AI-Enabled Autonomous Drones for Fast Climate Change Crisis Assessment

Daniel Hernández, Juan-Carlos Cano, Federico Silla, Carlos T. Calafate, and José M. Cecilia

***Abstract*—Climate change is one of the greatest challenges for modern societies. Its consequences, often associated with extreme events, have dramatic results worldwide. New synergies between different disciplines, including artificial intelligence (AI), Internet of Things (IoT), and edge computing can lead to radically new approaches for the real-time tracking of natural disasters that are also designed to reduce the environmental footprint. In this article, we propose an AI-based pipeline for processing natural disaster images taken from drones. The purpose of this pipeline is to reduce the number of images to be processed by the first responders of the natural disaster. It consists of three main stages: 1) a lightweight autoencoder based on deep learning; 2) a dimensionality reduction using the *t*-distributed stochastic neighbor embedding algorithm; and 3) a fuzzy clustering procedure. This pipeline is evaluated on several edge computing platforms with low-power accelerators to assess the design of intelligent autonomous drones to provide this service in real time. Our experimental evaluation focuses on flooding, showing that the amount of information to be processed is substantially reduced, whereas edge computing platforms with low-power graphics accelerators are placed as a compelling alternative for processing these heavy computational workloads, obtaining a performance loss of only 2.3× compared to its cloud counterpart version, running both the training and inference steps.**

***Index Terms*—Artificial vision, climate change, deep learning (DL), edge computing, sustainable ICT, unmanned aerial vehicles (UAVs).**

I. Introduction

CLIMATE has dramatic consequences all over the world, with effects having noticeably negative results [1]. The consequences of floods are undoubtedly one of the most dramatic ones among the many natural disasters, as they encompass loss of human life and loss of natural ecosystems. Floods also cause economic losses. Indeed, the effects of natural disasters have consequences where immediacy in decision making is essential. Improving preparedness for an effective response to these events is essential in situations where every minute counts. In this regard, technological advances can help to achieve this efficiency in response times, where sustainability, efficiency, and ubiquity should be the main ingredients of these developments [2].

Unmanned aerial vehicles (UAVs), commonly referred to as drones, are autonomous unmanned aircrafts that are widely used for different applications and tasks. The use of drones has gradually evolved from more recreational areas, such as photography and video, to more technical ones, such as border surveillance, precision agriculture, and infrastructure inspections, just to name a few [3]. Drones are also playing an important role in emergency and response protocols. They are currently widely used in the response stage and are also used, albeit to a lesser extent, in the other stages of a natural disaster, i.e., prevention, preparation, and recovery [4].

Natural disaster management situations are very stressful, and the use of technological tools such as drones could be helpful. However, using drones requires qualified personnel that can monitor and process the information generated by these tools. In particular, drones can generate an enormous amount of video and images that need to be analyzed by experts in conditions where it is very easy to make mistakes. The probability of errors can be reduced by using image processing techniques for the detection of potential risks in a natural disaster, including machine learning techniques (ML) and deep learning (DL), which can automate the process of image interpretation and clustering in order to speed-up decision making by managers, and to avoid possible human errors. However, these artificial intelligence (AI)-techniques, particularly their training, is a compute-intensive process, and although there is an industry-wide trend toward hardware specialization to improve performance and energy consumption [5], traditionally these workloads have been executed in a cloud-fashion approach. Nevertheless, the rescue of people, the identification of affected areas, and the prevention of the secondary effects of a natural disaster are all emergency tasks and, therefore, the information should be processed in real time, or at least, as quickly as possible.

An alternative that is emerging in the last decade is edge computing [6]. In edge computing, data processing is performed, totally or partially, on the devices that are at the edge of the network; i.e., at those devices that are closest to mobile devices or sensors. This distributed way of computing provides energy savings, scalability and responsive Web services for mobile computing, and offers a mechanism for data privacy in the Internet of Things (IoT) context. In addition, it offers

Manuscript received April 15, 2021; revised July 6, 2021; accepted July 12, 2021. Date of publication July 19, 2021; date of current version May 9, 2022. This work was supported in part by the Spanish Ministry of Science and Innovation under Grant RYC2018-025580-I, Grant RTI2018-096384-B-I00, and Grant RTC2019-007159-5; in part by the Fundación Séneca under Project 20813/PI/18; and in part by the "Conselleria de Educación, Investigación, Cultura y Deporte, Direcció General de Ciéncia i Investigació, Proyectos" under Grant AICO/2020/302. *(Corresponding author: Daniel Hernández.)*

The authors are with the Computer Engineering Department (DISCA), Universitat Politècnica de València, 46022 Valencia, Spain (e-mail: dhervic@doctor.upv.es; jucano@disca.upv.es; fsilla@disca.upv.es; calafate@disca.upv.es; jmcecilia@disca.upv.es).

Digital Object Identifier 10.1109/JIOT.2021.3098379

the possibility to mask transient cloud outages. Edge computing devices should be designed in such a way that energy efficiency is the main objective. To this end, leading processor companies are developing energy-efficient solutions with low power consumption and high performance. In particular, the Nvidia Jetson family of embedded systems can be highlighted, which include low-power graphics accelerators (GPUs) that deliver good performance for massively parallel applications with power consumption between 7.5 and 10 W [7].

In this article, we propose an AI-based pipeline for the identification of the drone-based images that are related to floods. We use a deep-learning-based autoenconder to highlight the main features of the images taken from the drones. Then, those features are reduced and clusterized to help first responders of natural disasters in dealing with large data sets. We also evaluate the AI-based pipeline in different low-power GPU-based edge computing platforms to figure out if they would be a compelling alternative to the main aim of developing an AI-based autonomous drone for emergency situations. Hence, the major contributions of this article are as follows.

1) A deep-learning-based lightweight autoencoder is proposed to identify the main features of aerial flood images.
2) An AI-based pipeline to reduce the amount of information to be supervised by first responders in natural disasters is designed.
3) An in-depth performance evaluation of different low-power GPU-based edge computing devices is provided to assess the feasibility of autonomous AI drones in natural disasters.
4) A particular case study that targets flooding scenarios is under study.

The remainder of this article is structured as follows. Section II provides the required knowledge related to the main research areas of this work. Section III introduces the general infrastructure of the AI-based pipeline to deal with aerial images of natural disasters. Section IV shows the experimental setup before showing the performance and quality evaluation of our approach in Section V. Finally, Section VI shows the conclusion and directions for future work.

II. Background and Related Work

This section provides the main background on the different topics related to this article: UAVs, DL for artificial vision, and edge computing.

A. UAVs in Natural Disasters

UAVs have recently experienced unprecedented growth, with countless areas of application foreseen for the coming years [8]. Regarding works proposing the use of drones for rescue and survivor search tasks, the first works in the area [9] studied the use of a single drone for this endeavour. Years later, drones have been used to analyze the effects of a landslide in Tibet [10], comparing the profile of the terrain before and after the catastrophe. More recently, Mehrdad *et al.* [11] proposed a technique specifically applicable to natural disasters that is able to quickly and efficiently combine aerial images, and which has direct applicability in the case of using a swarm of drones to obtain such images. However, if we are attempting to perform complex tasks in a short time, such as assessing the effects of a natural disaster and searching for survivors, the deployment of swarms of UAVs is a very interesting alternative. Drone swarms can improve the efficiency of individual systems by offering many advantages, including the possibility of extending mission coverage in a short time, thanks to the cooperation between UAVs [12].

