Neural Computing and Applications (2022) 34:15349–15368
https://doi.org/10.1007/s00521-022-07467-z

REVIEW

An automatic fire detection system based on deep convolutional neural networks for low-power, resource-constrained devices

Pedro Vinícius A. B. de Venâncio[1,2] · Adriano C. Lisboa[1,3] · Adriano V. Barbosa[2,4]

Received: 3 December 2021 / Accepted: 24 May 2022 / Published online: 30 June 2022

Abstract
Large-scale fires have been increasingly reported in the news media. These events can cause a variety of irreversible damage, what encourages the search for effective solutions to prevent and fight fires. A promising solution is an automatic system based on computer vision capable of detecting fire in early stages, enabling rapid suppression to mitigate damage, minimizing combat and restoration costs. Currently, the most effective systems are typically based on convolutional neural networks (CNNs). However, these networks are computationally expensive and consume a large amount of memory, usually requiring graphics processing units to operate properly in emergency situations. Thus, we propose a CNN-based fire detector system suitable for low-power, resource-constrained devices. Our approach consists of training a deep detection network and then removing its less important convolutional filters in order to reduce its computational cost while trying to preserve its original performance. Through an investigation of different pruning techniques, our results show that we can reduce the computational cost by up to 83.60% and the memory consumption by up to 83.86% without degrading the system's performance. A case study was performed on a Raspberry Pi 4 where the results demonstrate the viability of implementing our proposed system on a low-end device.

Keywords Deep convolutional neural network · Computational efficiency · Filter pruning · Large-scale fire detection

1 Introduction

The frequency of wildfires caused by human intervention has been growing alarmingly [1]. These fire events cause a variety of social, environmental and economic damage and are not restricted to areas of incidence, since fire is capable of spreading quickly to neighboring regions. Occasionally, fire proportions are so extreme as to make suppression and control practically impossible. Given these circumstances, it is essential to implement strategies to identify fires at early stages, so that it is possible to mitigate any damages or even avoid them. In large territorial areas, such as forests and plantations, however, it is not feasible to assign such tasks to humans. Besides demanding constant attention during surveillance, a complete analysis of the environment in search of potential fire occurrences can require abundant, costly labor with no guarantee of effectiveness.

Among the systems developed for fire detection, the most conventional ones are based on thermal, photoelectric or ionic sensors. However, their first successful applications were restricted to closed environments, as these devices have limited range and still require a certain

Adriano C. Lisboa and Adriano V. Barbosa contributed equally to this work.

Pedro Vinícius A. B. de Venâncio
pedrovinicius@ufmg.br

Adriano C. Lisboa
adriano.lisboa@gaiasd.com

Adriano V. Barbosa
adrianovilela@ufmg.br

1 Gaia Solutions on Demand, Technological Park, Belo Horizonte, Brazil

2 Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

3 Department of Computer Engineering, Federal Center for Technological Education of Minas Gerais, Belo Horizonte, Brazil

4 Department of Electronics Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

proximity to the fire source [2]. Smoke sensors, for example, have a limited response time, as carbon particles can take a long time to reach the detector, causing a phenomenon known as transport delay [3]. Thus, the scalability for open and extensive areas was only made possible through the paradigm of wireless sensor networks (WSNs), whose principle is to deploy a set of sensor nodes in the region of interest to monitor and report possible fire occurrences to a base station [4]. Although methods based on multiple sensors show promising results, using sensors for open areas requires considerable investment. Generally, forest regions comprise biomes whose vegetation cover is dense, with contiguous large trees. Ensuring a proper positioning of the sensors under these conditions becomes unfeasible, since access during the installation and maintenance of equipment is restricted. In addition, the extension of the monitoring area directly affects the cost of the project and further weakens the management of available resources for the WSN operation.

According to a continuous technological evolution in digital cameras and computers [5], as well as the need to corroborate fire detection through image visualization, several researchers have been interested in automating fire monitoring through computer vision. Running algorithms on local processing devices integrated with surveillance cameras to identify fires can be very interesting, as it may allow fast and assertive detections that are independent of a decision maker. The question then is: how to identify a wildfire? From its main visual indicators: fire and smoke. Smoke makes it possible to identify wildfires in bright environments, even over long distances, and fire makes it possible to identify wildfires in low-light environments.

The interest in computer vision techniques has led to the emergence of many fire detection systems based on such techniques. They can be divided into two strands: top-down approaches and bottom-up approaches. The top-down approaches basically consist of defining deterministic rules for identifying fire and smoke, i.e., it tries to model the recurrent visual patterns of these two classes. The most common methods in this approach use color segmentation in different color spaces, such as RGB [6], HSI [7, 8], YUV [9–11], YCbCr [12] and others, as well as motion analysis, with optical flow [8, 13], background subtraction [11, 13] and spatiotemporal persistence [6, 14, 15]. However, many limitations have been observed over the years. It is difficult, for example, for these approaches to generalize well to new scenarios. Well-defined rules used to identify fire in a scene with a certain lighting usually do not work well when the system is transferred to a different environment. In turn, motion analysis approaches depend heavily on camera stabilization. Any camera movement can hinder detection. Top-down approaches also suffer from high false alarm rates. Several studies report that fire and smoke have visual features which are very similar to those of other ordinary objects in surveillance scenes [14, 16]. For example, similarities can be found between saturated fires and car headlights or electric lamp lights, and also between smoke and mist or clouds.

Although top-down approaches have substantially increased the fire detection rate compared to sensor-based strategies, applications meant to notify a fire brigade or to trigger any preventive measures are considered viable only if they have a low false alarm rate. A high number of false positives would imply unnecessary mobilizations to contain non-existent fires and a gradual distrust of the system by its users. Bottom-up approaches emerged in this scenario. They consist of learning the visual patterns of fire and smoke from images and, later, using this knowledge to identify new fire occurrences. With the advancements in processing capacity and the massive production of data (big data era), artificial neural networks (ANNs) gained visibility again. A variety of architectures have been explored, such as multilayer perceptrons (MLPs) [17] and radial basis function (RBF) networks [18].

Recently, a growing trend toward deep architectures has been observed, as these networks are capable of performing more complex learning [19]. Among these architectures, convolutional neural networks [19, 20] are the ones that have stood out more in visual pattern recognition tasks. Unlike their classical counterparts, these networks, whose architectures are inspired on the human visual perception system, enable image processing through sparse connectivity and parameter sharing, as well as automatically extracting highly relevant visual features from the raw pixels of the image. Thus, the idea of using visual pattern recognition techniques as implemented by CNNs to monitor environments has seen consistent growth. In automatic fire detection applications, rule-based approaches have been gradually replaced by convolutional networks trained on surveillance camera images. These convolutional architectures have shown promising results and a good generalization ability which can lead them to be employed in new scenarios [21–24].

Despite their remarkable results, CNNs present some challenges. One of them is the trade-off between performance and inference speed that they assume [25]. As convolutional network architectures become more complex (deeper), they generally perform better in object recognition tasks [26]. On the other hand, convolutional networks, especially the deep ones, consume a lot of memory and entail a high computational cost, both in training and in prediction, which makes their execution on resource-constrained devices, such as smartphones, Internet of Things (IoT) devices and single board computers, practically unfeasible for emergency situations. Consequently, a monitoring scenario with multiple cameras integrated with local processing devices is no longer feasible without the support of graphics processing units (GPUs) or cloud processing services, both being quite expensive alternatives for large-scale surveillance.

To deal with this problem, convolutional network architectures were precisely designed for efficient fire recognition [23, 27–30]. However, designing a CNN that fits a resource-constrained device and that has high predictive performance requires significant human engineering due to its trial-and-error essence [31]. In order to obtain the best trade-off between computational cost and predictive performance, the specialist responsible for designing the network architecture should investigate a variety of possible combinations of the hyperparameters related to the network structure, such as input image size, types of activation functions, different stacks of layers, number of convolutional layers and each of which with their respective hyperparameters as well (e.g., number of filters, padding, stride, kernel size and dilation). Furthermore, the design specialist is limited to a certain number of convolutional layers (and possibly to a certain level of generalization [26]), as these are the most computationally expensive layers in a classical convolutional network [32].

Hence, several studies have been conducted with the purpose of optimizing the architectures and implementations of deep networks already proposed in the literature, which avoids the process of human engineering described above. The proposed solutions include binarization of weights and activations [33, 34], depth-wise separable convolution [35, 36], knowledge distillation [37, 38] and pruning approaches [39–42]. Pruning approaches comprise the benefits of all the other mentioned methods, such as memory reduction and decrease in the number of parameters, and are, therefore, the most explored by the scientific community nowadays. Among the existing pruning approaches for CNNs, we can mention filter pruning. Filters are tensors of learnable parameters that, when applied to an image, generate a new image (feature map) through the convolution operation. The resulting images make it possible to highlight different visual information contained in the original image. It turns out that during training, the learned filters may present a significant redundancy with other filters in the same layer or may even contribute little to the output of the network. Filter pruning removes the less relevant filters, reducing the computational cost during prediction, without decreasing the architectural depth and without significantly degrading the network's performance.

Thus, by significantly reducing the computational cost bottleneck of systems based on deep convolutional networks, filter pruning may enable the monitoring of large green areas with multiple cameras integrated to low-cost fire detection systems. Therefore, this work proposes an autonomous and efficient fire detection tool targeted at mobile devices and other low-power, resource-constrained devices. In order to achieve that, we investigate the effects of different filter pruning techniques applied to a deep convolutional network trained on a large dataset to detect fire and smoke. This task is hampered by the fact that the literature on parameter compression in convolutional networks focuses mostly on the task of image classification, where the objective is to assign a single object label to an image, without having to locate the object in the image. In turn, similar analyses for the object detection task [43, 44], especially in the particular case of fire detection, are mostly missing in the literature. Furthermore, a case study is conducted where an optimized network is deployed to a Raspberry Pi 4 Model B (single board computer). The results demonstrate that it is possible to obtain an optimized network that not only has better predictive performance than the original network but is also associated with smaller energy consumption, initialization and detection times.

