Sustainable Computing: Informatics and Systems 34 (2022) 100725

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Low-power deep learning edge computing platform for resource constrained lightweight compact UAVs

Andrea Albanese *, Matteo Nardello, Davide Brunelli

Department of Industrial Engineering, University of Trento, Via Sommarive 9, Trento, 38123, Italy

ARTICLE INFO

Keywords:
Deep Neural Networks
UAV
Sustainable computing
Edge inference
Visual navigation

ABSTRACT

Unmanned Aerial Vehicles (UAVs), which can operate autonomously in dynamic and complex environments, are becoming increasingly common. Deep learning techniques for motion control have recently taken a major qualitative step since vision-based inference tasks can be executed directly on edge. The goal is to fully integrate the machine learning (ML) element into small UAVs. However, given the limited payload capacity and energy available on small UAVs, integrating computing resources sufficient to host ML and vehicle control functions is still challenging. This paper presents a modular and generic system that can control the UAV by evaluating vision-based ML tasks directly inside the resource-constrained UAV. Two different vision-based navigation configurations were tested and demonstrated. The first configuration implements an autonomous landing site detection system, tested with two models based on LeNet-5 and MobileNetV2, respectively. This allows the UAV to change its planned path accordingly and approach the target to land. Moreover, a model for people detection based on a custom MobileNetV2 network was evaluated in the second configuration. Finally, the execution time and power consumption were measured and compared with a cloud computing approach. The results show the ability of the developed system to dynamically react to the environment to provide the necessary maneuver after detecting the target exploiting only the constrained computational resources of the UAV controller. Furthermore, we demonstrated that moving to the edge, instead of using cloud computing inference, decreases the energy requirement of the system without reducing the quality of service.

1. Introduction

In recent years, considerable research has been conducted regarding the design, development, and operation of autonomous Unmanned Aerial Vehicles (UAVs). Multi-rotor UAVs are potentially useful in a wide variety of scenarios. They can work in extreme environments to monitor and explore areas hardly reachable by operators, and they can perform tasks such as rescue operations [1,2], wild monitoring [3–5], pest detection [6–8], telecommunications relay [9–11], border surveillance [12–14], and many others [15–20].

While UAV control technologies have progressed steadily in recent years, the main control methods are still remote radio and pre-programmed missions. Nevertheless, these fields of application impose big constraints for normal operation tasks such as taking-off, navigation, and environment interaction. To overcome these limitations, researchers have proposed different approaches to ease those tasks [21–24].

The advancement of compact and low-power computing platforms sufficiently portable to be deployed on lightweight UAVs has permitted real-time image processing with machine learning on these devices [25]. Machine learning models have been integrated on UAVs for many years, but only recently, Deep Neural Networks (DNNs) have been exploited for various autonomous UAV operations. DNN models exhibit "excellent capabilities for learning high-level representations from raw sensor data" [26]. Unfortunately, DNNs require training on large and specialized datasets to achieve good results in many applications [27], and moving these models to edge devices is still challenging.

Usually, UAVs have limited on-board computation resources, requiring a significant amount of time to process the acquired data and decreasing the UAV responsiveness to the external environment. Moreover, continuously running complex algorithms imposes a considerable energy expense which can affect the operational lifetime.

To overcome this limitation, researchers have proposed to exploit cloud computing to offload part of the computation [28,29]. In the literature, we can find different architectures [30,31] to relieve the resource constraints of UAV processing capabilities. These approaches enable UAVs to benefit from the redundant resources of modern data

* Corresponding author.
E-mail address: andrea.albanese@unitn.it (A. Albanese).

https://doi.org/10.1016/j.suscom.2022.100725
Received 15 June 2021; Received in revised form 10 February 2022; Accepted 5 March 2022
Available online 18 March 2022

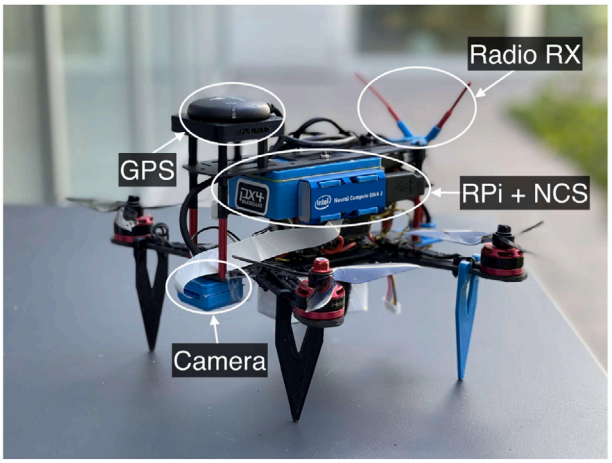


Fig. 1. Lightweight UAV prototype.

centers. However, even if cloud-based processing offloading have obtained some positive results [32], they have yet to achieve high Quality of Service (QoS) for resource-intensive applications mainly due to the following reasons:

- **Data transfer:** Data transmission from the UAV to the remote cloud can cause latency issues. This is because some UAV tasks are latency-intolerant. After all, they have to make near-real-time decisions, and even slight delays can cause dangerous consequences (i.e., collisions). Moreover, intensive data streaming causes long communication delay when the data is transferred to remote centralized data centers for processing.
- **Energy cost:** While computation offloading to cloud servers can reduce the energy consumption associated with data processing, it can increase the overall energy consumption due to the energy cost of transmitting the data. The problem can also worsen if the applications are communication-intensive or the UAV experiences poor network connectivity.

Therefore, in some conditions and locations, the time needed to transport data to the edge server and receive the analysis outcome may exceed the time with local processing at the UAV.

The challenge of optimizing resource utilization on resource-constrained devices has also been tackled for scalable IoT applications. Researchers [33–36] have proposed to move part of data analysis near the "sensor" where the data is being collected, using an edge computing approach. Edge computing permits to meet the processing requirements in a scalable, context-aware, and interoperable manner by enabling an efficient data processing distribution between devices, gateways, and cloud. Nevertheless, UAVs should be equipped with high-performance and ultra-low-power neural accelerators to perform learning model evaluation and meet the requirements of the latency-critical application.

This paper presents a framework platform for enabling machine learning on the edge for the fast evaluation of learning models directly on-board on UAVs. The proposed implementation exploits the computational power of Raspberry PI SBC equipped with a neural accelerator. By using a visual sensor, the UAV can promptly react to the changing environment in near-real-time. Two different ML tasks were tested and characterized to evaluate the proposed solution: **(1)** An autonomous landing algorithm based on the evaluation of a DNN; **(2)** An autonomous navigation algorithm based on people detection and tracking.

This paper contains the following contributions:

- The assessment that edge computing is more profitable than cloud computing from an energy point of view by testing it with a real-time system (e.g., drone).
- The implementation and characterization of a processing platform for vision-based inference on the edge. Different HW configurations were tested to assess the best one from an energy processing point of view.
- The implementation and characterization of different learning models to assess the platform functionalities even with the execution of multiple DL models with different complexities in the same run-time.
- The implementation of an autonomous vision-based control mechanism for lightweight UAVs. The prototype developed and used for testing the autonomous navigation functionalities is shown in Fig. 1.

The paper is organized as follows: in Section 2 related works about cloud computing and edge computing on UAVs are discussed. Section 3 presents the video processing algorithm implemented for dataset generation and the pre-processing stage. Section 4 shows the ML algorithms implementation and usage. Finally, Sections 5 and 6 present the platform evaluation and final remarks, respectively.

2. Related works

Advances in UAVs offer unprecedented opportunities to boost a wide array of IoT applications. Nevertheless, UAV platforms still face significant limitations, such as autonomy and weight, which impact their remote sensing capabilities. Therefore, capturing and processing the data required for developing autonomous and robust real-time object detection systems is still challenging.

A UAV must be fully aware of its states to successfully complete the scheduled mission, including location, navigation speed, heading direction, starting point, and target location. Various navigation methods have been proposed, and they can be mainly divided into three categories: inertial navigation [37,38], satellite navigation [39], and vision-based navigation [40,41].

Due to the rapid development of compact, low-power computer vision systems, vision-based navigation proves to be a primary and promising research direction for implementing autonomous navigation. However, UAVs constitute a specific case where the complexity of vision tasks is substantially increased due to the rapid movements that such a platform can experience. These movements are not limited to lateral-forward but also involve movements along the vertical axis, which affect the visioned size (i.e., scale) of the tracked object [41,42].

Recently, researchers have proposed to use ML algorithms to enhance autonomous UAVs reliability and efficiency [43]. In [44] a system for safe landing area detection was developed. It combines classical computer vision and deep learning algorithms to improve system performance. Our work reveals that this combination is a successful approach to improve the detection accuracy and to optimize the execution time as well as the power consumption. In [45], a two-stage training system to spot landing areas is proposed. The first stage consists of a CNN trained with synthetic images of landing areas. The second one employs a transfer learning approach during flight by using weights produced by the previously trained network. In this way, we can obtain a specialized model that can be used in different scenarios. In [46] a weed detection system for precision agriculture is executed autonomously directly on-board an Odroid-U3+ SBC. Results show how the SBC can compute the (x, y) target coordinates, color detection, and weed detection using an object-based image analysis algorithm.

