Heras, G.; Fernández-Andrés, J.; Sánchez-Soriano, J. Dataset: Roundabout Aerial Images for Vehicle Detection. *Data* **2022**, *7*, 47. [CrossRef]
25. Liu, R.; Ren, Z. Application of Yolo on Mask Detection Task. *arXiv* **2021**, arXiv:2102.05402.
26. Jocher, G.; Chaurasia, A.; Stoken, A.; Borovec, J.; Kwon, Y.; Michael, K.; Jain, M. Ultralytics/yolov5: v7.0—YOLOv5 SOTA Realtime Instance Segmentation. *Zenodo* **2022**.
27. Ouyang, H. DEYO: DETR with YOLO for Step-by-Step Object Detection. *arXiv* **2022**, arXiv:2211.06588.
28. Reis, D.; Kupec, J.; Hong, J.; Daoudi, A. Real-Time Flying Object Detection with YOLOv8. *arXiv* **2023**, arXiv:2305.09972.
29. Tan, M.; Pang, R.; Le, Q.V. EfficientDet: Scalable and Efficient Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020.
30. Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. *End-to-End Object Detection with Transformers*; Springer: Cham, Switzerland, 2020.
31. Yang, P.; Xiong, N.; Ren, J. Data Security and Privacy Protection for Cloud Storage: A Survey. *IEEE Access* **2020**, *8*, 131723–131740. [CrossRef]
32. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* **2018**, *6*, 6900–6919. [CrossRef]
33. Hua, H.; Li, Y.; Wang, T.; Dong, N.; Li, W.; Cao, J. Edge Computing with Artificial Intelligence: A Machine Learning Perspective. *ACM Comput. Surv.* **2023**, *55*, 1–35. [CrossRef]
34. Singh, S. Optimize cloud computations using edge computing. In Proceedings of the 2017 International Conference on Big Data, IoT and Data Science (BID), Pune, India, 20–22 December 2017; pp. 49–53. [CrossRef]
35. Kristiani, E.; Yang, C.T.; Huang, C.Y.; Wang, Y.T.; Ko, P.C. The Implementation of a Cloud-Edge Computing Architecture Using OpenStack and Kubernetes for Air Quality Monitoring Application. *Mob. Netw. Appl.* **2021**, *26*, 1070–1092. [CrossRef]
36. Deng, W.; Lei, H.; Zhou, X. Traffic state estimation and uncertainty quantification based on heterogeneous data sources: A three detector approach. *Transp. Res. Part B Methodol.* **2013**, *57*, 132–157. [CrossRef]
37. Zhou, Y.; Cheng, N.; Lu, N.; Shen, X.S. Multi-UAV-Aided Networks: Aerial-Ground Cooperative Vehicular Networking Architecture. *IEEE Veh. Technol. Mag.* **2015**, *10*, 36–44. [CrossRef]
38. CVAT Open Data Annotation Platform. Available online: https://www.cvat.ai (accessed on 12 October 2023).
39. Roboflow. Available online: https://roboflow.com/ (accessed on 5 October 2023).
40. Bloice, M.D.; Stocker, C.; Holzinger, A. Augmentor: An Image Augmentation Library for Machine Learning. *arXiv* **2017**, arXiv:1708.04680. [CrossRef]
41. Perez, L.; Wang, J. The Effectiveness of Data Augmentation in Image Classification Using Deep Learning. *arXiv* **2017**, arXiv:1712.04621.
42. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv* **2016**, arXiv:1603.04467.
43. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788. [CrossRef]
44. Xu, R.; Lin, H.; Lu, K.; Cao, L.; Liu, Y. A Forest Fire Detection System Based on Ensemble Learning. *Forests* **2021**, *12*, 217. [CrossRef]
45. Rosende, S.B.; Fernández-Andrés, J.; Sánchez-Soriano, J. Optimization Algorithm to Reduce Training Time for Deep Learning Computer Vision Algorithms Using Large Image Datasets with Tiny Objects. *IEEE Access* **2023**, *11*, 104593–104605. [CrossRef]

46. Hui, J. mAP (mean Average Precision) for Object Detection. Available online: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 (accessed on 12 September 2023).
47. Mariano, V.Y.; Min, J.; Park, J.-H.; Kasturi, R.; Mihalcik, D.; Li, H.; Doermann, D.; Drayer, T. Performance evaluation of object detection algorithms. In Proceedings of the 2002 International Conference on Pattern Recognition, Quebec City, QC, Canada, 11–15 August 2002; pp. 965–969. [CrossRef]
48. Padilla, R.; Netto, S.L.; da Silva, E.A.B. A Survey on Performance Metrics for Object-Detection Algorithms. In Proceedings of the 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), Niteroi, Brazil, 1–3 July 2020; pp. 237–242. [CrossRef]