The remainder of this paper is organized as follows. Section 2 presents theoretical concepts and tools necessary for the development of the proposed detection system. Section 3 presents the dataset used to train the networks, the experimental setup, the network training process (including the definition of hyperparameters) and an investigation of the techniques used to remove convolutional filters from the trained network, as well as modifications made to these techniques in order to adapt them to the fire and smoke detection problem. Section 4 presents and discusses the experimental results. Finally, conclusions and directions for future work are presented in Sect. 5.

2 Background

This section presents an overview of the main components used in the development of the proposed fire detection system. First, we define the architecture of the convolutional network used to perform the detection task. Next, we describe the general procedure for network compression based on the pruning of convolutional filters (specific implementations are discussed in Sect. 3.4). Finally, we present the evaluation metrics used to verify the network's predictive performance.

2.1 The YOLO (You Only Look Once) algorithm

Given the promising results achieved by CNNs in image classification [20], a notable trend has been the integration of these architectures into object detectors [45–49]. This gave rise to two categories of detectors: (i) two-stage detectors, which use a region proposal algorithm to identify regions in the image that are likely to contain objects of interest and then use the network to classify the objects in each of these regions; (ii) single-stage detectors (single shots), which perform both classification and localization of objects in a single forward propagation of the network. From predictions in a predefined set of bounding boxes (anchor boxes) [45, 47], it is unnecessary to use region proposal algorithms, which are typically a computational bottleneck in two-stage detectors. In the context of large-scale surveillance of emergency situations, there is a need for faster detectors that require less computational resources, which suggests a single-stage detector in this case.

One of the most used single-stage detectors is the You Only Look Once (YOLO) algorithm [46]. In a simplified way, YOLO, in its first version, resizes the input image to $N \times N$ pixels and then splits it into a grid of $S \times S$ cells. Each cell, responsible for detecting a single object, predicts B bounding boxes with their respective confidence scores and C class probabilities. In the fire detection problem, $C = 2$ (fire and smoke). If the center of an object falls within a specific cell, that cell is responsible for detecting the object. The object's bounding box is defined as the box with the highest confidence score above a predefined confidence threshold t_{conf}. In turn, the object's class is defined by the highest class probability. However, if the total number of cells is large, the bounding boxes in different cells will be very close to each other and can be assigned to the same object. To tackle this problem, YOLO uses the non-maximum suppression (NMS) algorithm [50] to post-process network predictions. This technique selects the bounding box with the highest detection probability and eliminates the boxes that most overlap it. After that, the process repeats for the box with the second highest detection probability and so on until all remaining boxes are analyzed.

Since the first version of YOLO, three more versions have been proposed, in addition to some extensions of these versions [47–49]. The fourth version (YOLOv4), which is used in this work for fire and smoke detection, introduces many improvements to the network architecture, as well as using bag of freebies and bag of specials methods, which improve the training process and predictive performance, respectively [49]. The YOLOv4 architecture can be decomposed into three main structures. The first one is the CSPDarknet53 [51] feature extraction network pre-trained on the ImageNet dataset [52]. On top of this backbone, the Spatial Pyramid Pooling (SPP) [53] module has been added, since it significantly increases the receptive field at the cost of almost no reduction in network inference speed. The Path Aggregation Network (PAN) [54] was also added, in order to aggregate information from different backbone levels to different detector levels. SPP and PAN are the neck structure. The last structure is the YOLOv3 [48] head, which predicts bounding boxes using dimension clusters as anchor boxes.

2.2 Pruning of convolutional filters

The idea to prune convolutional filters to optimize a CNN architecture comes from the fact that about 99% of overall floating point operations in a forward propagation pass are concentrated on the convolutional layers [32]. Basically, the pruning procedure consists of sorting the filters in each convolutional layer of the network according to some criterion and then removing the least important filters. The amount of removed filters is usually expressed as a percentage of the number of original filters in the layer and is referred to as the pruning rate ($p\%$). Given a convolutional layer l of a trained network, the filter pruning procedure for the layer can be summarized according to the following steps [39]:

1. Calculate the importance s_j of each of the $n^{[l+1]}$ filters $\mathcal{F}_j^{[l]} \in \mathbb{R}^{n^{[l]} \times k^{[l]} \times k^{[l]}}$ in the set $\mathcal{F}^{[l]} \in \mathbb{R}^{n^{[l+1]} \times n^{[l]} \times k^{[l]} \times k^{[l]}}$ of layer l according to some predefined criterion. In Sect. 3.4, several pruning techniques, each using a different criterion to define the filters to be removed, are investigated in an attempt to obtain a good trade-off between performance and computational cost during prediction.
2. Sort convolutional filters according to their importance s_j.
3. Prune $p\%$ of filters with the smallest importance. The depth of the set of filters $\mathcal{F}^{[l+1]}$ in layer $l+1$ is also reduced in this process. This dimension relationship between filters of adjacent layers is shown in Fig. 1.

After pruning $p\%$ of filters in all prunable layers of the network, i.e., layers that can have their filters removed without disrupting the forward propagation of the network, the remaining filters can present an undesirable behavior, since their contribution to the output can be highly correlated with some filter that was removed. Therefore, it is advisable to readjust the remaining parameters through fine-tuning. The network's ability to recover its accuracy after this process is called plasticity [43]. The steps mentioned above make up a single iteration of a pruning method. If more iterations are necessary or desired, the network at the end of iteration i is passed as input to iteration $i + 1$. Thus, pruning can be performed either iteratively until reaching $p\%$ of pruning per layer (iterative pruning) or directly removing $p\%$ of filters per layer in a single iteration (single pruning) [41]. The filter pruning techniques used in this work are discussed in detail in Sect. 3.4.

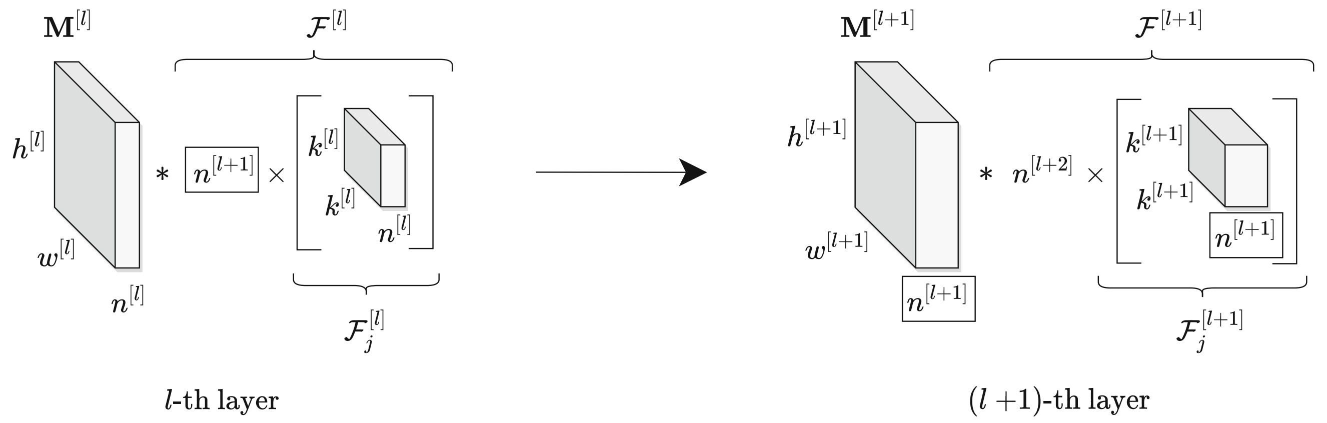


Fig. 1 Each filter of layer $l+1$ has height and width $k^{[l+1]}$ and depth (number of channels) $n^{[l+1]}$, i.e., equal to the number of filters applied on the previous layer l. Hence, pruning a filter $\mathcal{F}_j^{[l]}$ among the $n^{[l+1]}$ filters of layer l directly affects the depth of the set of feature maps $\mathbf{M}^{[l+1]}$ and the depth of the set of filters $\mathcal{F}^{[l+1]}$ of layer $l+1$

2.3 Evaluation metrics

In order to evaluate the performance of the fire detection network, as well as tune some of its hyperparameters, the metrics shown below, relevant in object detection algorithms, are used [55]. These metrics are necessary to compare the pruned networks to each other and also to the original, unpruned network.

1. Intersection over Union (IoU): it evaluates how good the location of a predicted object is, i.e., how close the predicted bounding box is to the ground-truth bounding box, which was manually labeled by experts. This similarity between the predicted and the ground-truth bounding boxes is computed as

$$\mathrm{IoU} = \frac{\text{area of overlap}}{\text{area of union}} \;, \tag{1}$$

where the numerator denotes the intersection of the predicted and ground-truth bounding boxes and the denominator denotes their union. First, an IoU metric is calculated for each individual class based on all correct detections (i.e., detections whose labels were correctly classified). These IoU values are then averaged over all classes to provide a global score called avg IoU.
2. Precision: is the proportion of objects detected correctly among all objects detected. It is given by

$$\text{precision} = \frac{\mathrm{TP}}{\mathrm{TP{+}FP}} \;, \tag{2}$$

where TP is the number of objects detected correctly (true positives) and FP is the number of objects detected incorrectly (false positives). A detection is considered correct (a true positive) when the predicted and ground truth labels match and the IoU between them is greater than or equal to a pre-determined threshold t_{IoU}. If one of these conditions is not met, i.e., the predicted and ground truth labels do not match or the IoU is below the threshold t_{IoU}, then there is a false positive.
3. Recall: is the proportion of objects detected correctly among all objects of interest (in our case, every smoke and fire event). It is given by

$$\text{recall} = \frac{\mathrm{TP}}{\mathrm{TP{+}FN}} \;, \tag{3}$$

where FN means the number of objects of interest that have not been detected.
4. F_1-score: aggregates precision and recall into a single interpretable value through the harmonic mean of these metrics. It is calculated as

$$F_1\text{-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision + recall}} \;. \tag{4}$$

5. Average Precision (AP): since precision and recall are strongly related to the confidence threshold t_{conf}, the precision–recall curve can be obtained to represent the trade-off between these two metrics as a function of t_{conf}. The area under the precision–recall curve of a single class is known as Average Precision (AP) and the mean of this metric over all the classes is called mean Average Precision (mAP) [56]. In this work, a threshold value $t_{\mathrm{IoU}} = 0.50$ is used to compute AP, which is then referred to as AP@0.50.