Machine learning methods can be divided into two categories: Cloud and Edge learning approaches.

2.1. Cloud architecture

The Internet of Things (IoT) has incremented the exchange of big data through internet nodes; thus it has initially privileged the use of cloud inference. This method overcomes the computational limitation that characterizes most robotic systems by moving the high-computing demand off-board. In a UAV scenario, this paradigm collects the data while flying, sends them remotely to a server to compute the inference, and waits for the feedback result. A cloud-based approach permits to use deep ML models. In [47], authors have developed a cloud-based object detection system for UAVs using a Region-based Convolutional Neural Network (R-CNN). As well-known, CNNs are characterized by an enormous number of operations (especially with the 2D convolution layers and with fully connected layers) to compute a prediction giving an input image. Thus, a cloud inference approach moves high computing inference to servers with unconstrained resources. This system was tested and characterized by comparing the execution speed and the prediction accuracy with the state of the art YOLO and SSD object detectors using aerial images. This test revealed that the proposed R-CNN presents the best accuracy (83.9 mAP) but the lowest speed (3.48 FPS). Furthermore, an additional comparison was conducted by considering the execution time of fast-YOLO on a local laptop and the R-CNN on a remote server as a simulated cloud. In the first case, YOLO takes about 7 s to carry out one application cycle, while R-CNN takes only 1,29 s by also considering the transmission latency, which is one-third of the whole time. Another cloud-based real-time object tracking by using UAVs called "Dronetrack" was developed in [48]. This system exchanges data between the drone, the server, and the user on the ground. In this way, it is possible to send the video acquired by the drone and use it to perform object tracking off-board. This system was tested in three different scenarios: (i) a walking person in a football pitch using a WiFi router operating with 3G/4G, (ii) a walking person in a city district with an optic fiber connection, and (iii) a moving vehicle in a city district with an optic fiber connection as well. Fast connections (e.g., optic fiber) can lead to better performance, thus reaching a real-time execution. However, the optic fiber coverage is still limited, and might not be available in every type of environment and situation. As confirmed by the state of the art, data transmission can be predominant and introduces a consistent latency that limits this approach to only some specific applications. The arrival of 5G can enhance the cloud architecture capability by improving speed and bandwidth. However, if the considered application needs the drone to interact actively and continuously with the surrounding environment, the cloud approach continues to have serious drawbacks.

2.2. Edge architecture

Cloud systems have revealed their powerful aspects because of their almost unlimited resources. However, it is not always possible to have a wireless and robust connection. Moreover, the introduced transmission latency might be consistent, thus limiting UAVs operations only in specific applications. Autonomous UAVs based on edge computing architecture avoid data transmission, and they can be scaled in every application achieving better execution performance. Furthermore, edge computing is already widespread in fixed robotic systems (e.g., surveillance and monitoring) to slim down the amount of data that needs to be transmitted. For instance, a fixed robotics system for pest detection was developed in [49,50]. This research has implemented a ML-based system for pest detection in apple orchards using Raspberry Pi3 equipped with Pi camera and Intel Movidius NCS. Here, the edge inference approach was used to process the data through a VGG16 CNN [51] for insect classification(i.e., the effective presence of dangerous insects). In this way, it is possible to decrease the amount of data to be transmitted to the remote server by limiting it to a simple notification of a few bytes.

The proposed hardware can be moved to mobile robotics to enable autonomous navigation. A perception, guidance, and navigation system for autonomous drone racing by using DL techniques was developed by [52], which navigates racing drones through gates. Even if the most common computer vision techniques have been revealed inefficient, it is still possible to estimate the gate center quickly and accurately by using CNNs. This system uses a NVIDIA Jetson TX2 – one of the most powerful embedded AI computing devices to perform inference near sensors – as a companion board to carry out all vision processing. The presented system was compared with other two state of the art networks (VGG-16 based SSD [53] and AlexNet based SSD [54,55]). The proposed approach resulted the fastest (28,95 FPS) but the worst in terms of detection accuracy (85,2% detection rate). The NVIDIA Jetson TX2 was also used for embedded real-time object detection in a UAV warning system [56]. It uses the state of the art Yolo V2 [57] network, which enables dangerous products recognition directly on-board. Specifically, the Jetson TX2 is responsible for the execution of the object detection and positioning algorithm. At the same time, the decision support engine back-end is responsible for the warning alarm in the case of people getting too close to dangerous areas. In this case, the system performance was evaluated in terms of image input resolution and detection frame rate by comparing Yolo V2 [57], and Tiny Yolo [58]. This research confirmed that using low-resolution input images and simple network architectures (e.g., Tiny Yolo) can lead to faster execution.

Embedded GPUs and SBCs can usually host models in the order of GBs, but MCUs have a few hundred KBs available. This limit sets the capability of a platform of running complex ML algorithms [59]. Edge processing needs model compression techniques to evaluate DNNs in edge platforms, using lightweight models optimized for real-time inference. The challenge consists of highly compressing the models without losing accuracy. If properly addressed, we can move the ML task into mobile devices, or IoT systems, while keeping the original QoS [60]. The most used compression techniques are quantization and pruning; the first quantizes weights from floating-point to integer to reduce the number of bits and, consequently, the model memory footprint. The second one cuts off weights close to 0 and their relative connections to reduce the model complexity and memory requirements. After this operation, the model usually needs to be retrained to fine-tune it with the missed branches. This approach leads to an optimized model that can reduce the QoS by losing prediction accuracy. However, surveys have shown that even if the model is highly compressed, the loss in accuracy can be negligible (i.e., in the order of 0.001%) [61–65].

2.3. Platform and neural network selection

Many researchers are looking to find the best trade-offs for executing AI tasks in resource-constrained environments, such as UAVs or MCUs. Both computing platforms and learning models have been investigated to find the best implementation. Edge platforms can mainly be divided into three big families of devices [66,67]: SBCs, Edge GPUs and FPGAs. SBCs such as Raspberry, Odroid, and BeagleBone are more flexible than FPGA-based platforms, and they can enable edge computing by processing raw sensor data with a small energy overhead. Alongside the study of general-purpose SBC-based solutions, researchers have proposed to exploit FPGAs [68,69]. These platforms can lead to better energy consumption and inference acceleration performance. However, their usage is strictly tailored for a specific application, thus not extensible for related applications with different requirements [70]. The last edge platform family is embedded GPUs, such as NVIDIA Jetson Nano and Google Coral. They are widely used in edge computing and can lead to faster neural inference, but at the cost of higher power consumption (in the order of 10 W). In our use case, embedded GPUs are oversized for the tasks, while FPGAs are too application-specific. For these reasons, we focused on the SBC family. Odroid and BeagleBone feature high execution performance but are still too energy-demanding.

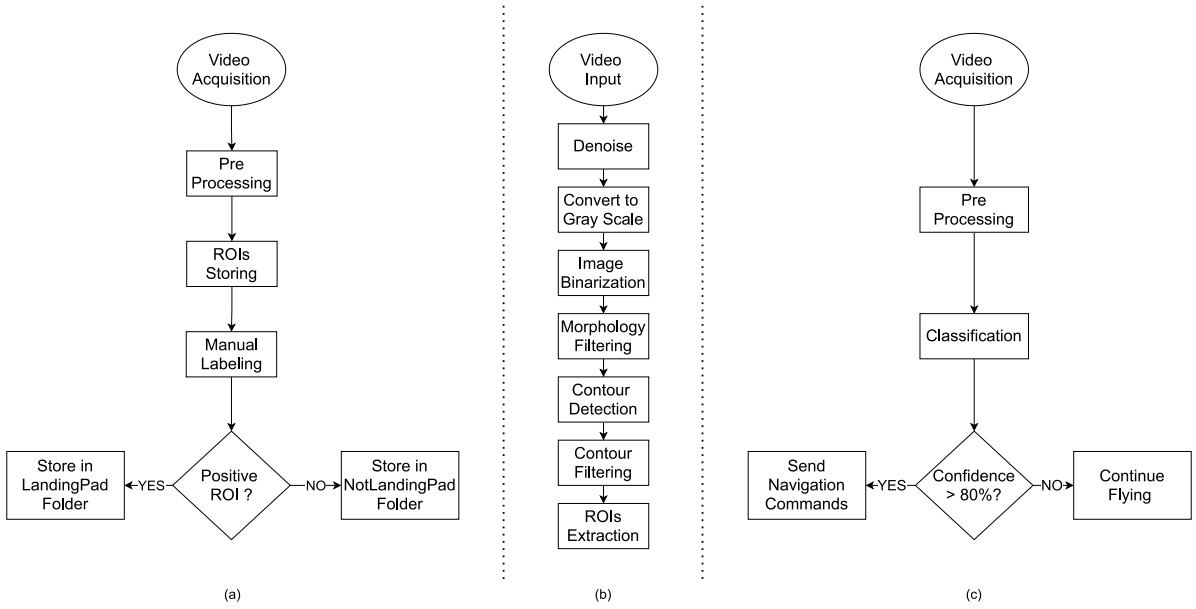


Fig. 2. Machine learning task workflow. (a) Dataset generation flowchart. (b) Pre-processing algorithm flowchart. (c) Landing pad detection algorithm flowchart. Flowchart (b) present the pre-processing block of flow chart (a) and (c).