To date, very few tests have been conducted with outdoor multicopter swarms, and even fewer have been conducted on large scale, the most notable to date being the test conducted by China, which managed to coordinate up to 1000 UAVs for the first time ever [13]. This lack of works in the literature is due to the fact that using a swarm of drones collaborating with each other to perform a cooperative task presents significant communication, synchronization, and quality-of-service issues [14].

B. Artificial Vision in Natural Disasters

Natural disasters present characteristics where immediacy in decision making is fundamental. There are image processing techniques for the detection of potential risks in a natural disaster, including ML techniques, such as support vector machines (SVMs), Bayesian nonparametric models, genetic algorithms (GAs), random forest (RF), fuzzy clustering (FC), or K-nearest neighbors (KNNs), that are used for image classification. Zhang *et al.* [15] proposed an extended motion diffusion-based (EMD) to detect changes in airport ground. Its method was verified from the airport ground video surveillance (AGVS) benchmark test by obtaining positive results in situations, such as fog and camouflage. In [16], a GA combined with a neural network was proposed to classify images coming from a flood; this proposal was compared with three FC methods, being that the proposed algorithm was able to obtain better results. Nijhawan *et al.* [17] proposed a hybrid framework composed of a DL algorithm, specifically a convolutional neural network (CNN), and a feature extraction algorithm, to classify images of natural disasters, such as avalanches, cyclones, tornadoes and fires, among others. The data used to test this framework were an artificial data set created by the authors. The proposed CNN is compared to RF, SVN, and KNN techniques, obtaining the CNN the best result. The same happens with the CNN proposed in [18]. In that study, the authors proposed the use of a CNN for flood image classification using images obtained from a UAV, producing such offline image classification. The authors made a comparison with an SVM technique, obtaining better results with the proposed CNN-based technique. In short, the techniques based on DL are the ones that achieve the best performance in image classification of natural disasters so far.

There are some works in the literature where DL techniques are applied for image classification in general and, in particular, for images of natural disasters. However, these techniques consume a lot of computational resources, and image classification is performed offline. However, the rescue of people, the identification of affected areas, and the prevention of

secondary effects of a natural disaster, are emergency tasks requiring information to be processed in real time. Hence, our article explores the design of these techniques so that they can be executed in low-power processors that can be introduced into the UAV swarm; this allows performing real-time image processing from different perspectives (thanks to the swarm and coordination of the UAVs) to make effective and accurate decisions in real time.

C. Edge Computing Platforms

In the history of computing, the paradigms of centralized and decentralized computing have alternated over time. In the early days, computing was developed using centralized processing with batch and time-sharing techniques. The development of personal computers in the 1980s brought about a shift to a decentralized approach. This approach was decentralized at the beginning of the 21st century with cloud computing. Cloud computing has now established itself as the most widely used approach, mainly driven by the rise of mobile devices and the IoT, for which cloud computing offers high-performance computing and storage services that are not available on these low-power, low-cost devices. However, the nearest cloud infrastructure running mobile and IoT application services may be too far away from the source of data.

Satyanarayanan [6] proposed a two-level architecture to pursue interactive performance of mobile applications. A first level consisting of a traditional cloud and a second level consisting of a network of cloudlets; i.e., dispersed elements containing state information cached from the first level [19]. In addition, Bonomi *et al.* also proposed a multitier architecture that they called fog computing. In this case, the authors designed this architecture motivated by the lack of scalability of IoT infrastructures [20]. As in the case of edge computing, the proximity of cloudlets (or fog nodes) to the nodes capturing data offers a number of benefits, in addition to the scalability benefits initially sought by the authors. These benefits include the availability of highly responsive cloud services, the reduction of end-to-end latency, the increased bandwidth and low jitter to services located at the edge, etc. Chen *et al.* [21] studied computation offloading in the fifth-generation networks and proposed a distributed learning method to address the technical challenges arising from uncertainties and limited resource sharing in an multiaccess edge computing (MEC) system. They provided a case study on resource orchestration to show the potential of the proposal, outperforming benchmark resource orchestration algorithms.

Edge computing provides computing power in close proximity to sensors or mobile devices. In fact, as mentioned above, there are compute-intensive applications for which this technology is opening up new development opportunities such as interactive mobile applications for augmented reality. Undoubtedly, the design of cloudlets has to be highly energy efficient, while providing the highest computational horsepower possible. Actually, microprocessor industry is releasing Systems on Chip (SoCs) that include low-power accelerators, such as graphics processing units (GPUs) or tensor processing units (TPUs). Among them, we may highlight the Nvidia's Jetson family [22], Intel's Movidius [23], or the Google's Coral project [24]. Thanks to these accelerators, the energy efficiency of edge devices can substantially increase.

Another important feature of edge/fog computing related to this work is the reduction of the amount of data that needs to be transferred to the cloud. This has great benefits, such as the reduction of network overhead, energy savings, cost reduction in the cloud, reduction of storage space, etc. For instance, Simeons *et al.* [25] developed a video processing system, known as GigaSight, where videos obtained from mobile devices are processed in the nearest cloudlet, sending only the results and some metadata to the cloud, drastically reducing the application's bandwidth and storage needs. This feature is of particular interest for the UAV environment, where autonomy is scarce. Furthermore, in the particular case of natural disasters, the reduction of data delivery through edge processing provides clarity in analyzing the information for first responders.

III. AI-Pipeline Proposed for Management of Natural Disasters

Natural disasters require immediacy so that decisions can be taken as quickly as possible to save lives. First-responders need tools that allow them to quickly assess the magnitude of the natural disaster. As previously mentioned, drones are capable of exploring wide areas inaccessible by first-responders, allowing a large number of images to be taken to assess the impact of the disaster. However, manually processing this large amount of information is very difficult for humans, even more in these types of critical scenarios.

This section introduces the AI-pipeline proposed to deal with unclassified drone-based images of natural disasters. The main objective of this AI-pipeline is to reduce the number of drone-generated images to be processed by the first-responders. This is developed through an unsupervised process that identifies the main features of the images through a deep-learning based autoencoder and reduces the dimensionality of these features to eventually perform a clustering process to group those images by similarity. This AI-pipeline outputs an image that represents each cluster. This image can be evaluated by first-responders to determine whether that group of images is of interest for decision making. Another important feature of natural disasters is that they usually occur in remote locations, where connectivity is limited. For this purpose, the last part of this work evaluates the complete execution of the proposed AI-pipeline on edge computing platforms, so that Internet connection will not be required to obtain the benefits of this system.