3 Methods

This section starts by presenting the dataset used to train the detection networks. This dataset contains thousands of labeled images of fire and smoke events and was specifically constructed for this project. Next, the experimental

Table 1 Number of images per category in the D-Fire dataset

Category	Description	Images
Fire	Images containing only fire events.	1164
Smoke	Images containing only smoke events.	5867
Fire and smoke	Images containing both fire and smoke events.	4658
None	Negative examples, i.e., images containing neither fire nor smoke events, but rather objects or environments that can be easily mistaken for fire or smoke by a human.	9838

setup used in this work is presented. We then describe the procedure used to train the networks on our dataset and how the network hyperparameters are defined. Finally, we present many filter pruning techniques used to optimize the network's architecture. This optimization process is meant to reduce the computational cost associated with the network so that it can run on low computing power devices while preserving its original detection performance.

3.1 The D-fire dataset

D-Fire is an image dataset for fire and smoke detection [57]. It contains a total of 21,527 images categorized according to Table 1. Although the number of images in the category "fire" (i.e., images containing only fire) is significantly smaller than the other categories, the number of bounding boxes labeled as fire in the images containing this class (categories "fire" and "fire and smoke") is usually high, with an average of 2.52 bounding boxes per image. On the other hand, the average number of boxes labeled as smoke in the "smoke" and "fire and smoke" categories is 1.13 bounding boxes per image. In total, there are 26,557 bounding boxes, where 11,865 bounding boxes are labeled as smoke and 14,692 as fire.

The classes considered in this work are quite abstract, in the sense that they can take quite different forms. Smoke, for example, may be more concentrated or more diffused, as it is easily influenced by the direction and intensity of the wind. Furthermore, its color and intensity may vary depending on the substances involved in the combustion process. Given these premises and the inherent difficulties of detection, images for the D-Fire dataset were collected from several sources, such as from the Internet, from legal fire simulations in the Technological Park of Belo Horizonte, Brazil, and from surveillance cameras monitoring landscapes at the Universidade Federal de Minas Gerais (UFMG) and at the Serra Verde State Park, in Belo Horizonte. In addition, some images were generated through montages in a photo editing software, where some artificial smoke spots were randomly selected and distributed in backgrounds with green landscapes typical of an environmental monitoring. This data augmentation was used in a previous work and showed promising results with networks trained on synthetic smoke images [58]. Thus, the dataset became quite diverse in terms of fire and smoke patterns, fire scenarios, camera positions and ambient lighting intensity. Figure 2 shows some examples of images present in the D-Fire dataset and their respective ground-truth bounding boxes.

3.2 Experimental setup

All experiments, except for the case study, were run on a machine with 256 GB of RAM and using a single GPU NVIDIA Quadro P5000 with 16 GB of GDDR5X. The system ran Ubuntu 18.04.3 LTS and CUDA compilation tools, release 10.1, V10.1.243. The training and fine-tuning procedures were implemented in the C language using the Darknet framework [56], whereas the filter pruning techniques were implemented in the Python language using PyTorch [60] and scikit-learn [61].[1] The D-Fire dataset was randomly divided into two independent and identically distributed sets: (i) training set, containing 80% of the images (17,221 images), and (ii) testing set, containing the remaining 20% of the images (4,306 images). Data augmentation techniques (e.g., saturation, exposure, hue, blur and mosaic [49]) were randomly applied to the training set.

3.3 Training procedure

The first step taken to develop the proposed fire detection system was the training of the YOLOv4 network. The training was performed using the Mini-batch Gradient Descent algorithm [63] over 30,000 iterations and with a momentum [64] 0.9 and a mini-batch size of 64. Each mini-batch was further subdivided into 16 partitions, so that only $64/16 = 4$ images were simultaneously processed by the GPU at a time. In order to avoid overfitting, we used early stopping, which could stop training after 5,000 iterations with no improvement, and also ℓ_2-regularization [65] with weight decay 0.0005, which employs a penalty

[1] All source code is available at [62].

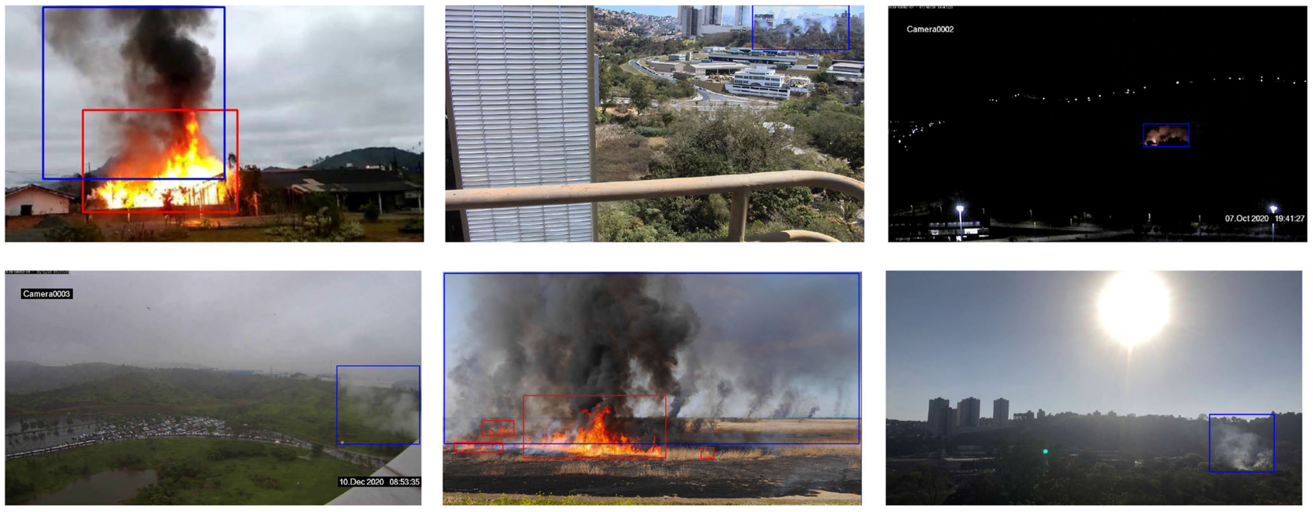


Fig. 2 Some positive examples of D-Fire dataset. The images were labeled by researchers and the annotations were peer-reviewed by experts [59]

on learned parameters with very large magnitudes. Learning rate warm-up technique [47] with exponent 4 starting at 1000 iterations and multistep scheduler [19] with reduction factor 0.1 at marks 24,000 and 27,000 iterations were used to improve the training. Image size was $N = 416$, grid cell size $S = N/32 = 13$ and $B = 3$ bounding boxes per grid cell [46].

The YOLOv4 network, when designed to detect two classes ($C = 2$),[2] takes 59.57 BFLOPs in the forward propagation pass when processing a single RGB image of size 416×416 pixels. As this high computational cost requires GPU for training and real-time detection, Bochkovskiy et al. [49] also proposed Tiny YOLOv4, a lighter and faster version, which takes only 6.789 BFLOPs for forward propagating the same image, what represents a computational cost reduction of about 88.86% when compared to the original YOLOv4. However, this speed comes at the cost of a reduced predictive performance. Therefore, the Tiny YOLOv4 network was used to further establish comparisons with the networks resulting from more intensive pruning (higher pruning rates). The Tiny YOLOv4 network was trained with the same hyperparameters mentioned above.

Learning rate values of 0.001 and 0.005 were used, respectively, for the Tiny YOLOv4 and the YOLOv4 networks, which were the values that maximized the mAP. In turn, confidence threshold values of 0.3 and 0.4 were used, respectively, for the Tiny YOLOv4 and the YOLOv4 networks, which were the values that maximized the F_1-score, since the mAP is invariant to the confidence threshold. In addition, the weights of the layers of both networks were initialized through the transfer learning in the ImageNet dataset [52].

3.4 Filter pruning techniques

Complex architectures, i.e., those with a high number of layers and a high number of convolutional filters, come at the cost of a computationally expensive detection, as well as a lower detection speed. While YOLOv4 processes 47 FPS, Tiny YOLOv4 processes 342 FPS under the experimental setup used in this work. Although compatible with the real-time processing premise ($\geq$ 30 FPS) [66], this is only possible using powerful processors such as GPUs. Furthermore, 256.04 MB of memory is required to store the weights of the YOLOv4 architecture, whereas for Tiny YOLOv4 only 23.53 MB is required.

In the case of large-scale environmental monitoring, solutions based on deep learning like ours, although feasible, demand a high investment in infrastructure, whether in GPUs or in cloud computing and storage services. Therefore, addressing the issues of high computational cost and high memory consumption directly in the development of the network may be more interesting. Considering that YOLOv4 has a total of 33,215 convolutional filters and 63,943,071 parameters, and that its pipeline is composed of a single neural network with 110 convolutional layers, of which only the layers preceding the 3 detection layers can not be modified ($l_p = 107$ prunable layers), it turns out that it is possible to optimize its end-to-end architecture by removing the less important filters for detecting fire and smoke. This is the reason behind our investigation of filter pruning techniques in the trained YOLOv4 network.