Moreover, they are not supported by a large community like Raspberry Pi, making them less appealing for prototyping. In addition, the camera used in our case study is not supported by Odroid and BeagleBone. Raspberry, instead, is a flexible platform supporting a wide range of peripherals and learning models and fits other application requirements such as connectivity, supported modules, execution performance, and power consumption [71].

Concerning learning models, in the state of the art, we can already find some notable implementations [72]. DroNet [73], for example, is an efficient CNN detector for real-time UAV applications. It uses network optimization and weights pruning to optimize the memory footprint of a modified Tiny-Yolo network. The improvement consists of using a single class classifier and changing the number of filters, layers, input size, convolutions, and pooling layers. With the proposed modifications, the new architecture reduces the feature map by 2. System performance was assessed with a vehicle detection application exploiting an Odroid-XU4 SBC. The platform achieved a frame rate execution around 8–10 FPS with an accuracy of 95%. Moreover, the same DL model was evaluated on Raspberry Pi3, which reveals worse performance due to the less capable CPU. As shown, DroNet is suitable for drones' real-time applications. However, it is based on a cloud architecture: the drone streams a video to a base station that performs the processing and sends back the navigation commands. Another interesting implementation, namely EdgeNet, is presented in [74]. It efficiently performs object detection with an edge detector approach [75]. The proposed network is composed of three main steps; the first consists of a lightweight CNN for object detection. It follows the selection of image regions that contain the detected objects to reduce the image size and processing complexity. Finally, it employs an optical flow-based tracker to reduce the processing demand without using the CNN-based object detector. EdgeNet can process high-resolution images, such as aerial images, with a low-power profile and a fast processing execution, making it suitable for edge applications. Even though, there are notable ML models in the state of the art, we have decided to stick with two older implementations, namely LeNet and MobileNetV2. These models are already well investigated and implemented online, speeding up the development phase. In this way, we focused more on evaluating the edge approach with respect to the cloud approach, also showing the lower energy figure by running the ML tasks directly on compact UAVs.

3. Video processing

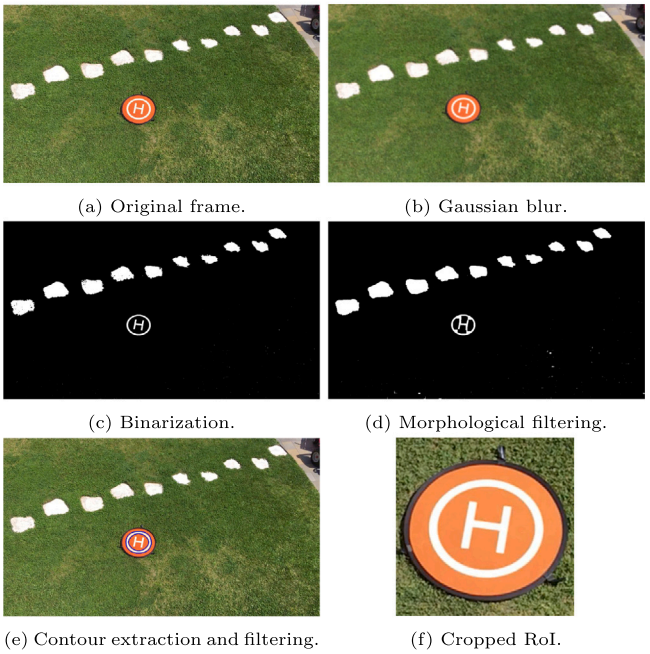
Video pre-processing is a fundamental step in vision-based AI applications to make correct predictions and train a CNN. A basic video processing algorithm highlights relevant features in a frame (e.g., contours, edges, contrast) and reduces the noise introduced by the image sensor. This section introduces the video processing algorithm developed to create a consistent dataset for the CNN training semi-automatically. The same algorithm is used as a pre-processing step in the final application workflow. For this purpose, many videos of different landing pads from different environmental situations were acquired to compose various datasets (e.g., landing pad in a garden, on the street, in a car parking).

3.1. Pre-processing implementation

Pre-Processing mainly consists of noise reduction, features highlighting, and cropping of the region of interest (RoI), which may contain a landing pad. It is implemented using *Python* program language and the *OpenCV* library.

Fig. 2b presents the algorithm workflow, encompassing the following steps:

- **Video Acquisition**: Video loading either as a direct stream from the RPi camera or as video file.
- **Denoising**: Gaussian blurring filter applied to remove high-frequency noise. A kernel mask of shape 9×9 is used.
- **Gray scaling**: Conversion from RGB to gray-scale.
- **Segmentation**: Image binarization through an adaptive Gaussian threshold.
- **Morphology Filtering**: Application of morphological filters with a structuring element of size 3×3 and unitary element values. Two iterations of opening and four iterations of dilation are applied to highlight each blob of interest.
- **Contour detection**: Finds each contour revealed from each processed frame.
- **Contour filtering**: Filtering out small contours (filter by area) and contours that do not present any hierarchy level (filter by hierarchy).
- **RoI cropping**: Finally, each region of interest around each filtered contour is extracted.



(a) Original frame. (b) Gaussian blur. (c) Binarization. (d) Morphological filtering. (e) Contour extraction and filtering. (f) Cropped RoI.

Fig. 3. Example of video processing algorithm workflow.

(a) Hand crafted. (b) Blue landing pad. (c) Red landing pad. (d) CiterX (synthetic). (e) Green square (synthetic). (f) Red landing pad (synthetic). (g) Storm drain. (h) Tile. (i) Bike.

Fig. 4. Example of dataset images.

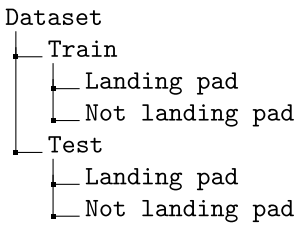


Fig. 5. Dataset directory tree.

After the pre-processing phase, each cropped RoI can be used both for creating the databases needed for the training phase, or as the input for the neural network evaluation. The step-by-step result of the video processing algorithm is presented in Fig. 3.

4. Landing pad detection

The landing pad detection problem can be expressed in terms of symbol/character recognition and image classification. For those reasons, a modified LeNet-5 [76,77] CNN architecture, presented in Fig. 7, was used. Initially developed for character recognition, it can be extended to classification problems, especially for this application that detects symbols on a landing area. It presents a straightforward architecture, which makes the right fit feasible for embedded applications.

This NN takes gray-scale images of size 64 × 64 as input which is processed through two 2D convolution layers with a *ReLU* activation function, which exhibits a better classification accuracy than the original activation function *tanh*. Furthermore, the second convolutional block does not use all the features extracted by the average pooling layer, permitting to learn different patterns and enhance the classification accuracy. This customization makes the network less computational demanding and suitable for embedded platforms. Finally, the last layer is a *Softmax* classifier which provides the confidence probability among two classes (binary classification): *Landing pad* and *Not landing pad*. Along with LeNet, MobileNetV2 [78,79] network, was trained over the same dataset to compare its performance. Network architecture is presented in Fig. 6. MobileNetV2 presents a more deep and complex architecture than LeNet, but, due to the inverted residual block, it drastically reduces by a factor of 13 the number of parameters compared to LeNet. LeNet is characterized by a fully connected layer to flatten the output which considerably increases the network number of parameters. On the other hand, MobileNetV2 architecture introduces two types of inverted residual blocks to lower the number of connections:

- The first is with stride 1 to collect most of the learnable features.
- The second is with stride 2 for downsizing to reduce the network complexity.

MobileNetV2 architecture is composed of three layers for all types of blocks: the first is a 1 × 1 convolution layer with ReLU6 activation function. The second one is a depth-wise convolution that, despite common convolutions, reduces the computational complexity of this operation. Finally, the third layer is another 1 × 1 convolution without any non-linearity. The idea is that the bottleneck blocks encode the model intermediate inputs and outputs, while the internal layer collects the model ability to represent lower-level features to higher-level concepts such as image classes. This makes the network highly efficient for developing mobile applications permitting a fast training stage and good accuracy.