A. Deep Learning-Based Autoencoder Structure

An autoencoder is a neural network architecture designed for feature learning from unlabeled data. It has a distinctive shape consisting of two layers; the first one is the encoder which is responsible for data compression so that it becomes smaller in size than its input by decreasing its dimensionality from the input layer to a central information layer. The other is the decoder, which attempts to regenerate the input data

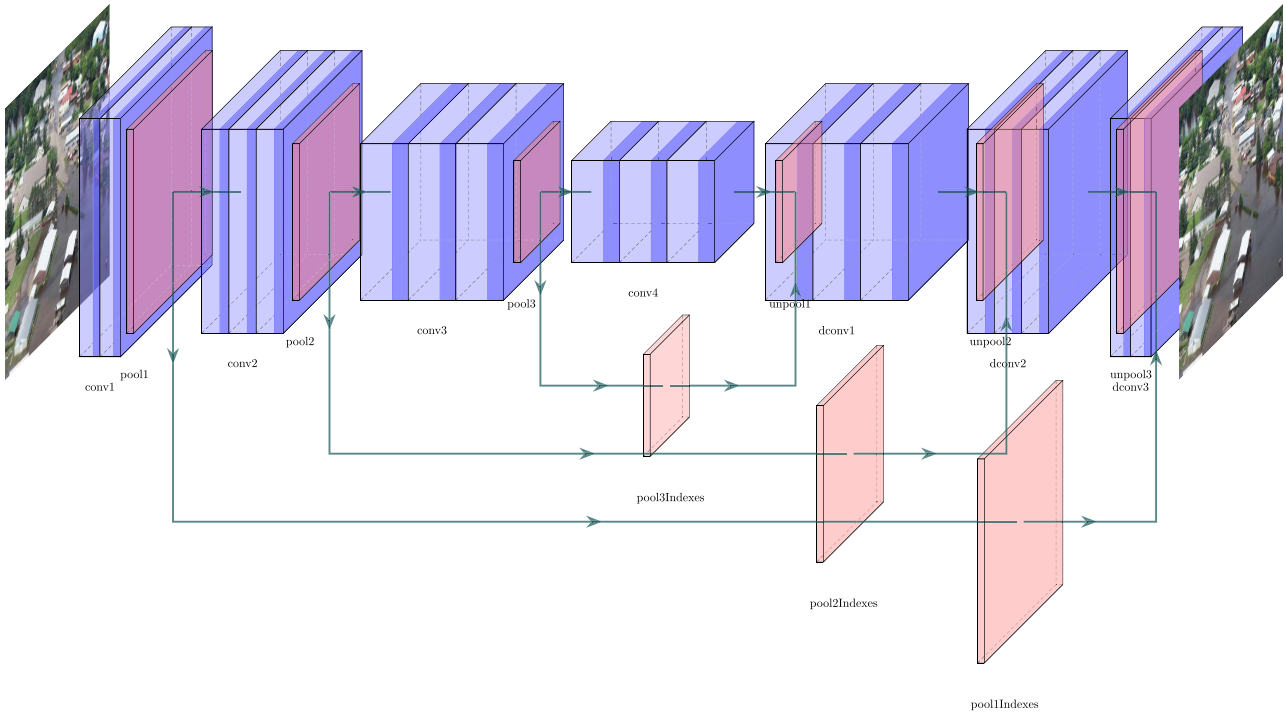


Fig. 1. DL-based autoenconder.

compressed by the first layer to regenerate the original input to the encoder phase as faithfully as possible.

The data to be processed is formed by a set of images, therefore the neural network that forms the autoencoder is composed of convolutional networks that behave better for this type of scenario. This type of autoencoder is known as convolutional autoencoder (CAE). The structure of the proposed autoencoder for this work is described in Fig. 1. For its design, we have tried to prioritize simplicity and the reduction of the total number of parameters as much as possible, since this network will be trained and used for the clustering process entirely in the devices located at the edge, and therefore the limitations of memory and computational capacity have been taken into account.

One of the differences with traditional autoencoders is that no fully connected layers have been used within the autoencoder. Moreover, average pooling operations are performed on the feature map extracted from the filters of the central layer for the feature extraction process. The blocks that compose the CAE are convolutional, as well as the pooling layers for the encoding and compression phase, and deconvolutional and unpooling blocks for the regeneration phase of the original input.

Convolutional blocks, called "conv" in Fig. 1, are responsible for filtering an input to create a feature map that summarizes the presence of features detected in that input. In contrast, deconvolutional elements, called "deconv" in Fig. 1, apply a 2-D transposed convolution operator on an input image composed of several input planes. This operation can be viewed as the gradient of Conv2d with respect to its input, also known as fractional convolution. With this operation, we will decompress the abstract representation generated by the convolutional layers into something more visual.

Max Pooling, "pool" in Fig. 1, is a sample-based discretization process designed to filter out noisy activations by retaining only the robust activations in the upper layers, but the spatial information is lost during pooling. This reduces its dimensionality and allows assumptions to be made about the features contained in the binned subregions. Unpooling, "unpool" in Fig. 1, is the opposite operation. It captures example-specific structures by tracing the original locations with strong activations back to image space. As a result, it reconstructs the detailed structure that was done in the pooling phase. Pooling and unpooling layers do not have tuning parameters, although they will have to share parameters between them since indices generated by each of the pooling layers in the encoding part have to be sent to their respective unpooling layer when decoding in order to reconstruct the original dimensionality prior to the pooling operation. This operation can be seen in layers *poolxIndexes* in Fig. 1.

B. *t*-SNE

The second main step of the proposed pipeline is a dimensionality reduction. The objective of this step is twofold: 1) for easy viewing of information and 2) for reducing the computational complexity. We use a *t*-distributed stochastic neighbor embedding (*t*-SNE) algorithm [26], which is a nonlinear technique that reduces the number of dimensions of the input data. More specifically, we use the *t*-SNE implementation made available by the scikit-learn Python library [27]. The algorithm searches for joint probabilities based on similarities between data points. In our case, each data point is the average flatten pooling obtained from the convolutional autoencoder. This input is a 240-feature vector, and we apply *t*-SNE to obtain a

2-D one before performing the clustering. The t-SNE algorithm attempts to minimize the so-called Kullback–Leibler (KL) divergence between the input data and the joint probabilities of the low-dimensional embedding. This divergence is a way to measure the difference between two distributions. The t-SNE procedure follows the following equations. First, (1) shows the calculation of the conditional probability of point x_j to be next to point x_i

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma^2\right)}{\sum_{k\neq i}\exp\left(-\|x_i - x_k\|^2/2\sigma^2\right)}. \tag{1}$$

Then, the joint probability distribution is calculated based on the conditional distributions [see (2)]

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k\neq l}\left(1 + \|y_i - y_l\|^2\right)^{-1}}. \tag{2}$$

Finally, the KL divergence is calculated for both distributions P and Q in the probability space of x to optimize their distribution [see (3)]

$$\mathrm{KL}(P\|Q) = \sum_i P(i)\ln\frac{P(i)}{Q(i)}. \tag{3}$$

C. Clustering Algorithm

Clustering algorithms are data analysis techniques that organise n-dimensional data points into groups or *clusters*. Each cluster is composed of those points that are most similar to each other based on a metric defined in the algorithm [28]. Several clustering algorithms have been proposed in the literature as applied to different scientific applications [29], [30]. They are mainly divided into two main groups, i.e., hard and soft clustering techniques. In hard clustering techniques, such as the well-known k-means algorithm, a data point can only belong to one group. However, in soft clustering techniques, a probability of belonging to each group is assigned to each data point. Among the soft clustering techniques, we may highlight the fuzzy c-means (FCM) algorithm [31] and the fuzzy minimal (FM) algorithm [32]. Another important feature of clustering algorithms is the number of clusters to be generated; k-means and FCM require us to define the number of clusters to be developed. In contrast, FM does not need this parameter to be set in advance without the need for the clusters to be compact well-separated (CWS). This feature enables unsupervised clustering that does not condition the sets of images to be obtained from each drone mission and, thus, it is what motivated us to introduce FM at the final stage of the pipeline proposed in this article. In what follows, we briefly introduce the FM algorithm and refer the reader to [32] and [33] for insights.