[2] All discussions about computational cost in this work are based on YOLOv4 and Tiny YOLOv4 networks designed for fire and smoke detection ($C = 2$ classes). However, we note that the computational cost increases as C increases.

Filter pruning techniques differ mainly in how they quantify the importance of each convolutional filter for the final network prediction. The pruning techniques used in this work are organized into four categories: (i) techniques based on importance criteria; (ii) techniques based on multiple projections; (iii) techniques based on clustering and (iv) random pruning techniques. Except for techniques based on importance criteria and the random ones, the other pruning techniques presented here are modifications of existing techniques in the literature. The technique based on multiple projections was originally proposed for image classification task and needed to be adapted for fire detection task. The technique based on clustering, despite originally proposed for object detection tasks, does not actually prune filters, but instead combines them (feature interaction instead of feature selection), which can make the comparison to other techniques a little unfair. Thus, this technique was modified too.

3.4.1 Importance criteria

Among the filter pruning techniques available in the literature, the simplest ones are based on importance criteria, where each filter is individually evaluated according to some quantitative metric. Here, we investigate the most common ones, such as the ℓ_1-norm [39] and ℓ_2-norm [40]. The ℓ_1-norm of the j-th filter of layer l is given by

$$\left\| \mathcal{F}_j^{[l]} \right\|_1 = \sum_{d=1}^{n^{[l]}} \sum_{h=1}^{k^{[l]}} \sum_{w=1}^{k^{[l]}} \left| \mathcal{F}_{j\ dhw}^{[l]} \right| , \tag{5}$$

where $|\cdot|$ means absolute value and $k^{[l]}$, $k^{[l]}$ and $n^{[l]}$ are the filter's width, height and depth, respectively. In turn, the ℓ_2-norm of the j-th filter of layer l is given by

$$\left\| \mathcal{F}_j^{[l]} \right\|_2 = \sqrt{\sum_{d=1}^{n^{[l]}} \sum_{h=1}^{k^{[l]}} \sum_{w=1}^{k^{[l]}} \left(\mathcal{F}_{j\ dhw}^{[l]} \right)^2} . \tag{6}$$

In both cases (ℓ_1-norm and ℓ_2-norm), the $p\%$ of filters with the lowest norm are removed in each layer, as filters with small norm tend to produce weaker activation maps than filters in the same layer with higher norms [39].

3.4.2 Multiple projections

Importance criteria look at each convolutional filter individually, i.e., they typically consider neither the interactions between filters of the same layer nor the filter's effects on the network's output. A filter that by itself has little relevance to the network's performance can significantly improve accuracy when combined with other filters. In contrast, a filter that is particularly relevant can become redundant or even detrimental to the network's performance when used with other filters.

Techniques based on feature projections onto low-dimensional latent spaces model the relationship between dependent and independent variables. In the particular problem of fire detection, we are interested in capturing the relationship between the convolutional filter's outputs, i.e., the feature maps (independent variables), and the fire and smoke class labels (dependent variables) and, from this relationship, prune the $p\%$ less discriminative convolutional filters in each layer. In this work, we investigate only one technique in the multiple projections category, namely the PLS(Multi)+VIP pruning technique [41], which we have adapted to the fire and smoke detection task.

The multiple projections technique requires defining the dependent and independent variables to be used for projection. The independent variables are defined by propagating each image in the training set through the network in order to produce a feature map for each convolutional filter. Since feature maps have high dimensionality, the global max pooling operation [41] is employed to reduce the dimension of each feature map to a single value, so that only the feature with the highest value is extracted, that is, the most evident feature of the filter's output, as illustrated in Fig. 3. Thus, the convolutional layer l of a network with $n^{[l+1]}$ filters produces $n^{[l+1]}$ feature maps of size $h^{[l+1]} \times w^{[l+1]}$ which are reduced to a single feature vector of size $n^{[l+1]}$. Assuming a training set with m images, there will be m feature vectors of size $n^{[l+1]}$ and, consequently, a matrix

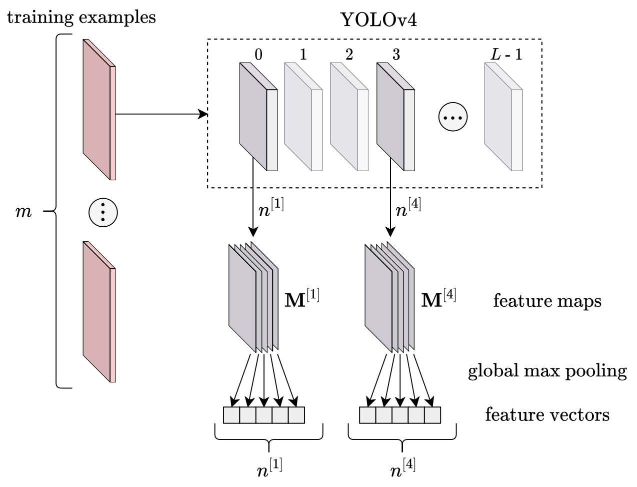


Fig. 3 Representation of convolutional filter outputs before batch normalization or non-linear activation (feature maps) as feature vectors, which are used for projections onto low-dimensional spaces (latent spaces). In this figure, feature maps with $n^{[l+1]}$ channels are represented as $n^{[l+1]}$ slices of unit depth rather than a volume. In addition, higher opacity volumes exemplify layers whose filters can be pruned without problems in the forward propagation of the network

of independent variables $\mathbf{X}^{[l]} \in \mathbb{R}^{m \times n^{[l+1]}}$ for each of the l_p prunable layers.

This filter representation step was defined by Jordao et al. [41] for CNNs in image classification tasks, but it can also be employed in object detectors based on CNNs, such as YOLO, since the filter output will always be a feature map, regardless of the task in which the convolutional network is applied. However, the label representation cannot be transferred directly to the object detection task, where an image can contain objects of different classes and, therefore, have multiple class labels. For example, an image of the dataset may contain 3 fire events and 2 smoke events while another image may contain only 1 smoke event. Therefore, we propose a multiclass modeling to represent the dependent variables. In this representation, class labels are assigned to an image according to the incidence area of each class. Thus, images containing more than one class are assigned the label of the class occupying the most pixels in the image.

Let u_i and v_i be the number of smoke and fire events present in image i, respectively. The label representation starts by creating a real-valued vector $\mathbf{y} \in \mathbb{R}^m$ whose values are given by

$$y_i = \begin{cases} 1, & \text{if } \sum_{q=1}^{v_i} A_q^{(\text{fire})} < \sum_{p=1}^{u_i} A_p^{(\text{smoke})} \\ 2, & \text{if } \sum_{q=1}^{v_i} A_q^{(\text{fire})} \geq \sum_{p=1}^{u_i} A_p^{(\text{smoke})} \\ 0, & \text{if } u_i = v_i = 0 \end{cases} \tag{7}$$

where $A_q^{(\text{fire})}$ indicates the area of the q-th bounding box labeled as fire and $A_p^{(\text{smoke})}$ indicates the area of the p-th bounding box labeled as smoke. The first condition indicates that the sum of the areas of all fire events in image i is less than the sum of the areas of all smoke events. In this case, image i is labeled as smoke ($y_i = 1$). The second condition indicates otherwise, that is, the sum of the areas of all fire events is greater than or equal to the sum of the areas of all smoke events. In this case, image i is labeled as fire ($y_i = 2$). Finally, the third condition indicates an image with no events of any of the classes, i.e., absence of fire and smoke ($y_i = 0$). The dependent variables vector $\mathbf{y}$ is then converted to a binary matrix $\mathbf{Y} \in \mathbb{B}^{m \times (C+1)}$ through one-hot encoding. In other words, class labels $y_i = 0$ are encoded as $\mathbf{y_i} = [1\ 0\ 0]$, class labels $y_i = 1$ are encoded as $\mathbf{y_i} = [0\ 1\ 0]$ and class labels $y_i = 2$ are encoded as $\mathbf{y_i} = [0\ 0\ 1]$.

The next step is to project each of the l_p independent variables onto a latent space $\mathbb{R}^r$ (multiple projections), where $r \ll n^{[l+1]}$. To this end, and similar to what is done in Jordao et al. [41], we employ a discriminative feature projection method called Partial Least Squares (PLS) [67], which estimates a projection matrix $\mathbf{W}^{[l]} \in \mathbb{R}^{n^{[l+1]} \times r}$ for dimensionality reduction. Each component $\mathbf{w_i}^{[l]} \in \mathbf{W}^{[l]} = \left[\mathbf{w_1}^{[l]}, \mathbf{w_2}^{[l]}, \ldots, \mathbf{w_r}^{[l]}\right]$ represents the maximum covariance between the independent and dependent variables

$$\mathbf{w_i}^{[l]} = \arg\max_{\mathbf{w}} \left(\operatorname{cov}\left(\mathbf{X}^{[l]}\mathbf{w}, \mathbf{Y}\right)\right), \qquad \text{subject to: } ||\mathbf{w}|| = 1. \tag{8}$$

To solve the optimization problem defined by Equation (8), we can use Singular Value Decomposition (SVD) [68] or Nonlinear Iterative Partial Least Squares (NIPALS) [69]. We chose NIPALS, since SVD finds all $n^{[l+1]}$ components, whereas NIPALS finds only the first r components, what makes it considerably faster than SVD [41]. NIPALS convergence is achieved when no further changes in $\mathbf{w_i}^{[l]}$. A maximum number of iterations (e.g., 500) is usually defined in order to ensure that the method eventually stops.