4.1. Dataset creation and organization

The dataset construction and organization are important steps for a well NN training. It is preferable to have a consistent dataset with various images to obtain an accurate model that can generalize the prediction better. To this end, the dataset was constructed by using the algorithm explained in Section 3.1 which processes multiple video streams of different kinds of landing pads. In this way, it is possible to generate a dataset with thousands of images in a semi-automatic way by following the workflow in Fig. 2a. However, not all obtained RoIs present a landing pad, and sometimes the algorithm shows RoIs with

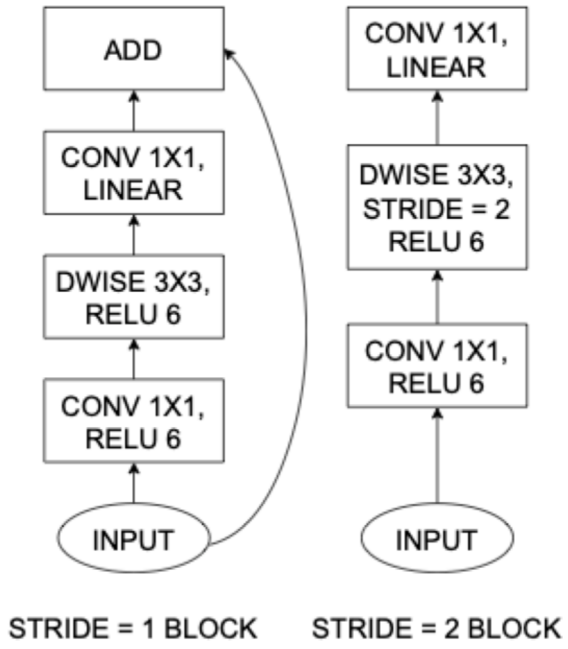


Fig. 6. MobileNetV2 architecture.

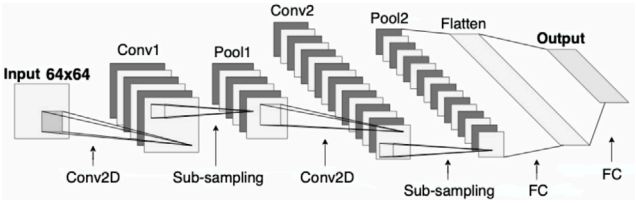


Fig. 7. LeNet-5 architecture.

Table 1
Landing pad detection algorithm training parameters.

	Network architecture	
	LeNet-5	MobileNet V2
Epochs	20	20
Batch size	32	16
Initial learning rate	10^{-3}	10^{-3}
Input image	Single channel 64 × 64	Single channel 100 × 100
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Binary cross-entropy
Source framework	Tensorflow	Tensorflow
Training accuracy	0.998	0.999
Validation accuracy	0.992	0.999

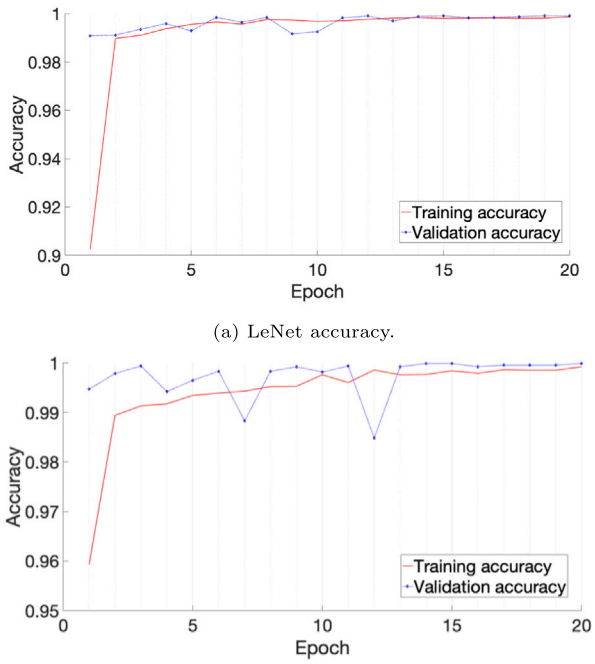
miscellaneous objects such as storm drains, tile, and plants. Considering that the pre-processing algorithm may fail in this way, we chose to use this kind of object to construct the *Not landing pad* class.

Furthermore, synthetic images – generated following [80] – representing landing pads with different shapes and symbols were added to the dataset to obtain a more heterogeneous model. The dataset was organized in folders as Fig. 5 to facilitate the automatic label extraction from each image. The training dataset consists of 13 576 images for the *Landing pad* class and 12 807 images for the *Not landing pad* class.

The obtained dataset is composed of various types of landing pads and of different miscellaneous objects, as shown in Fig. 4.

4.2. CNN training

The final application requires that the trained CNN could be deployed on a micro-computer to perform edge computing while still



(a) LeNet accuracy.

(b) MobileNetV2 accuracy.

Fig. 8. Accuracy comparison.

Table 2
LeNet and MobileNetV2 prediction figures.

	Accuracy	Recall	Precision	F-score
LeNet-5	99.4	99.5	99.9	99.7
MobileNetV2	99.4	99.6	100	99.8

presenting good performances. Thus, it is necessary to focus on the neural network's memory footprint and model complexity. For this purpose, many training sessions with different parameters (i.e., number of epochs, input image size and depth, and batch size) were executed. The best trade-off was found by using parameters shown in Table 1 and by performing data augmentation to enhance the dataset and the network accuracy. Plots in Fig. 8 show the training behavior of both networks over the epochs. Notice that MobileNetV2 presents a pair of abrupt drops in validation accuracy due to fewer trainable parameters. However, despite these drops, its validation accuracy presents a linear increasing profile.

4.3. CNN test

Both networks were tested with about 3000 images that were retrieved from the dataset before the training session. Moreover, few images of even different landing pads found on the web (about 30) were added to the test dataset to assess the network generalization capability. The results are summarized in Table 2. Both networks reach satisfying results in accuracy and precision, making these neural models suitable for this application.

An example of the result of the landing pad detection from an aerial video collected during an on-field test is shown in Fig. 9(a).

4.4. Embedded implementation

The overall algorithm has to run on the Raspberry Pi microcomputer equipped with a Pi Camera and the Movidius NCS, which accelerates and optimizes the neural inference. Furthermore, the *OpenVINO* toolkit was used to optimize the landing pad detection model to

Table 3
People detection model specifications.

Parameter	Value
Average Precision (AP)	88.62%
Pose coverage	Standing upright,parallel to image plane
Support of occluded pedestrians	YES
Occlusion coverage	<50%
Min pedestrian height	100 pixels (on 1080p)
GFlops	2.300
MParams	0.723
Source framework	Caffe
Input shape	[1 × 3 × 320 × 544]
Color order	BGR

an Intermediate Representation (IR) and to perform video processing and neural inference on the Movidius NCS.

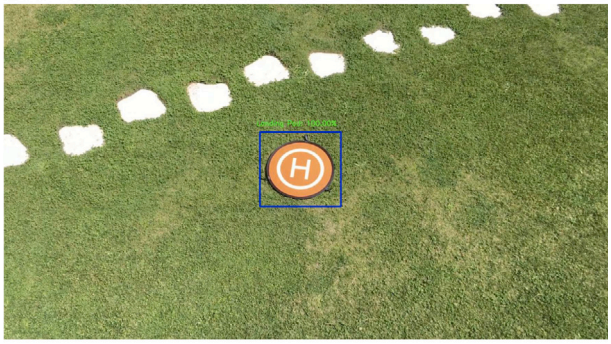
The workflow of the landing pad detection algorithm is shown in Fig. 2c, and it consists of the following steps. The Pi Camera enables a continuous streaming mode at 30 fps, and each frame is collected. Then, the frame is processed with the pre-processing algorithm shown in Section 3.1 which outputs the RoIs that may contain a landing pad. Because the pre-processing algorithm is not reliable as a landing pad detector, each RoI is classified with the CNN shown in Section 4.2 which gives an accurate result by identifying the effective presence of a landing pad. The classification step produces a confidence level that reveals how the CNN is sure that the input RoI is a landing pad. For this purpose, a threshold value of 0.8 is set to have the best trade-off between precision and recall. It is worth to notice that the pre-processing step involves the usage of Raspberry together with Movidius NCS. In contrast, the classification step involves only the Movidius NCS, which boosts the neural inference execution time.