The FM algorithm is an iterative fixed-point algorithm whose main purpose is to minimize an objective function given by (4) and (5). The FM algorithm has two main functions shown in Algorithm 1. The FM needs two input parameters: $varepsilon_1$, which sets the maximum allowable degree of error, and $varepsilon_2$, which shows the difference between the potential minima. Once these parameters are set, the r factor is calculated for the input data set before calculating the prototypes

$$J_{(v)} = \sum_{x\in X}\mu_{xv}\cdot d_{xv}^2 \tag{4}$$

where

$$\mu_{xv} = \frac{1}{1 + r^2\cdot d_{xv}^2}. \tag{5}$$

The r-factor is a nonlinear function for measuring the degree of homogeneity and isotropy breakdown of a data set. This function is shown in (6). Factor r takes as a partition hypothesis that clusters are created when isotropy and/or homogeneity is broken

$$\frac{\sqrt{|C^{-1}|}}{nr^F}\sum_{x\in X}\frac{1}{1 + r^2 d_{xm}^2} = 1 \tag{6}$$

where $|C^{-1}|$ is the determinant of the inverse of the covariance matrix, m is the mean of the sample X, d_{xm} is the Euclidean distance between x and m, and n is the number of elements of the sample.

Algorithm 2 shows the calculation of prototypes developed by the FM approach. This procedure is based on a scoring function that has, as an argument, the previously calculated r factor. The membership function is described in (5), which measures the probability of an element x to belong to a particular cluster in which v is the prototype. In fact, these prototypes are the output of the FM that represents the most significant images of our pipeline.

Algorithm 1 FMs Algorithm Pseudocode

```
LoadDataSet()
Choose input variables ε1 and ε2.
r=CalculateFactorR(dataset)
PrototypeCalculation(dataset, r)
```

Algorithm 2 Baselines of the FM *PrototypeCalculation*. n Is the Size of the Input Data Set. V Is a Vector With the Prototypes Found by FM. F Is the Dimensionality (2 in Our Case)

```
Initialize V = { } ⊂ R^F.
for i = 1; i < n; i++ do
    v(0) = x_i, t = 0, E(0) = 1
    while E(t) ≥ ε1 do
        t = t + 1
        μ_xv = 1/(1+r^2·d_xv^2), using v(t−1)
        v(t) = Σ_{x∈X}(μ_xv^(t))^2 · x / (μ_xv^(t))^2
        E(t) = Σ_{α=1}^{F}(v(t)^α − v(t−1)^α)
    end while
    if Σ_α^F(v^α − w^α) > ε2, ∀w ∈ V then
        V ≡ V + {v}.
    end if
end for
```

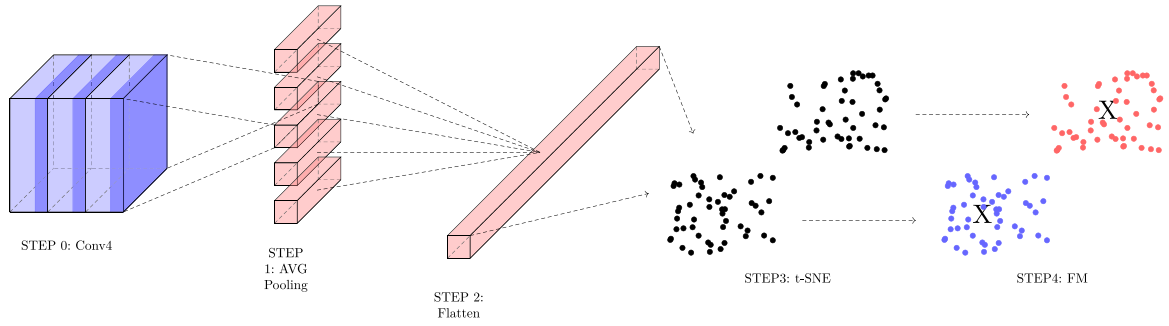


Fig. 2. AI-pipeline for the latent space clustering extracted from the autoencoder.

D. AI-Pipeline Ensemble

Fig. 2 shows the proposed AI-pipeline applicable after CAE training. The pipeline begins with the raw images that go through the autoencoder shown in Fig. 1. The trained autoencoder is able to compress the main features in the conv4 layer. This layer outputs a matrix with a transformation of the input image that has been obtained by applying a set of filters. At this point, the layers behind conv4 will be removed, and this will be the new output layer of the network. Then, an iteration of the whole data set will be performed to obtain all the filters extracted from the network for each image.

Each image processed by the autoencoder is converted into a set of matrix-represented filters.The Average (AVG) polling is applied on 2×2 blocks to each of these filters in order to create a downsampled (pooled) feature map as shown in step 2 at Fig. 2. The *AVG polling* obtains an 1-D vector for each filter by calculating the mean of each block of each filter. This means that each 2×2 square of each filter is sampled downward to the mean value of the square. These 1-D vectors will be concatenated to feed step 3 called *Flatten* in Fig. 2, forming a single vector where all relevant CAE information will be extracted.

Once a flatten vector for each of the images is obtained, the *t*-SNE algorithm is applied so that all images will be embedded into a 2-D array grouped by the similarity detected after applying *t*-SNE. It is important to note that this matrix contains all the images without creating any clusters; i.e., all images are considered as homogeneous points. Then, this matrix is clustered based on image similarity. Since the optimal number of clusters to be obtained is not known, a fuzzy logic-based clustering will be applied using the FM algorithm, where the coordinates of the most representative images of the data set (i.e., the prototypes of the cluster) will be obtained along with the percentage of belonging to each of the image clusters generated. With this extracted information, every image can be labeled for subsequent submission. Images representing the prototypes will be sent for follow-up by the first responders.

E. Solution Deployment and Execution Flow

Having defined the problem and the proposed AI-pipeline, Fig. 3 shows the execution flow that should be carried out in real natural disaster management scenarios. Drones start from a standby position (step 1) in order to prepare for take-off before they begin taking images of the affected area. Once images have been taken in step 2, the proposed AI-pipeline is executed (step 3). The first step of this pipeline would be the training of the autoencoder, which is performed with the images captured by the drone during the first flight. It is important to note that this training is only performed once for each affected area, and the information learned by the autoencoder can be reused to encode the images of the following flights over the same affected area. Therefore, the training stage, the most computationally expensive, would be executed once and the inference stage would be executed as many times as surveillance missions are performed by the drone. In other words, the training stage will provide the neural network weights that will be used in the following missions in the inference stage, running only the first part of the network (i.e., Conv1–Conv4 in Fig. 1). After the inference stage, the feature vector will be generated and reduced using *t*-SNE and clustered using FM. Finally, once the most significant images have been selected with the execution of the AI-pipeline, they are sent in step 4.

Regarding the execution flow, all steps of the proposed AI-pipeline could be executed at the edge. The pipeline is designed to have a low memory footprint, as it will be shown in Section IV. Furthermore, the pipeline only uses the images captured during the mission (step 2 in Fig. 3). Therefore, there is no need to perform a knowledge transfer with the pretrained network weights using other data sets.

IV. Experimental Setup

This section provides an overview of the data set used to train and test the AI-pipeline proposed in Section III. Then, the main hardware features and software details of our experimental environment are described.