After using the NIPALS algorithm to project matrix $\mathbf{X}^{[l+1]}$ for each of the l_p independent variables, the contribution to the low-dimensional latent space of each convolutional filter in a given layer must be estimated. To achieve this goal, the Variable Importance in Projection (VIP) technique [70] is employed, where the importance score $f_j^{[l]}$ of a filter $\mathcal{F}_j^{[l]}$ is computed by

$$f_j^{[l]} = \sqrt{n^{[l+1]} \frac{\sum_{i=1}^{r} S_i \left(\frac{w_{ij}^{[l]}}{||\mathbf{w_i}^{[l]}||^2}\right)}{\sum_{i=1}^{r} S_i}}, \tag{9}$$

where S_i is the sum of squares explained by the i-th component and $w_{ij}^{[l]} / \left|\left|\mathbf{w_i}^{[l]}\right|\right|^2$ is the importance of the j-th filter of layer l. Finally, the $p\%$ of filters with the lowest importance score $f_j^{[l]}$ are pruned in each of the l_p prunable layers of the network.

3.4.3 Clustering

Another set of pruning techniques that consider interactions between convolutional filters are the cluster-based ones. In these techniques, filters in the same convolutional layer are grouped by similarity. The idea behind clustering is to group similar filters in the same cluster and then keep just one filter to represent each cluster. The premise is that similar filters produce similar responses and, therefore, are redundant, contributing little or even worsening the network's output. Inspired by Gosh et al. [44], we propose a pruning technique based on Hierarchical Agglomerative Clustering (HAC), which we call HAC+PCC.

Given a convolutional layer l with $n^{[l+1]}$ filters, the HAC+PCC method calculates a similarity matrix $\mathbf{S} \in$

$\mathbb{R}^{n^{[l+1]} \times n^{[l+1]}}$ based on all the pairwise comparisons between the filters. The similarity between filters $\mathcal{F}_i^{[l]}$ and $\mathcal{F}_j^{[l]}$ is defined by the Pearson correlation coefficient (PCC)

$$\operatorname{corr}\left(\mathcal{F}_i^{[l]}, \mathcal{F}_j^{[l]}\right) = \frac{\operatorname{cov}\left(\mathcal{F}_i^{[l]}, \mathcal{F}_j^{[l]}\right)}{\sqrt{\sigma^2\left(\mathcal{F}_i^{[l]}\right)\sigma^2\left(\mathcal{F}_j^{[l]}\right)}} , \tag{10}$$

where the three-dimensional filters are converted into a one-dimensional vector (flattening), $\operatorname{cov}\left(\mathcal{F}_i^{[l]}, \mathcal{F}_j^{[l]}\right)$ is the covariance between filters $\mathcal{F}_i^{[l]}$ and $\mathcal{F}_j^{[l]}$ and $\sigma^2\left(\mathcal{F}_i^{[l]}\right)$ is the variance of values in the filter $\mathcal{F}_i^{[l]}$.

The flattened filters are clustered using HAC with complete-linkage [44]. The distance matrix is defined as the complement of the similarity matrix ($\mathbf{D} = 1 - \mathbf{S}$) and the number of clusters as the number of filters that we want to keep in layer l, i.e., $\lfloor n^{[l+1]}(1-p) \rfloor$ clusters, where $\lfloor . \rfloor$ denotes the operation of rounding down the number. This process results in similar filters being in the same cluster, while dissimilar filters go to different clusters. Clusters with more than one filter, therefore, suggest redundancy amongst its filters, which are then considered potential candidates for pruning.

In contrast with the original proposal [44], where filters from the same cluster are replaced by a single filter (the average of the filters in the cluster, similar to feature interaction or epistasis), our technique identifies the filters to be pruned by comparing each filter in a multi-filter cluster with all the other clusters.[3] The correlation between filter $\hat{\mathcal{F}}_{ik}^{[l]}$, the k-th filter of cluster $\hat{\mathcal{F}}_i^{[l]}$, and cluster $\hat{\mathcal{F}}_j^{[l]}$ is

$$\operatorname{corr}\left(\mathcal{F}_{ik}^{[l]}, \mathcal{F}_j^{[l]}\right) = \max\left(\left\{\left|\operatorname{corr}\left(\mathcal{F}_{ik}^{[l]}, \mathcal{F}_{j1}^{[l]}\right)\right|, \ldots, \left|\operatorname{corr}\left(\mathcal{F}_{ik}^{[l]}, \mathcal{F}_{j\bar{n}}^{[l]}\right)\right|\right\}\right) , \tag{11}$$

where $\bar{n}$ is the number of filters in cluster $\hat{\mathcal{F}}_j^{[l]}$. The similarity between this filter and all the other clusters is given by

$$\operatorname{sim}\left(\mathcal{F}_{ik}^{[l]}\right) = \max\left(\left\{\left|\operatorname{corr}\left(\mathcal{F}_{ik}^{[l]}, \mathcal{F}_1^{[l]}\right)\right|, \ldots, \left|\operatorname{corr}\left(\mathcal{F}_{ik}^{[l]}, \mathcal{F}_g^{[l]}\right)\right|\right\}\right) , \tag{12}$$

where g is the number of clusters not containing the filter $\hat{\mathcal{F}}_{ik}^{[l]}$, i.e., $\lfloor n^{[l+1]}(1-p) \rfloor - 1$ clusters. After computing the similarity measure for all the filters in cluster $\hat{\mathcal{F}}_i^{[l]}$, only the least similar filter is kept in the cluster and used to represent it. This process is repeated for all the clusters that contain more than one filter. Thus, at the end of this process, only one filter is kept in each cluster. Finally, all the aforementioned procedure is performed for each prunable layer of the network, such that the $p\%$ most homogeneous filters in each prunable layer are removed.

[3] Obviously, clusters with a single filter do not need to undergo this process.

3.4.4 Random

All pruning techniques discussed so far in this section define some criterion for choosing which convolutional filters will be removed from the network. In addition to these techniques, two more strategies are considered here. Both techniques randomly prune $p\%$ of the convolutional filters from each prunable layer of the network. Thus, in these techniques, there is no estimate of the importance of filters in each layer. Random pruning is useful as a baseline for comparison: since criteria-based techniques are considerably more elaborate (and much more computationally expensive), one expects that they also provide better results. Otherwise, they are not worthwhile.

In the first random strategy, called Random+FS, the remaining filter values are discarded and the network is then trained from scratch with its weights randomly initialized. In turn, the second random strategy, called simply Random, does not discard the values of the remaining filters but rather performs a fine-tuning procedure. By considering these two strategies, we can better measure the plasticity of the YOLOv4 network in the fire detection problem. Also, this allows us to verify how the performance of the pruned networks is impacted not only by the way filters are chosen for removal but also by how the remaining filter weights are readjusted.

4 Results

In this section, an analysis is carried out to assess the performance of the network in real scenarios. The trained network is then submitted to an optimization procedure, where the filter pruning techniques described in Sect. 3.4 are applied at nine different pruning rates. This leads to different trade-offs between computational cost and predictive performance. Finally, the pruned networks are evaluated on a low-power, resource-constrained device, namely the Raspberry Pi 4 [71].

4.1 Performance analysis

After the original (unpruned) YOLOv4 and Tiny YOLOv4 networks have been trained (see Sect. 3.3 for how the networks are trained and their hyperparameters are chosen), their performance is evaluated through a fivefold cross-validation procedure. The results are shown in Table 2.

Table 2 Performance metrics on the test set

Network	mAP@0.50 (%)	AP@0.50$_{\text{smoke}}$ (%)	AP@0.50$_{\text{fire}}$ (%)	F_1-score	avg IoU (%)
Tiny YOLOv4	61.4740 ± 0.6488	67.4280 ± 0.8962	55.5200 ± 0.5990	0.6300 ± 0.0071	51.8880 ± 0.5865
YOLOv4	73.9840 ± 0.4029	80.5120 ± 0.4321	67.4520 ± 0.5440	0.7240 ± 0.0055	60.5080 ± 0.3136

Each metric value represents the mean μ and the standard deviation σ of the five models from the fivefold cross-validation, i.e., $\mu \pm \sigma$

From Table 2 we can see that YOLOv4 has average performance superior to Tiny YOLOv4 in all the considered evaluation metrics. This result is expected, since the YOLOv4 architecture has 89 more convolutional layers and, therefore, is capable of extracting more refined features from the images, that is, constituting visual representations in more complex abstraction levels. In other words, the YOLOv4 network detects more true fire and smoke events (true positives) and fewer false events (false positives) than the Tiny YOLOv4 network. Also, the locations of the objects detected by the YOLOv4 network are, on average, more accurate. Hence, we decided to prune only the YOLOv4 network, as pruning the Tiny YOLOv4 network could further reduce its predictive performance, which is already lower.

It is also interesting to note that both networks have greater difficulty in detecting fire than smoke, that is, AP@0.50$_{\text{smoke}}$ > AP@0.50$_{\text{fire}}$. A possible explanation is that the labeling of fire events for training tends to be more subjective, such that small fire events next to each other can be labeled either individually (one bounding box per object) or collectively by a single bounding box. In addition, the fire events present in the D-Fire images are usually in the early stages of propagation and, therefore, are associated with regions with a smaller number of pixels. Thus, the processing of these regions by several layers can considerably reduce the spatial resolution of feature maps throughout the architecture and, consequently, make small fire spots difficult to see in these low-resolution maps. This is a known limitation of single-stage object detectors, such as YOLO, that has been investigated since the first proposed architectures and has already witnessed a significant improvement to date [48, 49].

Although the YOLOv4's performance metrics are not exceptionally large, i.e., above 90%, their values should be carefully analyzed. The metrics AP@0.50, mAP@0.50 and F_1-score, in particular, penalize whenever an object of interest is not detected, which is plausible. However, there are a large number of imagens in the dataset that have more than one fire spot, some of which are quite small. Failing to detect fire or smoke in an image with multiple fire spots significantly affects these metrics, but may not be as critical in real monitoring with a mid-range static camera, for example. In this context, the camera's field of view is constant and the distance from the camera to the monitored area is not large. Thus, identifying a single fire or smoke event can already be decisive for the suppression of multiple outbreaks in the same region. Figure 5b illustrates this case.