4.5. Execution of multiple DL models

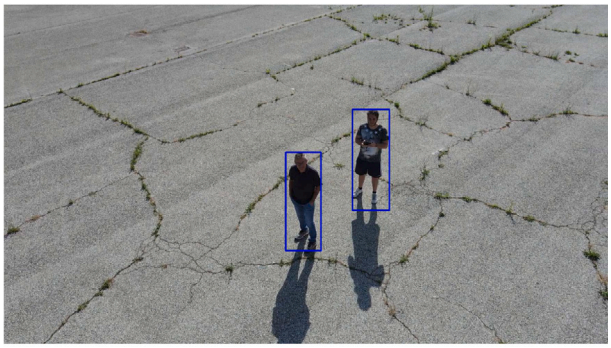
This paper addresses the problem of the fast deployment of DL models in resource-constrained platforms such as UAVs. The developed framework platform capability was assessed by running multiple DL models in the same runtime. For this purpose, a pre-trained model for people detection[1] (specifications in Table 3) based on MobileNetV2-like backbone [78,79] was used and tested.

This model was executed in the target hardware together with the landing pad detection model. As shown in Fig. 10, they work as two subordinate systems. When the drone is flying, it first checks the presence of people and, if recognized, it plans a path to avoid collisions with them. If any person is detected, it looks for a safe landing area (i.e., execution of the landing pad detection algorithm) and, if the confidence is higher than 80%, it autonomously lands on the detected landing pad. The pre-trained model used for people detection is well suited to recognize pedestrians from an aerial survey, as shown in Fig. 9(b). However, due to the dataset used for the training session, the model needs a minimum pixel-wise pedestrian height of about 100 pixels on a 1080p image. This makes high altitude people detection difficult. It is thus clear that, if the drone flies too high, it cannot achieve the desired detection accuracy. To enhance the people detection algorithm capability, it is possible to further train the model with a sparser dataset with aerial images representing people of different sizes and positions (e.g., people close or very remote to the camera system). Moreover, a transfer learning approach [81–83] can be considered to improve the overall accuracy. It starts from an already-trained network for related tasks and fine-tunes it to obtain one with the required detection accuracy. For instance, it is possible to use a base network for people detection and extend its capability to detect tiny tiled people (i.e., a dozen pixels). By doing so, the model will not be limited by the people's height, and it will also be suitable for aerial applications.

[1] Source: https://docs.openvinotoolkit.org/latest/omz_models_model_person_detection_retail_0013.html.



(a) Landing pad detection example.



(b) People detection example.

Fig. 9. Demo of the proposed algorithms for people and landing pad detection.

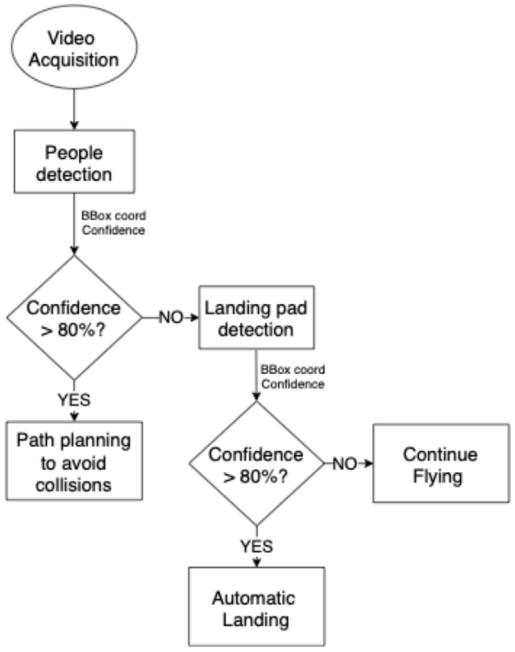


Fig. 10. Execution of multiple DL models flowchart.

5. Results

The presented framework platform was characterized by measuring the execution time and power consumption of the landing pad detection algorithm along with the people detection algorithm. In this way, it is possible to find out the system bottlenecks and improve the platform. Furthermore, two different hardware configurations were tested to compare their performance: Raspberry Pi3 with Movidius NCS v2 and

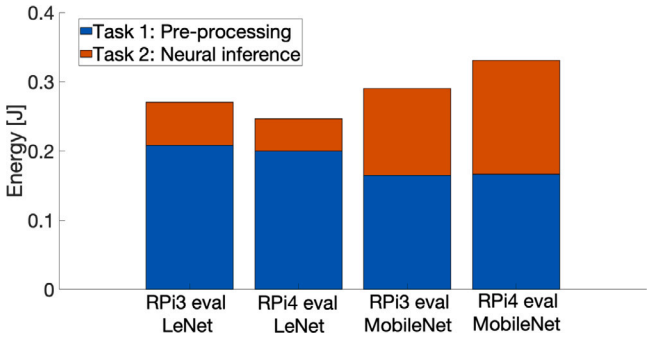


Fig. 11. Landing pad detection energy comparison.

Table 4
Tasks energy breakdown and execution time of the tested platforms by executing only the landing pad detection algorithm. The best trade-off is provided by Raspberry Pi4 evaluating LeNet boosted with NCS.

Configuration	Task 1 [J]	Task 2 [J]	Total [J]	Elapsed time [ms]	FPS
RPi3 eval LeNet	0.208	0.062	0.27	74.69	13.37
RPi4 eval LeNet	**0.2**	**0.046**	**0.246**	**44.90**	**22.26**
RPi3 eval MobileNetV2	0.165	0.125	0.29	76.73	13.04
RPi4 eval MobileNetV2	0.166	0.165	0.331	59.59	16.78

Raspberry Pi4 with Movidius NCS v2. For this purpose, the application workflow, referred to Fig. 2c, was split into two general tasks:

- **Task 1**: Video pre-processing.
- **Task 2**: Neural inference.

Video streaming and post-processing tasks (i.e., sending the land command) were neglected because they require two orders of magnitude less time than the other two tasks, thus not affecting the measurement results. The measurement campaign was conducted by processing 50 frames acquired from the camera installed in a static scene with a landing pad and a few people. Then, the measurements were averaged, and each task's energy consumption and execution time were evaluated. Different configurations were considered to get a complete comparison for the two target hardware:

- Landing pad detection with LeNet.
- Landing pad detection with MobileNetV2.
- People detection and landing pad detection with LeNet.
- People detection and landing pad detection with MobileNetV2.

5.1. Landing pad detection on Raspberry with NCS

The execution of a single DL model was initially considered. Fig. 11 shows the energy consumption among the different hardware and neural network configurations for one application cycle (i.e., processing one frame). The required energy for each task and the relative execution performance are summarized by Table 4.

It is possible to notice that the most power-hungry task is video pre-processing. It uses an ad-hoc video processing routine that increases the computational requirement. Furthermore, MobileNetV2 architecture increases the power consumption in the second task because of its complex architecture. By considering the energy consumption and the execution performance, the best configuration is the Raspberry Pi4 evaluating LeNet architecture for landing pad detection. It consumes 0.246 J for processing one frame, and it achieves real-time performance processing 22.26 frames per second.

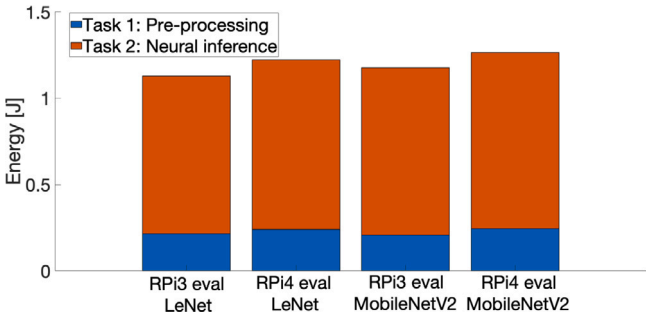


Fig. 12. People and landing pad detection energy comparison.

Table 5
Tasks energy breakdown and execution time of the tested platforms by executing the landing pad detector along with the people detection algorithm. The best trade-off, considering the energy consumption, is provided by Raspberry Pi3 evaluating LeNet boosted with NCS. If the execution time is the key metric, the best trade-off is provided by Raspberry Pi4 evaluating LeNet boosted with NCS.

Configuration	Task 1 [J]	Task 2 [J]	Total [J]	Elapsed time [ms]	FPS
RPi3 eval LeNet	0.216	0.912	**1.128**	309.8	3.23
RPi4 eval LeNet	0.240	0.983	1.223	213.88	**4.68**
RPi3 eval MobileNetV2	0.207	0.969	1.176	322.86	3.1
RPi4 eval MobileNetV2	0.244	1.02	1.264	226.33	4.42

5.2. People detection and landing pad detection on Raspberry and NCS

The same comparison of Section 5.1 was conducted by running multiple DL models in the same run-time. In this case, the landing pad detector is running together with a people detection model to assess the platform limits. Even though both models share the same sensed data and the same NCS where the neural inference is performed, they are executed in a subordinate way and work as two independent systems. Fig. 12 presents the energy consumption among the different configurations. Table 5 further clarify the required energy and execution performance.