A. Data Set

The AI-pipeline previously introduced requires a data set to train the autoencoder. To the best of our knowledge there are few data sets that meet our requirements; i.e., natural disaster and aerial images. Particularly, we use aerial image data set for emergency response applications (AIDER) [34], [35] for the training and testing procedures, as will be shown in Section V. This data set contains images from five different categories. Four of them are related to disaster events, such as Fire/Smoke, Flood, Collapsed Building/Rubble, and Traffic Accidents, and

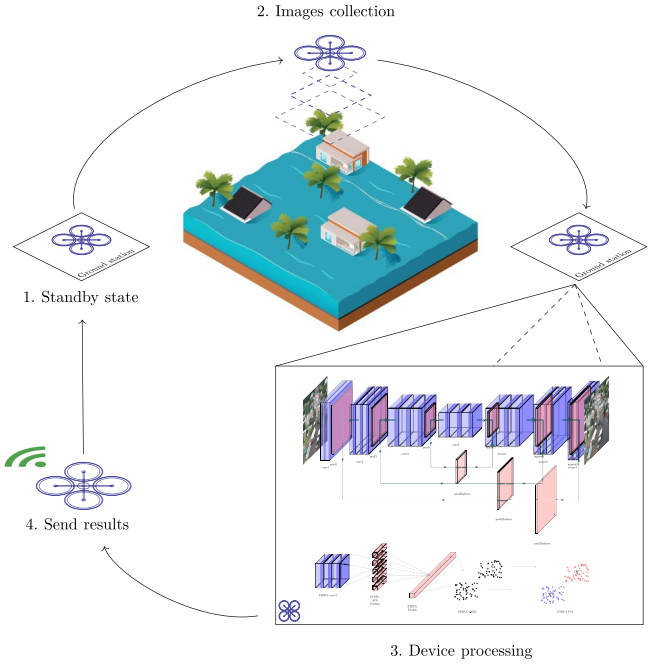


Fig. 3. Natural disasters management overview.

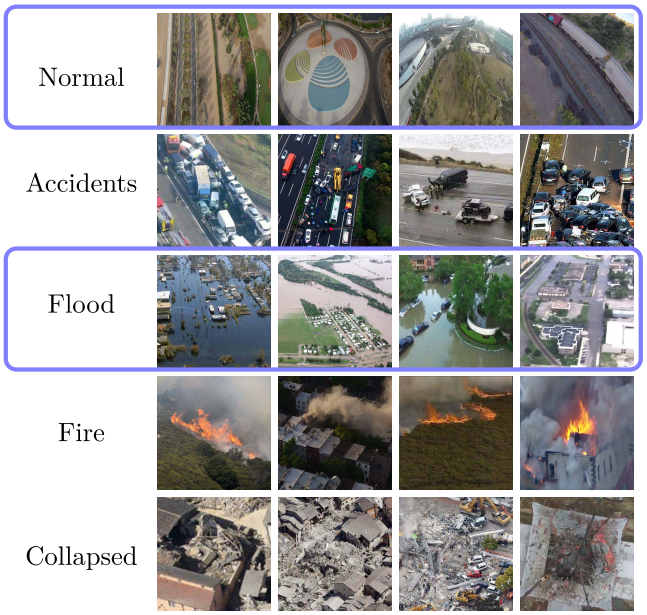


Fig. 4. Classes within the AIDER data set.

the latter is the control case; i.e., there is not any sort of accident on it. Fig. 4 shows all categories contained in this data set with random examples of them to illustrate its content. It is important to note that AIDER is only composed of aerial images that were obtained by several online sources, such as Google or Bing images, Youtube or news agencies websites. Particularly, authors used the keywords "Aerial View," "UAV," and/or "Drone," along with the particular event they wanted to include in the data set, such as "flood" or "fire." Moreover, images also have different viewpoints, resolutions, and illumination conditions. It is important to note that authors manually inspected all images to make sure that they are related to the expected disaster, and also that the event is centered at the image. The latter is to guarantee that any geometric transformation during augmentation does not remove the object of interest from the image. Finally, the data set is not well balanced to replicate real world scenarios; i.e., it contains more images from the control class. In particular, the data set is composed of about 500 images for each disaster class, and over 4000 images for the control class. In our case, the data set is even more imbalanced as we have removed the images from Fire/Smoke, Collapsed Building/Rubble, and Traffic Accidents and will only use the control class and the flooding class. As shown in Fig. 4, we will use the set made up of the normal and flood images as the framework on which we will perform the clustering tasks. Before being processed by the clustering pipeline, all images have been resized to a side size of 255 pixels, and all of them have been cropped at the center.

B. Hardware and Software Environment

This section introduces the hardware and software environment used to perform the experiments in Section V (see Table I). We focused on four different architectures: a high performance computing (HPC) node called Pedra, and three low-power edge computing devices from the NVIDIA Jetson family (i.e., Jetson nano, Jetson TX2, and Jetson AGX Xavier). Although Pedra cannot be mounted directly on the UAVs (due to its weight, size, and energy consumption), it could be used via a cloud solution when mobile Internet speed and coverage are sufficient. The main purpose of this comparison is to determine whether the AI-pipeline proposed in Section III adapted for GPUs decreases the calculation time, if it can also be performed in the edge in a reasonable time frame and, if so, which platform is the most suitable one. Table I introduces the main features of the hardware platforms targeted.

The software environment is based on gcc v7.4.0, CUDA v10.2 with cuDNN and Python v3.6 with pytorch v1.8.0 built for edge devices, and torchvision v0.9.0 built for edge devices and scikit-learn v0.24.1.

On the other hand, note that the training stage of the neural network described above has been performed on all the devices described in Table I in order to achieve a process that is able to run on the edge from the beginning to the end. This approach differs from the usual practice of first training on a high-performance cluster, and then trying to apply techniques to adapt the trained model to low-performance devices. In our case, we designed a model with a total size of 88 275 trainable parameters (the lightness of the network is evident) that is efficient enough for the target task. The main parameters of the training procedure were 30 epochs, MSELoss loss criterion, and Adam Optimizer. The batch size to feed each epoch has been adapted depending on the memory limitations of each device that was running the training process at that time.

It is worth mentioning that although this AI-pipeline is designed to train a new autoencoder for each available data set, if the new images are similar to those used in a previous data set, e.g., the same geographical area or same day time, with similar weather, a previously trained encoder could be reused

TABLE I
SPECIFICATION OF THE VARIOUS GPU PLATFORMS USED IN OUR EXPERIMENTS

	Pedra	Jetson AGX Xavier	Jetson TX2	Jetson Nano
CPU	Intel Silver 4216	NVIDIA Carmel ARM v8.2	ARMv8	ARM Cortex-A57 MPcore
2xGPU (NVIDIA)	GeForce RTX 2080 Ti	Volta	Pascal	Maxwell
Memory [Gb]	12 DDR5	32 LPDDR4x	8 LPDDR4	4 LPDDR4
Size [mm]	73.4 x 8.7 x 44.8	105 x 105	50 x 87	70 x 45
Weight [g]	17,000	280	85	61
Energy consumption [W]	80-100	10-30	7.5	3-5

to obtain the most relevant images for this new particular scenario. Another option could be to perform new incremental training epochs starting from a previously trained autoencoder to which new images are added, thus reducing the convergence time with respect to performing a training process from the beginning.

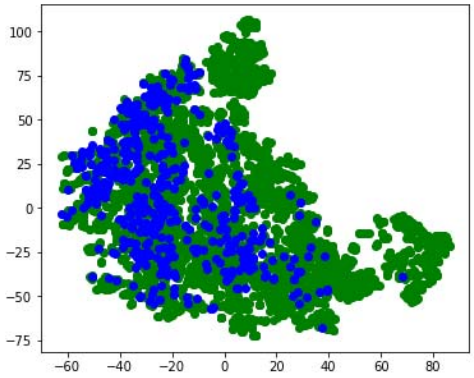
V. EVALUATION

This section shows the quality results obtained by training and validating the aerial images data set shown in Section IV-A. Moreover, the performance of several CPU-GPU-based computing solutions is presented to evaluate edge computing platforms as potential infrastructure for processing AI-based workloads on drones.