In order to validate such considerations, the F_1-score was re-calculated, on the test set, for the five models of each network obtained in the fivefold cross-validation, but in two different scenarios that disregard multiple objects per scene and, consequently, their locations. The first simplification turns the problem into a multiclass classification problem ($C = 4$), where the possible class labels for each image were defined as: (i) background, (ii) smoke, (iii) fire and (iv) fire and smoke. We also converted the detection problem into a binary classification problem ($C = 2$), where the possible class labels for each image can only be fire or non-fire. In other words, we had to simplify the output of our detection network, as it predicts multiple bounding boxes with their respective probabilities of fire and smoke classes. If there is at least one fire or smoke event in the image, we define that the scene has a fire. Otherwise, that is, if there is no occurrence of these two classes, the simplified output of our model is non-fire. This second simplification is a modeling of the fire recognition problem widely used by previous works [23, 27, 28, 30], which validate their proposed convolutional neural networks in datasets such as BoWFire [72]. For future comparisons, we also evaluated in the BoWFire test set ($C = 2$) our detection networks trained on the D-Fire dataset, as both datasets have images with very similar real situations (e.g., fire accidents). Figure 4 shows some positive examples of the BoWFire dataset, i.e., examples that are labeled as fire.

According to Table 3, the F_1 values of the Tiny YOLOv4 and YOLOv4 networks increased, on average, about 35.81% and 25.98%, respectively, in the multiclass classification scenario of the D-Fire dataset ($C = 4$) when compared to the mean F_1 values in the detection task (Table 2). In the binary classification scenario of the D-Fire dataset ($C = 2$), these increases were even greater. The F_1 values increased by 46.21% for the Tiny YOLOv4 network and by 33.12% for the YOLOv4 network. This level of predictive performance for binary classification is also achieved in the BoWFire test set and demonstrates a good

Fig. 4 Some positive examples of BoWFire dataset [72]

Table 3 Network performance for different classification scenarios according to the F_1-score on test sets

Network	D-Fire ($C = 4$)	D-Fire ($C = 2$)	BoWFire
Tiny YOLOv4	0.8556 ± 0.0045	0.9211 ± 0.0043	0.9224 ± 0.0126
YOLOv4	0.9121 ± 0.0029	0.9638 ± 0.0033	0.9421 ± 0.0082

generalization ability of networks trained on our dataset. In addition to highlighting the greater complexity of detecting multiple fires, these results show that the trained detectors are able to satisfactorily identify the presence of fire or smoke in a scene, albeit not identifying all individual occurrences.

In the case of the IoU metric presented in Table 2, very high values indicate that the predicted bounding boxes closely match the bounding boxes defined in the labeling process (ground-truth). Considering that the classes are very abstract and that delimiting their regions of incidence when building the dataset is a manual process with considerable inter-annotator variability, a value above 50%, which was observed for both networks, already implies a good location of the objects [55].

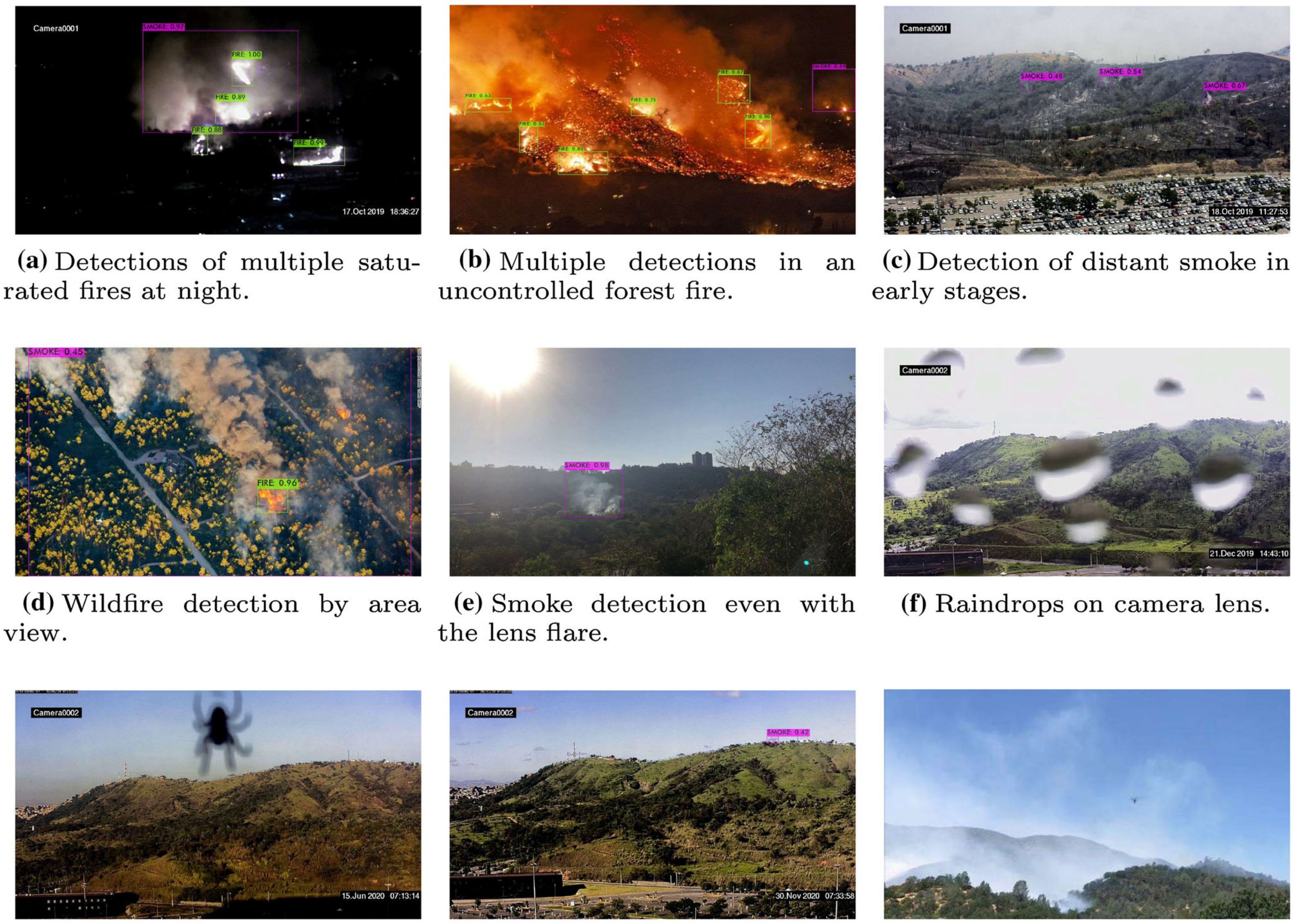
Some detections on the test set are presented in Fig. 5 with the aim of better illustrating YOLOv4's performance in identifying fire and smoke. Figure 5a–e illustrate successful detections in real fire situations under a variety of angles, environments and lighting conditions. Figure 5f–g show some of the challenging scenarios, i.e., scenes with a high degree of uncertainty involved, typical of surveillance videos, such as raindrops and insects on the camera lens. Such robustness can be mainly attributed to the D-Fire dataset, whose images allow to satisfactorily model the intraclass variance of fire and smoke and, consequently, provide a generalized learning to the network. Figure 5h and i, in turn, present some limitations, that is, a false positive and a false negative, respectively.

4.2 Architecture optimization

The six pruning techniques, i.e., Random+FS, Random, ℓ_1-norm, ℓ_2-norm, PLS(Multi)+VIP with $r = 2$ components (as suggested in [41]), and HAC+PCC, are used to prune the trained YOLOv4 network at nine different pruning rates (from 10% through 90%). Since all techniques remove a constant number of filters per layer, the computational cost of the optimized architectures, after pruning, is the same for all of them. This is shown in Table 4. It is interesting to note the removing $p\%$ of the filters in each layer of the network does not necessarily imply eliminating $p\%$ of the parameters of the full architecture. Given that a filter is a three-dimensional tensor and that the YOLOv4 detection network is composed mostly of convolutional layers, removing $p\%$ of its filters implies removing more than $p\%$ of the total parameters of the network, which shows that filter pruning can be a very promising approach to the problem of compressing deep convolutional networks.

The same hyperparameters defined for training were then considered for the fine-tuning step, with the exception of the learning rate, which was individually tuned for each pruning technique. To avoid a high number of experiments, the learning rate tuning was carried out through the Grid Search method with fivefold cross-validation of each technique at only two arbitrary pruning rates (p = 30% and p = 80%). Coincidentally, the learning rate that maximized the mean mAP@0.50 (across the five folds) at these two pruning rates was 0.005 for all techniques. This learning rate was also used to all other values of p. Figure 6 shows the predictive performance of the pruned networks on the test set, as measured by the mAP@0.50, AP@0.50 (fire and smoke classes), F_1-score and avg IoU metrics, as a function of the pruned network's computational cost in BFLOPs (10^9 FLOPs).

According to the trade-off curves presented in Fig. 6, it is possible to note, across all evaluation metrics considered, that there is not a big difference between the performances of the networks optimized by different pruning techniques. These results are similar to those reported by Mittal et al. [43] with the Faster R-CNN network and the PASCAL VOC 2007 dataset and show that the YOLOv4 network considered in this work also presents high plasticity. In other words, given the same pruning rate, our fire detection network achieves similar performance recoveries after fine-tuning regardless of the technique used to remove the less promising convolutional filters. Despite the small differences between the investigated techniques and the fact that, because of the high computational cost, each network was



(a) Detections of multiple saturated fires at night.

(b) Multiple detections in an uncontrolled forest fire.

(c) Detection of distant smoke in early stages.

(d) Wildfire detection by area view.

(e) Smoke detection even with the lens flare.

(f) Raindrops on camera lens.

(g) Spider walking on camera lens.

(h) Cloud detected as smoke.

(i) Sparse smoke not detected.