Here, the most power-hungry task turned out to be the second because of the people detection algorithm. It requires an extremely high computational power due to its large image input size and the complex architecture that characterizes this DL model. Moreover, the first task energy consumption is slightly increased compared to the previous case because the people detector requires an additional pre-processing phase. If we consider the energy consumption, the best configuration is the Raspberry Pi3 evaluating LeNet, which requires about 100 mJ less than the other configurations. However, its execution performance is limited, achieving a maximum processing speed of 3.23 FPS. This is due to the less performing Raspberry Pi3 SOC. On the other hand, when the execution time matters most, the best configuration turns out to be the Raspberry Pi4 evaluating LeNet. It needs 1.223 J to process one frame, but the performance is increased to 4.68 FPS.

5.3. Drone flight time reduction

The presented platform aims to make UAVs autonomous; thus, the combination of Raspberry, Movidius NCS, and Camera has to be mounted on-board on drones and plugged into their energy source (i.e., battery). However, the available energy is limited. A precious resource to judiciously use to ensure a long enough flight time. The drone's battery has to power all the avionics for flight control and, in addition, the RPi SBC running the autonomous navigation algorithms. For this reason, we evaluated the degradation of the flight time caused by the energy overhead associated with the AI platform.

A 250 mm class drone (i.e., small size) equipped with a 1500 mAh battery operating at 14.8 V was taken into account, achieving 15 min

Table 6
Drone flight time reduction overview. Columns show the flight time and the corresponding energy demand. Rows consider three different implementations: without the autonomous navigation system (**normal**), with the landing pad detector (**single model**) and with people and landing pad detector (**multiple models**).

	Flight time [min]	Available energy [kJ]
Normal	15	80
Single model	14.1	76
Multiple models	14	75.6

of flight time in normal conditions. The best performing – or the most energy efficient – configuration was considered for estimating the drone battery lifetime reduction. Table 6 compares the flight time and the corresponding energy for three configurations: (i) normal mode, (ii) execution of a single model, and (iii) execution of multiple models. Results show that the proposed platform reduces the drone flight time by 6% on average, not consistently affecting the drone's flight time.

5.4. Edge computing energy sustainability

Recent research has highlighted that the edge computing approach can outperform the cloud by avoiding data transmission and speeding up the application execution. To show how the energy consumption is distributed among the two different approaches, we compared the best configuration investigated above for landing pad detection (i.e., RPi4 evaluating LeNet) with the same application running on a Parrot Mambo mini-drone in cloud mode. The detection accuracy does not change because the landing pad detection model and the test images are the same; nevertheless, the application's platforms are different, thus the performance in terms of energy consumption and execution time is different. The mini-drone utilizes a laptop as a ground station which features an Intel core i5-3360M CPU@2.80 GHz × 4. It is connected to the mini-drone with a private wi-fi network which enables data exchange among mini-drone and ground station. The mini-drone acquires an image from its ground camera and sends it to the station; then, it performs image pre-processing (Task 1) and neural inference (Task 2), and sends back the landing command if a safe landing area is detected. Considering cloud computing, it is worthwhile to analyze the energy consumed from the drone side for the networking overhead without considering the processing on the ground station because it is plugged into an unlimited energy source. In this way, it is possible to find out the most energy-effective approach and assess the platform energy sustainability. The comparison is presented in Fig. 13. It shows that the networking overhead for data exchange in cloud mode is much more expensive (i.e., 2.16 J) than the overall processing with edge computing (i.e., 0.246 J). This confirms the energy sustainability of the edge computing approach compared to the cloud one. Moreover, edge computing outperforms cloud computing, considering the execution time. The first one takes 45 ms to process one frame, while the second needs 1.4 s. This comparison clearly shows that edge computing satisfies real-time requirements in a resource-constrained environment at low energy consumption. Cloud computing degrades its performance due to the transmission latency (i.e., 1.34 s) and the required energy for data exchange.

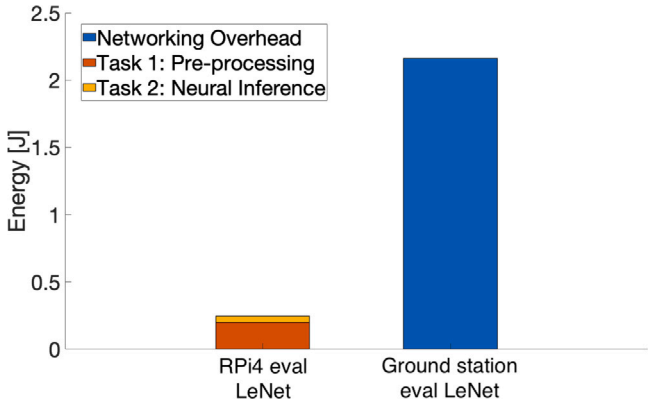


Fig. 13. Energy consumption comparison of edge computing and cloud computing for landing pad detection.

6. Conclusion

Together with computer vision techniques, deep learning algorithms are widely spread in autonomous UAV vision systems. However, some autonomous real-time applications are still challenging. This paper addresses the problem by implementing a platform for the fast-deployment of DL algorithms directly onboard UAVs, with an edge computing approach. In this way, all image processing and neural inferences are computed near the image sensor, avoiding big data management that characterizes cloud architectures. The platform performance was assessed with an autonomous landing algorithm that exploits ML functionalities. It was trained and compared among two different CNN architectures: LeNet and MobileNetV2. An additional pre-trained model for people detection was deployed on the proposed platform to characterize and compare the performance by running multiple models in the same run-time. The best hardware and NN architecture configuration were evaluated by considering the fastest implementation and less energy demanding. Finally, the energy impact of the platform on the drone flight time was taken into account, showing a marginal flight time reduction. Future work will encompass the study of recent more efficient DL networks such as DroNet [73] and EdgeNet [74]. Once trained the two models with the same dataset, we will compare the different DL networks to find the best implementation for enabling autonomous navigation and environmental awareness.

CRediT authorship contribution statement

Andrea Albanese: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Matteo Nardello:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Davide Brunelli:** Conception and design of study, Analysis and/or interpretation of data, Writing – review & editing.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.suscom.2022.100725.

Acknowledgment

All authors approved the version of the manuscript to be published. This work was supported by the Italian Ministry for University and Research (MUR) under the program "Dipartimenti di Eccellenza (2018-2022)".

References

[1] M.B. Bejiga, A. Zeggada, A. Nouffidj, F. Melgani, A convolutional neural network approach for assisting avalanche search and rescue operations with UAV imagery, Remote Sens. 9 (2) (2017) http://dx.doi.org/10.3390/rs9020100, URL https://www.mdpi.com/2072-4292/9/2/100.

[2] E. Lygouras, N. Santavas, A. Taitzoglou, K. Tarchanidis, A. Mitropoulos, A. Gasteratos, Unsupervised human detection with an embedded vision system on a fully autonomous UAV for search and rescue operations, Sensors 19 (16) (2019) http://dx.doi.org/10.3390/s19163542, URL https://www.mdpi.com/1424-8220/19/16/3542.

[3] M.-B. Leduc, A.J. Knudby, Mapping wild leek through the forest canopy using a UAV, Remote Sens. 10 (1) (2018) http://dx.doi.org/10.3390/rs10010070, URL https://www.mdpi.com/2072-4292/10/1/70.

[4] Y. Zhang, Y. Zhang, Z. Yu, A solution for searching and monitoring forest fires based on multiple UAVs, in: 2019 International Conference on Unmanned Aircraft Systems (ICUAS), 2019, pp. 661–666, http://dx.doi.org/10.1109/ICUAS.2019.8797786.

[5] Y. Angel, M. Morton, Y. Malbeteau, S.S.C. Negrão, G.M. Fiene, M.A.A. Mousa, M. Tester, M. McCabe, Multitemporal monitoring of phenotypic traits in wild tomato species (s. pimpinellifolium) using UAV-based hyperspectral imagery, in: AGU Fall Meeting Abstracts, Vol. 2019, 2019, pp. B31K–2415.

[6] F. Vanegas, D. Bratanov, K. Powell, J. Weiss, F. Gonzalez, A novel methodology for improving plant pest surveillance in vineyards and crops using UAV-based hyperspectral and spatial data, Sensors 18 (1) (2018) http://dx.doi.org/10.3390/s18010260, URL https://www.mdpi.com/1424-8220/18/1/260.

[7] J. Kaivosoja, J. Hautsalo, J. Heikkinen, L. Hiltunen, P. Ruuttunen, R. Näsi, O. Niemeläinen, M. Lemsalu, E. Honkavaara, J. Salonen, Reference measurements in developing UAV systems for detecting pests, weeds, and diseases, Remote Sens. 13 (7) (2021) http://dx.doi.org/10.3390/rs13071238, URL https://www.mdpi.com/2072-4292/13/7/1238.