A. Quality Evaluation and Memory Footprint

As explained in Section III-D, all images of the target data set are fed into the pipeline. First, the autoencoder compresses the information into a feature vector that is reduced to two dimensions for later visualisation. Such a visualisation is shown in Fig. 5. In Fig. 5(a), the images are represented as data points where the blue points are images related to floods, and green points are the other images. Fig. 5(b) shows the clustering generated by *t*-SNE, which creates an image cloud where, for each coordinate extracted from the clustering phase, the original image corresponding to the index of that position has been drawn. It is important to note that, although the data set is labeled so that the class of each image is known in advance, our methodology does not use this information at any point. This information is used only for evaluation purposes, and to figure out which images are actually related to the flooding class.

The cluster prototypes generated by the FM algorithm are shown in Fig. 6(a), where 48 different groups have been detected for the targeted data set. It is worth noting that the main objective of this work is to synthesize the information sent by the drone in an unsupervised manner. In this way, the drone would only send 48 out of the 5500 images captured during the mission. These 48 images are the most representative ones in the data set. From these images, the natural disaster managers would be able to identify which images are of real interest for their work, being able to access all the images in that cluster if required. In particular, Fig. 6(b) shows the



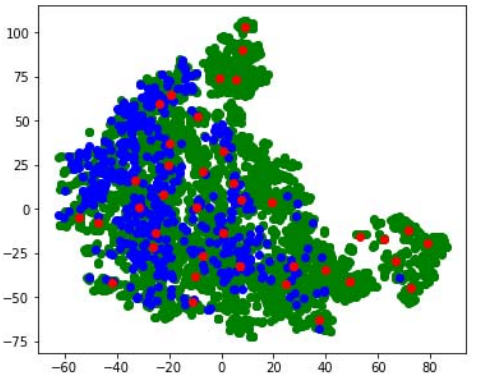
(a)

(b)

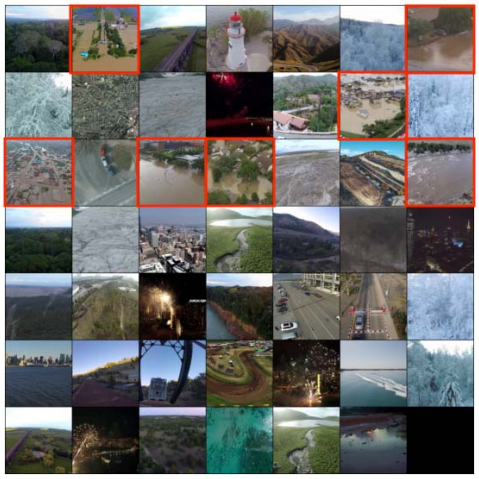
Fig. 5. Data points and images after running the t-SNE algorithm. (a) Data points for flooding images (blue), and others (green). (b) Images corresponding to data points.

prototype images that are related to flooding (highlighted in red).

Finally, it is worth mentioning that there are DL models, which can be used for feature extraction, and that are usually pretrained on data sets, such as MobileNetV2 [36], Inception V3 [37], ResNet50 [38], or VGG16 [39]. The use of these pretrained models can be a good alternative to an autoencoder in high-performance environments where the highest possible accuracy is required. However, these models have a high memory footprint, and they are very heavy for edge computing devices. Fig. 7 shows a memory footprint comparison between these models and our proposal, Edge CAE. It can be seen that



(a)



(b)

Fig. 6. Data points and images after running the FM algorithm. (a) Data points of flooding images (blue), others (green), and prototypes obtained by the FM algorithm (red). (b) Images closer to the prototype found by FM.

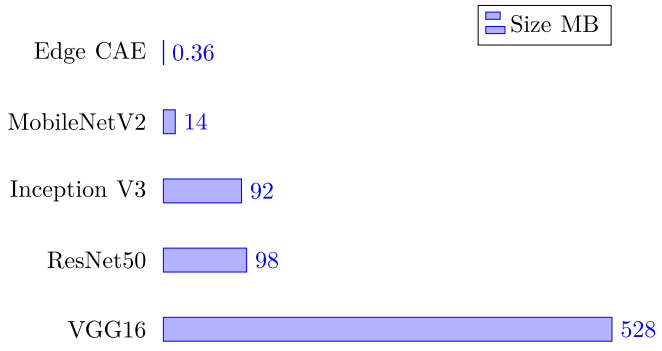


Fig. 7. Memory footprint of similar models used for feature extraction.

our proposal is 38× lighter than the lightest proposal in the state-of-the-art, and orders of magnitude lighter than the rest. It makes sense, as these networks have been designed for supercomputing environments, which makes them not feasible in resource-constrained IoT environments.

A further consideration is that these models have been pretrained on classified images using a different sample than the type of images used for the clustering phase. Therefore, this proposal intends to perform a sandbox execution from start to end without relying on data other than that collected by the drone in an actual mission.

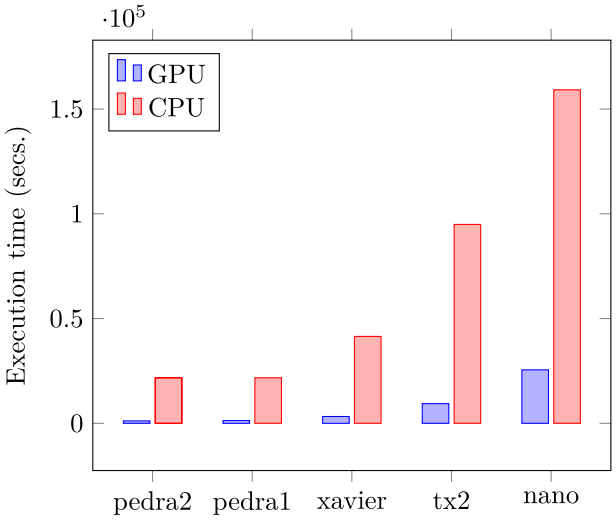


Fig. 8. Execution time GPU/CPU for the autoencoder's training stage.

B. Performance Evaluation

This section evaluates the performance (i.e., execution time) of the AI-pipeline previously presented in Section III. We focused on four different architectures (see Table I): an HPC node called Pedra, and three low-power edge computing devices from the NVIDIA Jetson family (i.e., Jetson nano, Jetson TX2, and Jetson AGX Xavier). The main purpose of this comparison is to determine whether the AI-pipeline proposed adapted for GPUs reduces the calculation time, and whether it can be performed in the edge in a reasonable time frame and, if so, which platform is the most suitable.

First, the training stage of the autoenconder is evaluated in Fig. 8. It shows the execution time for CPU and GPU versions in all targeted architectures. We ran 30 epochs, which is the actual number of epochs needed for convergence. Several conclusions can be drawn from these numbers. First, the use of GPUs increases the performance in all platforms, including edge computing platforms. The performance difference between CPU and GPU in the server version is up to 16.5× speed-up factor by using a single GPU. This difference increases by a factor of 2.33× when two GPUs are targeted on the same server (i.e., Pedra). GPU performance numbers are also interesting on the edge computing side. The use of low-power GPUs in edge devices also increases application performance substantially. The Xavier GPU delivers up to 12.7× speed-up factor compared to the sequential code, executed on the Xavier ARM-based CPU. In the same way, the TX2 GPU delivers up to 10.11× speed-up factor compared to its sequential counterpart version. The difference decreases when using the Jetson Nano GPU, with a speed-up factor of up to 4.5× when using the GPU instead of the CPU. In fact, this GPU is a very low-power, low-cost solution that cannot offer a performance as high as its counterparts, but its use is definitely a good contribution in terms of efficiency.