Fig. 5 Qualitative results. YOLOv4 network running on images from the test set with $t_{\mathrm{conf}} = 0.40$. The green and pink bounding boxes indicate fire and smoke events, respectively, and the values in each box correspond to the YOLOv4 network's confidence score in detection

Table 4 Relation between the computational cost of the YOLOv4 network and the pruning rate used to optimize its architecture

Pruning rate p (per layer) (%)	Pruned filters	Parameter reduction (%)	Memory (MB)	BFLOPs ($416 \times 416 \times 3$)
10	3261	18.81	207.90	48.610
20	6581	35.82	164.37	38.497
30	9888	50.78	126.08	29.506
40	13,208	63.80	92.75	21.758
50	16,576	74.96	64.18	14.988
60	19,837	83.88	41.33	9.768
70	23,157	90.91	23.34	5.561
80	26,464	95.92	10.49	2.516
90	29,784	98.96	2.69	0.674

submitted to the fine-tuning step only once (single pruning), some interesting aspects can be discussed. For example, we note that, as the pruning rate increases, the performance differences between the different techniques also increase, especially for mAP@0.50, AP@0.50 (fire and smoke) and F_1-score metrics. For $p \geq 50\%$, we note that at least one of the importance-based techniques (ℓ_1-norm or ℓ_2-norm) outperforms all other techniques. In

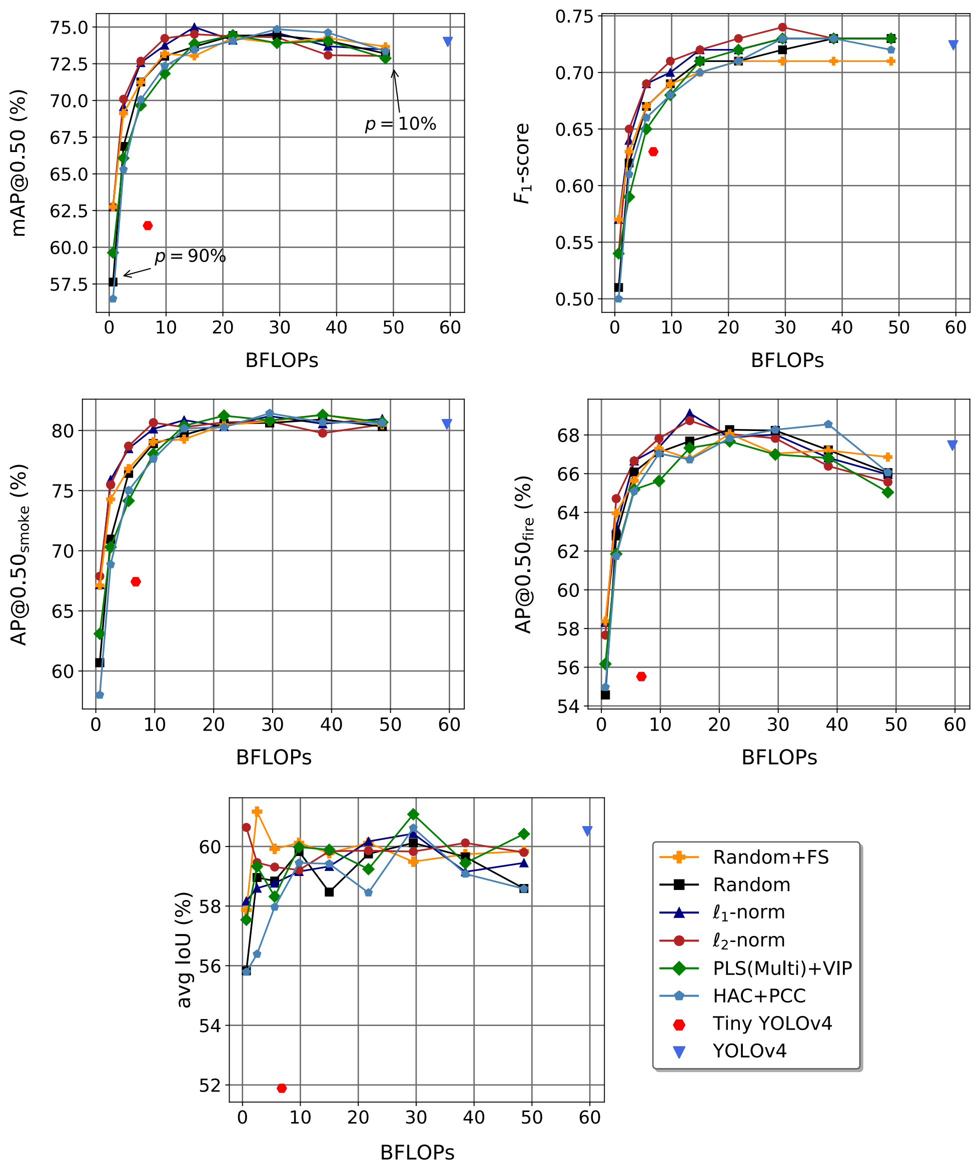


Fig. 6 Trade-off curves obtained by different filter pruning techniques. The five panels correspond to the five evaluation metrics described in Sect. 2.3. For comparison purposes, the individual markers at the top right and lower left corner of the panels represent, respectively, the performance of the unpruned YOLOv4 and Tiny YOLOv4 networks as shown in Table 2

general, filters with small norms produce activation maps that lead to relatively lower activation values and therefore removing them from the network usually has a smaller impact on the final prediction [40]. Moreover, as ℓ_1-norm and ℓ_2-norm techniques are less expensive in the filter selection process (since they do not need to process data to rank filters by importance, i.e., they are data-agnostic), they are also preferable for $p <$ 50% in our case. Also, the Random and HAC+PCC techniques typically have the worst performances.

Another aspect to be highlighted in Fig. 6 is the shape of the curves for the avg IoU metric. In contrast to what is observed for the other evaluation metrics, the IoU curves do not show a trade-off between performance and computational cost at different pruning rates. This happens because the IoU only measures the quality of the predicted bounding boxes with respect to the ground-truth bounding boxes and, therefore, does not penalize situations where the network fails to detect one or more objects in the scene (false negatives). Thus, high IoU values do not correspond to a high number of correct fire and smoke detections, but rather to good locations of the events that were correctly detected.

An intriguing observation concerning the mAP@0.50, as well as the AP@0.50 of both classes, is that, although not taking advantage of the remaining filters to initialize the fine-tuning procedure, the Random+FS technique still presented competitive results. However, these results must be interpreted with caution. As discussed previously, the AP metric is calculated based on a range of confidence threshold values. However, the trained detection networks are deployed using a fixed confidence threshold, whose best value for YOLOv4 in our case was $t_{\text{conf}} = 0.4$. A fairer comparison can be made using the F_1-score metric, which allows an evaluation based exclusively on the best confidence threshold, simulating deployment. In this case, for $p \leq 30\%$ all pruning techniques lead to a pruned network whose performance is superior to that obtained by Random+FS technique. For higher pruning rates, this is no longer true. For pruning rates greater than 40%, network complexity is greatly reduced, with more than 63.80% of the parameters having been removed (see Table 4). Thus, since the number of iterations for fine-tuning was fixed at 30,000 iterations for all pruning rates, the remaining, randomly initialized, filters, which are fewer now, are able to converge to a local optimum with similar quality to the local optimum found by the other techniques (all of them using the remaining filters as a starting point for fine-tuning).

From this perspective, an additional experiment was carried out in order to analyze the network performance for all pruning techniques, for p = 80%, as a function of the number of fine-tuning iterations. We evaluated six equally spaced number of iterations in the range from 5000 to 30,000 iterations using the multistep scheduler with a 0.1 reduction factor at the 80% and 90% marks of the total iterations. For example, considering a 15,000-iteration fine-tuning process, the learning rate, which was previously set in 0.005, will decrease by 90% at 12,000 and 13,500 iterations. The other hyperparameters are kept the same.

Figure 7 shows that the difference in performance of the pruned networks decreases as number of iterations used in the fine-tuning process increases. The highest difference, which happens at 5000 iterations, is 25.21% for the mAP@0.50 and 51.72% for the F_1-score between the ℓ_2-norm and HAC+PCC techniques. At 30,000 iterations, the difference between these same techniques is 7.34% for the mAP@0.50 and 6.56% for the F_1-score. In addition, a training convergence trend can be observed from the line segments present between consecutive points of the curves, whose slopes decrease as the number of iterations increases. For example, in the interval between 25,000 and 30,000 iterations, the F_1-score decreases for the Random+FS and the PLS(Multi)+VIP techniques while the mAP@0.50 metric decreases for the PLS(Multi)+VIP technique, suggesting that excessive fine-tuning iterations can depreciate the generalization ability of the network.

Our experimental results also show that it is possible to preserve or even surpass the performance of the fire detection networks presented in Table 2 at a very low computational cost. Pruning 50% of the convolutional filters in each prunable layer of the YOLOv4 network using the ℓ_1-norm, for example, results in reductions of 74.96% in network parameters, 74.93% in memory and 74.84% in BFLOPs, with an increase of 1.37% in the mAP@0.50 on the test set. The highest pruning rate that results in a network whose predictive performance according to the mAP@0.50 metric is closest to the unpruned YOLOv4 network is $p = 60\%$ using the ℓ_2-norm technique. In scenarios with a constrained computation budget, even lighter networks may be required. In these cases, a network pruned at p = 70% could be used. At this rate, all pruned networks significantly outperform the Tiny YOLOv4 network in all evaluation metrics, as well as reduce the computation cost and memory consumption by 18.09% and 0.81%, respectively. In the best performance case for p = 70%, where the network is also pruned using the ℓ_2-norm, the improvements compared to the Tiny YOLOv4 network are about 18.24% in the mAP@0.50, 16.71% in the AP@0.50$_{\text{smoke}}$, 20.06% in the AP@0.50$_{\text{fire}}$, 9.52% in the F_1-score and 14.29% in the avg IoU, confirming once again the benefits of removing redundant or irrelevant convolutional filters from the network.