[8] E.C. Tetila, B.B. Machado, G. Astolfi, N.A. de Souza Belete, W.P. Amorim, A.R. Roel, H. Pistori, Detection and classification of soybean pests using deep learning with UAV images, Comput. Electron. Agric. 179 (2020) 105836, http://dx.doi.org/10.1016/j.compag.2020.105836, URL https://www.sciencedirect.com/science/article/pii/S016816991831055X.

[9] F. Qi, X. Zhu, G. Mang, M. Kadoch, W. Li, UAV network and IoT in the sky for future smart cities, IEEE Netw. 33 (2) (2019) 96–101, http://dx.doi.org/10.1109/MNET.2019.1800250.

[10] A. Tiurlikova, N. Stepanov, K. Mikhaylov, Improving the energy efficiency of a LoRaWAN by a UAV-based gateway, in: 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2019, pp. 1–6, http://dx.doi.org/10.1109/ICUMT48472.2019.8970857.

[11] R.A. Nazib, S. Moh, Energy-efficient and fast data collection in UAV-aided wireless sensor networks for hilly terrains, IEEE Access 9 (2021) 23168–23190, http://dx.doi.org/10.1109/ACCESS.2021.3056701.

[12] R. Vijayanandh, J. Darshan Kumar, M. Senthil Kumar, L. Ahilla Bharathy, G. Raj Kumar, Design and fabrication of solar powered unmanned aerial vehicle for border surveillance, in: P.J. Rao, K.N. Rao, S. Kubo (Eds.), Proceedings of International Conference on Remote Sensing for Disaster Management, Springer International Publishing, Cham, 2019, pp. 61–71.

[13] M.L. Laouira, A. Abdelli, J.B. Othman, H. Kim, An efficient WSN based solution for border surveillance, IEEE Trans. Sustain. Comput. 6 (1) (2021) 54–65, http://dx.doi.org/10.1109/TSUSC.2019.2904855.

[14] F. Granelli, C. Sacchi, R. Bassoli, R. Cohen, I. Ashkenazi, A dynamic and flexible architecture based on UAVS for border security and safety, in: C. Palestini (Ed.), Advanced Technologies for Security Applications, Springer Netherlands, Dordrecht, 2020, pp. 295–306.

[15] N. Kalatzis, M. Avgeris, D. Dechouniotis, K. Papadakis-Vlachopapadopoulos, I. Roussaki, S. Papavassiliou, Edge computing in IoT ecosystems for UAV-enabled early fire detection, in: 2018 IEEE International Conference on Smart Computing (SMARTCOMP), 2018, pp. 106–114, http://dx.doi.org/10.1109/SMARTCOMP.2018.00080.

[16] A.D. Boursianis, M.S. Papadopoulou, P. Diamantoulakis, A. Liopa-Tsakalidi, P. Barouchas, G. Salahas, G. Karagiannidis, S. Wan, S.K. Goudos, Internet of things (IoT) and agricultural unmanned aerial vehicles (UAVs) in smart farming: A comprehensive review, Internet Things (2020) 100187, http://dx.doi.org/10.1016/j.iot.2020.100187, URL https://www.sciencedirect.com/science/article/pii/S2542660520300238.

[17] N. Zhao, W. Lu, M. Sheng, Y. Chen, J. Tang, F.R. Yu, K.-K. Wong, UAV-assisted emergency networks in disasters, IEEE Wirel. Commun. 26 (1) (2019) 45–51, http://dx.doi.org/10.1109/MWC.2018.1800160.

[18] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, I. Moscholios, A compilation of UAV applications for precision agriculture, Comput. Netw. 172 (2020) 107148, http://dx.doi.org/10.1016/j.comnet.2020.107148, URL https://www.sciencedirect.com/science/article/pii/S138912862030116X.

[19] N. Ayyappaa, A.Y. Raj, A. Adithya, R. Murali, A. Vinodh, Autonomous drone for efficacious blood conveyance, in: 2019 4th International Conference on Robotics and Automation Engineering (ICRAE), 2019, pp. 99–103, http://dx.doi.org/10.1109/ICRAE48301.2019.9043820.

[20] B.D. Song, K. Park, J. Kim, Persistent UAV delivery logistics: MILP formulation and efficient heuristic, Comput. Ind. Eng. 120 (2018) 418–428, http://dx.doi.org/10.1016/j.cie.2018.05.013, URL https://www.sciencedirect.com/science/article/pii/S0360835218302146.

[21] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, D. Scaramuzza, Vision-based autonomous quadrotor landing on a moving platform, in: 2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), 2017, pp. 200–207, http://dx.doi.org/10.1109/SSRR.2017.8088164.

[22] F. Sadeghi, S. Levine, CAD2Rl: Real single-image flight without a single real image, 2017, arXiv:1611.04201.

[23] D.K. Kim, T. Chen, Deep neural network for real-time autonomous indoor navigation, 2015, arXiv:1511.04668.

[24] P. Santana, L. Correia, R. Mendonça, N. Alves, J. Barata, Tracking natural trails with swarm-based visual saliency, J. Field Robot. 30 (1) (2013) 64–86, http://dx.doi.org/10.1002/rob.21423.

[25] V. Sze, Y.-H. Chen, T.-J. Yang, J.S. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proc. IEEE 105 (12) (2017) 2295–2329, http://dx.doi.org/10.1109/JPROC.2017.2761740.

[26] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, P. Campoy, A review of deep learning methods and applications for unmanned aerial vehicles, J. Sensors 2017 (2017) 3296874, http://dx.doi.org/10.1155/2017/3296874.

[27] P. Zhu, L. Wen, X. Bian, H. Ling, Q. Hu, Vision meets drones: A challenge, 2018, arXiv:1804.07437.

[28] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, in: MCC '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 13–16, http://dx.doi.org/10.1145/2342509.2342513.

[29] N. Mohamed, J. Al-Jaroodi, I. Jawhar, H. Noura, S. Mahmoud, UAVFog: A UAV-based fog computing for internet of things, in: 2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2017, pp. 1–8, http://dx.doi.org/10.1109/UIC-ATC.2017.8397657.

[30] M. Chen, Y. Hao, Y. Li, C.-F. Lai, D. Wu, On the computation offloading at ad hoc cloudlet: architecture and service modes, IEEE Commun. Mag. 53 (6) (2015) 18–24, http://dx.doi.org/10.1109/MCOM.2015.7120041.

[31] G. Mohanarajah, D. Hunziker, R. D'Andrea, M. Waibel, Rapyuta: A cloud robotics platform, IEEE Trans. Autom. Sci. Eng. 12 (2) (2015) 481–493, http://dx.doi.org/10.1109/TASE.2014.2329556.

[32] W. Chen, B. Liu, H. Huang, S. Guo, Z. Zheng, When UAV swarm meets edge-cloud computing: The QoS perspective, IEEE Netw. 33 (2) (2019) 36–43, http://dx.doi.org/10.1109/MNET.2019.1800222.

[33] M. Mukherjee, V. Kumar, A. Lat, M. Guo, R. Matam, Y. Lv, Distributed deep learning-based task offloading for UAV-enabled mobile edge computing, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2020, pp. 1208–1212, http://dx.doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162899.

[34] D. Callegaro, M. Levorato, Optimal edge computing for infrastructure-assisted UAV systems, IEEE Trans. Veh. Technol. 70 (2) (2021) 1782–1792, http://dx.doi.org/10.1109/TVT.2021.3051378.

[35] S. Mahmoud, N. Mohamed, Broker architecture for collaborative UAVs cloud computing, in: 2015 International Conference on Collaboration Technologies and Systems (CTS), 2015, pp. 212–219, http://dx.doi.org/10.1109/CTS.2015.7210423.

[36] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39, http://dx.doi.org/10.1109/MC.2017.9.

[37] E. Petritoli, F. Leccese, M. Leccisi, Inertial navigation systems for UAV: Uncertainty and error measurements, in: 2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace), 2019, pp. 1–5, http://dx.doi.org/10.1109/MetroAeroSpace.2019.8869618.

[38] O.J. Woodman, An Introduction to Inertial Navigation, Tech. Rep. UCAM-CL-TR-696, University of Cambridge, Computer Laboratory, 2007, http://dx.doi.org/10.48456/tr-696, URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf.

[39] B.W. Parkinson, T. Stansell, R. Beard, K. Gromov, A history of satellite navigation, Navigation 42 (1) (1995) 109–164, http://dx.doi.org/10.1002/j.2161-4296.1995.tb02333.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.2161-4296.1995.tb02333.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2161-4296.1995.tb02333.x.

[40] M.H.F.M. Fauadi, S. Akmal, M.M. Ali, N.I. Anuar, S. Ramlan, A.Z.M. Noor, N. Awang, Intelligent vision-based navigation system for mobile robot: A technological review, Period. Eng. Nat. Sci. 6 (2) (2018) 47–57.