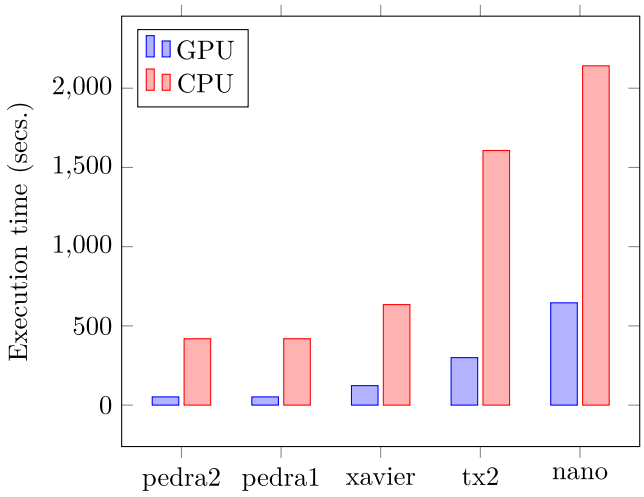


Fig. 9. Inference comparison of GPU/CPU for the entire data set.

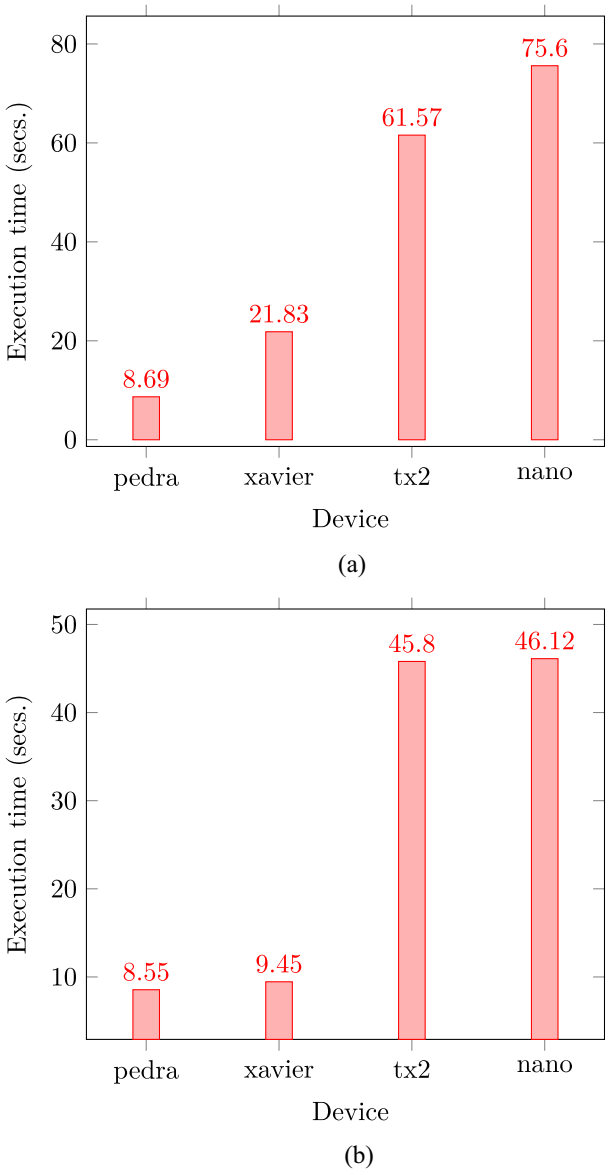


Fig. 10. Execution time of the last two steps of the AI-pipeline. (a) *t*-SNE algorithm. (b) FM clustering algorithm.

Another relevant point is the performance difference between edge and cloud approaches. The cloud infrastructure, Pedra, defeats edge devices by a wide margin, as expected. Pedra, using two GPUs, achieves a speed-up factor of up to 2.8× compared to AGX Xavier, 8.15× compared to Jetson TX2, and 22.17× compared to Jetson Nano. It is important to note that these figures refer to training, which is not well suited to be executed at the edge. For the inference stage, the performance differences between the computing platforms are similar (see Fig. 9), but the overall execution time is much shorter.

Finally, the last two steps of the proposed AI-pipeline are shown in Fig. 10. Execution times of these two steps are very low compared to the training and inference of the neural network. Therefore, these processes have been executed on CPU as their computation is hidden by the first step of the pipeline. In particular, the cross-platform differences for this algorithm are similar to those discussed above, with the pedra HPC server showing the best performance, followed by Xavier, TX2 and Jetson Nano. While it is true that the differences between Pedra and Xavier are 2.5× in speedup factor, and up to 7× between Pedra and TX2, in general execution times in the edge are reduced, with both algorithms taking less than a minute to run.

VI. Conclusion and Future Work

Autonomous UAVs could play a "key role" in addressing the consequences of climate change. However, hardware and software developments are needed for these drones to really be determinant actors in these tasks. The intersection between AI and edge computing is undoubtedly the answer today, allowing to transform autonomous drones into useful tools under different emergency situations. This article has proposed an AI-based pipeline to be run on edge computing platforms in order to enable efficient processing of drone images of natural disasters.

Our results reveal that the use of GPUs in edge computing platforms increases performance by up to a 12.7× speed-up factor, providing computational horsepower that enables the full execution of the proposed AI-pipeline. The computational differences between edge and cloud platforms are still large; in the range of 2.8×–22.17× speed-up factor, but the development of efficient platforms for the execution of specific workloads, such as those within DL, shows a roadmap that enables the development of applications for relevant autonomous and intelligent systems such as the one proposed here.

We definitely believe that smart autonomous drone technology can be a milestone in the fight against climate change. However, there is still a lot of work to be done from different perspectives. In terms of communication, extending the results of this article to a swarm of drones can provide a greater coverage of the area to be inspected, which is much needed in this type of natural catastrophes. The inclusion of highly energy-efficient processors in such low autonomy devices is a must in order to enable AI-based applications; tinyML is actually

a good step forward in this direction. More processing steps could be added in this AI-pipeline which, after manual labeling or pseudolabeling of the clusters, the information stored in the autoencoder will be reused to categorize all the images and send the desired information in a more granular way. Finally, real-case scenarios will be approached with first-responders to figure out new features and requirements.

REFERENCES

[1] R. Dellink, E. Lanzi, and J. Chateau, "The sectoral and regional economic consequences of climate change to 2060," *Environ. Resour. Econ.*, vol. 72, no. 2, pp. 309–363, 2019.

[2] S. K. Sood and K. S. Rawat, "A scientometric analysis of ICT-assisted disaster management," *Nat. Hazards*, vol. 106, pp. 2863–2881, Jan. 2021.

[3] H. Shakhatreh *et al.*, "Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48572–48634, 2019.

[4] M. A. R. Estrada and A. Ndoma, "The uses of unmanned aerial vehicles—UAV's- (or drones) in social logistic: Natural disasters response and humanitarian relief aid," *Procedia Comput. Sci.*, vol. 149, pp. 375–383, Mar. 2019.

[5] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for deep learning," 2019. [Online]. Available: arXiv:1907.10701.

[6] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[7] H. Halawa, H. A. Abdelhafez, A. Boktor, and M. Ripeanu, "NVIDIA Jetson platform characterization," in *Proc. Eur. Conf. Parallel Process.*, 2017, pp. 92–105.

[8] P. J. Hardin and R. R. Jensen, "Small-scale unmanned aerial vehicles in environmental remote sensing: Challenges and opportunities," *GISci. Remote Sens.*, vol. 48, no. 1, pp. 99–111, 2011.

[9] P. A. Rodriguez *et al.*, "An emergency response UAV surveillance system," in *Proc. AMIA Annu. Symp.*, vol. 2006, 2006, p. 1078.

[10] Q. Wen *et al.*, "UAV remote sensing hazard assessment in Zhouqu debris flow disaster," in *Proc. Remote Sens. Ocean Sea Ice Coastal Waters Large Water Regions*, vol. 8175, 2011, Art. no. 817510.

[11] S. Mehrdad, M. Satari, M. Safdary, and P. Moallem, "Toward real time UAVs' image mosaicking," in *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 41, ISPRS, Paris, France, 2016, pp. 941–946.

[12] I. Bekmezci, O. K. Sahingoz, and Ş. Temel, "Flying ad-hoc networks (FANETs): A survey," *Ad Hoc Netw.*, vol. 11, no. 3, pp. 1254–1270, 2013.

[13] J. Lin and P. Singer, "China is making 1,000-UAV drone swarms now," in *Popular Science*, vol. 8. New York, NY, USA: Bonnier, 2018.

[14] E. A. Marconato, J. A. Maxa, D. F. Pigatto, A. S. R. Pinto, N. Larrieu, and K. R. L. J. C. Branco, "IEEE 802.11n vs. IEEE 802.15.4: A study on communication QoS to provide safe FANETs," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. Workshop (DSN-W)*, Toulouse, France, 2016, pp. 184–191.

[15] X. Zhang, H. Wu, M. Wu, and C. Wu, "Extended motion diffusion-based change detection for airport ground surveillance," *IEEE Trans. Image Process.*, vol. 29, pp. 5677–5686, 2020.

[16] A. Singh and K. K. Singh, "Satellite image classification using genetic algorithm trained radial basis function neural network, application to the detection of flooded areas," *J. Vis. Commun. Image Represent.*, vol. 42, pp. 173–182, Jan. 2017.

[17] R. Nijhawan, M. Rishi, A. Tiwari, and R. Dua, "A novel deep learning framework approach for natural calamities detection," in *Information and Communication Technology for Competitive Strategies*. Singapore: Springer, 2019, pp. 561–569.

[18] A. Gebrehiwot, L. Hashemi-Beni, G. Thompson, P. Kordjamshidi, and T. E. Langan, "Deep convolutional neural network for flood extent mapping using unmanned aerial vehicles data," *Sensors*, vol. 19, no. 7, p. 1486, 2019.

[19] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.

[20] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[21] X. Chen, C. Wu, Z. Liu, N. Zhang, and Y. Ji, "Computation offloading in beyond 5G networks: A distributed learning framework and applications," *IEEE Wireless Commun.*, vol. 28, no. 2, pp. 56–62, Apr. 2021.

[22] B. Blanco-Filgueira, D. García-Lesta, M. Fernández-Sanjurjo, V. M. Brea, and P. López, "Deep learning-based multiple object visual tracking on embedded system for IoT and mobile edge computing applications," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5423–5431, Jun. 2019.

[23] M. H. Ionica and D. Gregg, "The movidius myriad architecture's potential for scientific computing," *IEEE Micro*, vol. 35, no. 1, pp. 6–14, Jan./Feb. 2015.

[24] S. Cass, "Taking AI to the edge: Google's TPU now comes in a maker-friendly package," *IEEE Spectr.*, vol. 56, no. 5, pp. 16–17, May 2019.

[25] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *Proc. 11th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2013, pp. 139–152.

[26] L. J. P. Van der Maaten and G. E. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.

[27] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Jan. 2011.

[28] A. Nagpal, A. Jatain, and D. Gaur, "Review based on data clustering algorithms," in *Proc. IEEE Conf. Inf. Commun. Technol.*, Thuckalay, India, 2013, pp. 298–303.

[29] Z. Cui, X. Jing, P. Zhao, W. Zhang, and J. Chen, "A new subspace clustering strategy for AI-based data analysis in IoT system," *IEEE Internet Things J.*, early access, Feb. 2, 2021, doi: 10.1109/JIOT.2021.3056578.

[30] J. M. Cecilia, I. Timón, J. Soto, J. Santa, F. Pereñíguez, and A. Muñoz, "High-throughput infrastructure for advanced ITS services: A case study on air pollution monitoring," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 7, pp. 2246–2257, Jul. 2018.

[31] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The fuzzy *c*-means clustering algorithm," *Comput. Geosci.*, vol. 10, nos. 2–3, pp. 191–203, 1984.

[32] I. Timón, J. Soto, H. Pérez-Sánchez, and J. M. Cecilia, "Parallel implementation of fuzzy minimals clustering algorithm," *Expert Syst. Appl.*, vol. 48, pp. 35–41, Apr. 2016.

[33] J. M. Cebrian, B. Imbernón, J. Soto, J. M. García, and J. M. Cecilia, "High-throughput fuzzy clustering on heterogeneous architectures," *Future Gener. Comput. Syst.*, vol. 106, pp. 401–411, May 2020.

[34] C. Kyrkou, *AIDER (Aerial Image Dataset for Emergency Response Applications)*, Zenodo, Jun. 2020, doi: 10.5281/zenodo.3888300.

[35] C. Kyrkou and T. Theocharides, "Deep-learning-based aerial image classification for emergency response applications using unmanned aerial vehicles," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Long Beach, CA, USA, 2019, pp. 517–525.

[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.

[37] X. Xia, C. Xu, and B. Nan, "Inception-v3 for flower classification," in *Proc. 2nd Int. Conf. Image Vis. Comput. (ICIVC)*, Chengdu, China, 2017, pp. 783–787.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.

Daniel Hernández received the B.S. degree in computer science and the master's degree in massive data analysis technologies from the Catholic University of Murcia, Murcia, Spain, in 2018 and 2020, respectively. He is currently pursuing the Ph.D. degree with the Universitat Politècnica de València, Valencia, Spain.

His main research interests include artificial vision, deep learning, and edge computing.

Juan-Carlos Cano received the M.Sc. and Ph.D. degrees in computer science from the Universitat Politècnica de València (UPV), Valencia, Spain, in 1994 and 2002, respectively.

He is a Full Professor with the Department of Computer Engineering, UPV. His current research interests include vehicular networks, mobile *ad hoc* networks, and pervasive computing.

Federico Silla received the M.S. and Ph.D. degrees in computer engineering from the Technical University of Valencia, Valencia, Spain, in 1995 and 1999, respectively.

He is a Full Professor with the Department of Computer Engineering, Universitat Politècnica de València, Valencia. His current research interests include GPU virtualization techniques and interconnection networks.

Carlos T. Calafate graduated (Hons.) degree from the University of Porto, Porto, Portugal, in 2001, and the Ph.D. degree in informatics from the Universitat Politècnica de València (UPV), Valencia, Spain, in 2006.

He is a Full Professor with the Department of Computer Engineering, UPV. His research interests include *ad hoc* and vehicular networks, UAVs, smart cities & IoT, QoS, network protocols, video streaming, and network security.

José M. Cecilia received the B.S. degree in computer science from the University of Murcia, Murcia, Spain, in 2005, the M.S. degree in computer science from the University of Cranfield, Cranfield, U.K., in 2007, and the Ph.D. degree in computer science from the University of Murcia in 2011.

He is a Ramón y Cajal Research Fellow (Associate Professor Tenure track) with the Computer Engineering Department, Universitat Politècnica de València, Valencia, Spain. His research interest includes HPC, IoT, AI, and social sensing.