[41] J. Courbon, Y. Mezouar, N. Guénard, P. Martinet, Vision-based navigation of unmanned aerial vehicles, Control Eng. Pract. 18 (7) (2010) 789–799, http://dx.doi.org/10.1016/j.conengprac.2010.03.004, Special Issue on Aerial Robotics. URL https://www.sciencedirect.com/science/article/pii/S0967066110000808.

[42] S. Li, M.M. Ozo, C. De Wagter, G.C. de Croon, Autonomous drone race: A computationally efficient vision-based navigation and control strategy, Robot. Auton. Syst. 133 (2020) 103621, http://dx.doi.org/10.1016/j.robot.2020.103621, URL https://www.sciencedirect.com/science/article/pii/S0921889020304619.

[43] S.Y. Choi, D. Cha, Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art, Adv. Robot. 33 (6) (2019) 265–277.

[44] M. Paszkuta, J. Rosner, D. Peszor, M. Szender, M. Wojciechowska, K. Wojciechowski, J.P. Nowacki, UAV on-board emergency safe landing spot detection system combining classical and deep learning-based segmentation methods, in: Asian Conference on Intelligent Information and Database Systems, Springer, 2021, pp. 467–478.

[45] P. Mathur, Y. Jangir, N. Goveas, A generalized Kalman filter augmented deep-learning based approach for autonomous landing in MAVs, in: 2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA), IEEE, 2021, pp. 1–6.
[46] B.H.Y. Alsalam, K. Morton, D. Campbell, F. Gonzalez, Autonomous UAV with vision based on-board decision making for remote sensing and precision agriculture, in: 2017 IEEE Aerospace Conference, IEEE, 2017, pp. 1–12.
[47] J. Lee, J. Wang, D. Crandall, S. Šabanović, G. Fox, Real-time, cloud-based object detection for unmanned aerial vehicles, in: 2017 First IEEE International Conference on Robotic Computing (IRC), IEEE, 2017, pp. 36–43.
[48] A. Koubâa, B. Qureshi, Dronetrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet, IEEE Access 6 (2018) 13810–13824.
[49] A. Segalla, G. Fiacco, L. Tramarin, M. Nardello, D. Brunelli, Neural networks for pest detection in precision agriculture, in: 2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), IEEE, 2020, pp. 7–12.
[50] A. Albanese, M. Nardello, D. Brunelli, Automated pest detection with DNN on the edge for precision agriculture, IEEE J. Emerg. Sel. Top. Circuits Syst. 11 (3) (2021) 458–467, http://dx.doi.org/10.1109/JETCAS.2021.3101740.
[51] S. Liu, W. Deng, Very deep convolutional neural network based image classification using small training sample size, in: 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), IEEE, 2015, pp. 730–734.
[52] S. Jung, S. Hwang, H. Shin, D.H. Shim, Perception, guidance, and navigation for indoor autonomous drone racing using deep learning, IEEE Robot. Autom. Lett. 3 (3) (2018) 2539–2544.
[53] X. Xie, X. Han, Q. Liao, G. Shi, Visualization and pruning of SSD with the base network VGG16, in: Proceedings of the 2017 International Conference on Deep Learning Technologies, 2017, pp. 90–94.
[54] S.-H. Lee, C.-H. Yeh, T.-W. Hou, C.-S. Yang, A lightweight neural network based on AlexNet-SSD model for garbage detection, in: Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference, 2019, pp. 274–278.
[55] S. Gu, L. Ding, Y. Yang, X. Chen, A new deep learning method based on AlexNet model and SSD model for tennis ball recognition, in: 2017 IEEE 10th International Workshop on Computational Intelligence and Applications (IWCIA), IEEE, 2017, pp. 159–164.
[56] N. Tijtgat, W. Van Ranst, T. Goedeme, B. Volckaert, F. De Turck, Embedded real-time object detection for a UAV warning system, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 2110–2118.
[57] J. Redmon, A. Farhadi, YOLO9000: better, faster, stronger, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 7263–7271.
[58] I. Khokhlov, E. Davydenko, I. Osokin, I. Ryakin, A. Babaev, V. Litvinenko, R. Gorbachev, Tiny-YOLO object detection supplemented with geometrical data, in: 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), IEEE, 2020, pp. 1–5.
[59] L. Deng, G. Li, S. Han, L. Shi, Y. Xie, Model compression and hardware acceleration for neural networks: A comprehensive survey, Proc. IEEE 108 (4) (2020) 485–532, http://dx.doi.org/10.1109/JPROC.2020.2976475.
[60] R. Mishra, H.P. Gupta, T. Dutta, A survey on deep neural network compression: Challenges, overview, and solutions, 2020, arXiv preprint arXiv:2010.03954.
[61] S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang, et al., Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm, 2019, arXiv preprint arXiv:1903.09769.
[62] E.-H. Yang, H. Amer, Y. Jiang, Compression helps deep learning in image classification, Entropy 23 (7) (2021) 881.
[63] Z. Hu, X. Zou, W. Xia, S. Jin, D. Tao, Y. Liu, W. Zhang, Z. Zhang, Delta-DNN: Efficiently compressing deep neural networks via exploiting floats similarity, in: 49th International Conference on Parallel Processing-ICPP, 2020, pp. 1–12.
[64] M. Nagel, M. Fournarakis, R.A. Amjad, Y. Bondarenko, M. van Baalen, T. Blankevoort, A white paper on neural network quantization, 2021, arXiv preprint arXiv:2106.08295.
[65] M. Shafique, T. Theocharides, V.J. Reddy, B. Murmann, TinyML: Current progress, research challenges, and future roadmap, in: 2021 58th ACM/IEEE Design Automation Conference (DAC), IEEE, 2021, pp. 1303–1306.
[66] P. Puchtler, R. Peinl, Evaluation of deep learning accelerators for object detection at the edge, in: U. Schmid, F. Klügl, D. Wolter (Eds.), KI 2020: Advances in Artificial Intelligence, Springer International Publishing, Cham, 2020, pp. 320–326.
[67] S. Hossain, D.-j. Lee, Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with GPU-based embedded devices, Sensors 19 (15) (2019) http://dx.doi.org/10.3390/s19153371, URL https://www.mdpi.com/1424-8220/19/15/3371.
[68] B. Joudat, M.Z. Lighvan, The role of machine learning in IIoT through FPGAs, in: H. Karimipour, F. Derakhshan (Eds.), AI-Enabled Threat Detection and Security Analysis for Industrial IoT, Springer International Publishing, Cham, 2021, pp. 121–137, http://dx.doi.org/10.1007/978-3-030-76613-9_7.
[69] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, D. Chen, FPGA/DNN co-design: An efficient design methodology for 1oT intelligence on the edge, in: 2019 56th ACM/IEEE Design Automation Conference (DAC), 2019, pp. 1–6.
[70] R. Wu, X. Guo, J. Du, J. Li, Accelerating neural network inference on FPGA-based platforms—A survey, Electronics 10 (9) (2021) 1025.
[71] M. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, F. Hussain, Machine learning at the network edge: A survey, 2019, arXiv preprint arXiv:1908.00080.
[72] M.G.S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, F. Hussain, Machine learning at the network edge: A survey, ACM Comput. Surv. 54 (8) (2021) http://dx.doi.org/10.1145/3469029.
[73] C. Kyrkou, G. Plastiras, T. Theocharides, S.I. Venieris, C.-S. Bouganis, DroNet: Efficient convolutional neural network detector for real-time UAV applications, in: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2018, pp. 967–972.
[74] G. Plastiras, C. Kyrkou, T. Theocharides, Edgenet: Balancing accuracy and performance for edge-based convolutional neural network object detectors, in: Proceedings of the 13th International Conference on Distributed Smart Cameras, 2019, pp. 1–6.
[75] S. Bhardwaj, A. Mittal, A survey on various edge detector techniques, Proc. Technol. 4 (2012) 220–226, http://dx.doi.org/10.1016/j.protcy.2012.05.033, URL https://www.sciencedirect.com/science/article/pii/S221201731200312X. 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.
[76] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[77] T. Guo, J. Dong, H. Li, Y. Gao, Simple convolutional neural network on image classification, in: 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(, IEEE, 2017, pp. 721–724.
[78] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.
[79] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv preprint arXiv:1704.04861.
[80] F. Argenteri, Enhancing UAV capabilities with machine learning on board, 2020, URL https://github.com/frank1789/MasterThesis.
[81] S. Niu, Y. Liu, J. Wang, H. Song, A decade survey of transfer learning (2010–2020), IEEE Trans. Artif. Intell. 1 (2) (2020) 151–166, http://dx.doi.org/10.1109/TAI.2021.3054609.
[82] F.L. Da Silva, A.H.R. Costa, A survey on transfer learning for multiagent reinforcement learning systems, J. Artificial Intelligence Res. 64 (2019) 645–703.
[83] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, Proc. IEEE 109 (1) (2021) 43–76, http://dx.doi.org/10.1109/JPROC.2020.3004555.