Similar to Table 3, we also evaluated the predictive performance of the best pruned network at each pruning rate in binary and multiclass classification scenarios and compare them in the BoWFire test set with four convolutional networks of previous works [23, 27, 28, 30]. All these CNNs were designed by experts to perform the

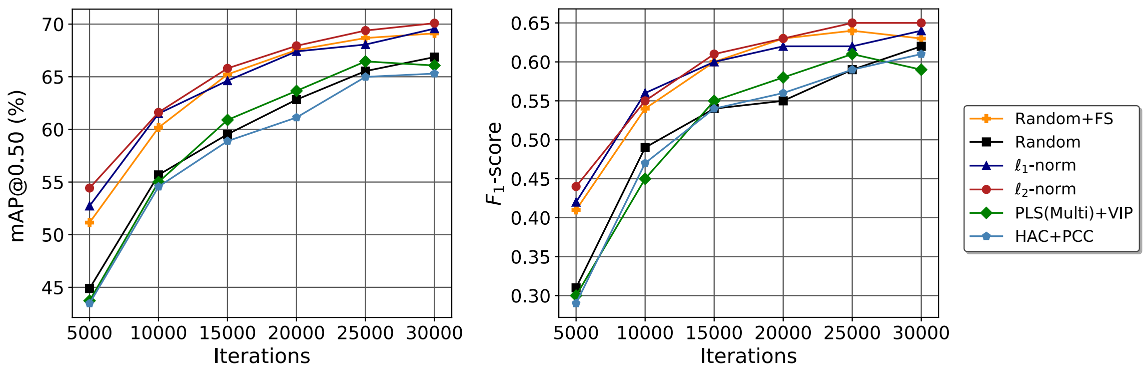


Fig. 7 Convergence analysis. Performance of pruned networks with $p = 80\%$ according to mAP@0.50 (left) and F_1-score (right) on the test set as a function of the number of fine-tuning iterations

binary classification task (fire and non-fire), that is, they do not discriminate fire from smoke. Additionally, only one of these methods provide fire location [27]. This location is estimated through a binarization stage right after the CNN prediction and results in an additional computational cost during image processing. In contrast, all versions of YOLO predicts bounding boxes and class probabilities directly from full images in one single evaluation [46] and does not require any post-processing steps to estimate the location of objects. The results of these previous methods were taken from their original papers.

Based on Table 5, we can highlight the YOLOv4 network pruned at $p = 20\%$ using the HCC+PCC technique, which outperforms its unpruned counterpart in all scenarios and test sets even with a parameter reduction of 35.82%. This pruned network has a F_1-score of 5.72% more than the EMNFire [30], the best one among the previous fire classification networks. However, its memory consumption is only lower than the memory consumption of ANetFire, which is 233MB (41.75% more than ours) [23]. If we prune to a maximum of $p = 60\%$, we still get better predictive results than all previous methods, but now we also have a network with a lower memory consumption than GNetFire, which is 43.30MB (4.77% more than ours) [28]. In summary, these results show that we can obtain efficient networks for fire classification without designing a specific architecture for this purpose from scratch.

4.3 Case study

Finally, we conducted a case study to analyze the optimized fire detection networks on a low-power, resource-constrained device. The experiments were performed on a Raspberry Pi 4 Model B with Broadcom BCM2711, Quad Core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz, 4 GB of RAM, OpenGL ES 3.1, Vulkan 1.0 and a 32 GB Micro-SD Card used for both operating system installation and data storage. The main aspects investigated were: (i) initialization time, that is, the time required to load all parameters of the YOLOv4 network, (ii) detection time, that is, the time taken to detect fire and smoke events in a single 416×416 RGB image and (iii) energy consumption when one of our optimized networks is running. Each network (pruned and unpruned) was run 35 times. In order to prevent the device's performance from being degraded due to overheating ($\geq 80°$C) during the experiments, a dual fan cooler was attached to the single-board computer.

From Fig. 8, it is possible to note that, as the pruning rate increases, both the initialization and the detection times decrease (not only the average values, but also the standard deviations). Another interesting observation is that the variation in the average initialization times, even between the two extreme pruning rates (i.e., p = 10% and p = 90%), is quite small compared to the average detection times. We noticed the same trend of detection time for energy consumption, which is to be expected, since energy consumption is given by the power consumption (constant at 6.3W) times the detection time in seconds. This further corroborates the relevance of filter pruning as an alternative to reduce energy consumption and increase the detection speed of convolutional networks, since the initialization of parameters is done only once while the detection, in contrast, is performed for each video frame.

Although employing intensive pruning results in faster networks, this behavior is predictable, since filter pruning directly implies parameter reduction, as well as BFLOPs reduction. The ideal scenario would be to reduce energy consumption and initialization and detection times as much

Table 5 Performance of pruned YOLOv4 networks (best at each pruning rate) for different classification scenarios and also of other fire classification networks according to the F_1-score on test sets

Method	D-Fire ($C = 4$)	D-Fire ($C = 2$)	BoWFire
Random+FS ($p = 10\%$)	0.8941	0.9531	0.9143
HAC+PCC ($p = 20\%$)	0.9187	0.9662	0.9672
HAC+PCC ($p = 30\%$)	0.9078	0.9619	0.9547
PLS(Multi)+VIP ($p = 40\%$)	0.9055	0.9605	0.9237
ℓ_1-norm ($p = 50\%$)	0.8967	0.9518	0.9388
ℓ_2-norm ($p = 60\%$)	0.8983	0.9557	0.9312
ℓ_2-norm ($p = 70\%$)	0.8639	0.9298	0.9076
ℓ_2-norm ($p = 80\%$)	0.8223	0.8990	0.8722
ℓ_2-norm ($p = 90\%$)	0.7236	0.8029	0.8037
ANetFire [23]	–	–	0.8809
CNNFire [27]	–	–	0.8946
GNetFire [28]	–	–	0.8543
EMNFire [30]	–	–	0.9148

Unfortunately, it was not possible to evaluate the performance of previous methods on D-Fire test set, since the source codes of these networks were not available (symbol "–")

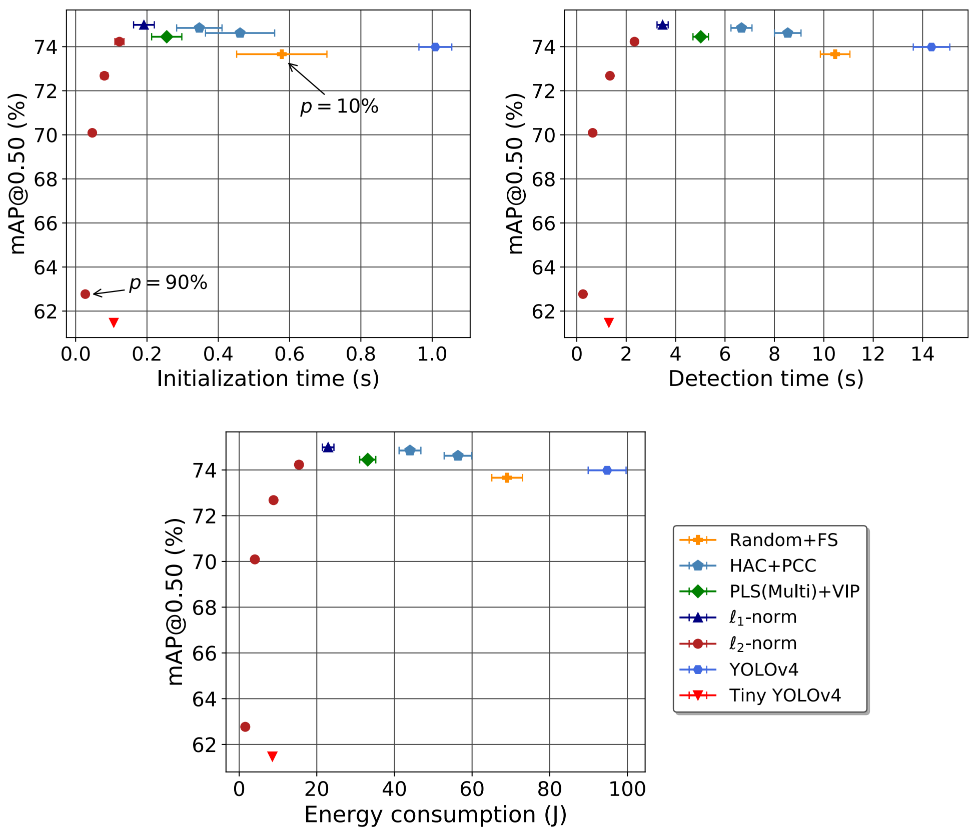


Fig. 8 Pruned networks with the best predictive performances on the test set, for each pruning rate, and unpruned networks (YOLOv4 and Tiny YOLOv4) according to the average performances in the fivefold cross-validation (shown in Table 2) on a Raspberry Pi 4 Model B. The horizontal bars indicate the standard deviation of the corresponding metric over 35 runs. Markers without visible bars, such as those for high pruning rates, indicate a very low standard deviation

as possible while causing as little impact as possible on the network's predictive performance. This is also shown in Fig. 8, where only the best performance according to the mAP@0.50 on the test set is shown for each pruning rate. The performance of the best network (ℓ_1-norm at $p = 50\%$), when compared to the unpruned YOLOv4 network, is 1.36% better on the test set in terms of the mAP@0.50 metric, while requiring, on average, only 1/5 of the initialization time, 1/4 of the detection time and, consequently, 1/4 of the energy consumption. As previously discussed (and seen in Fig. 6), $p = 60\%$ is the rate that results in a pruned network with performance closest to the original, unpruned YOLOv4 network. At this rate, the pruned network, although associated with a slight (0.34%) increase in the mAP@0.50, results in a reduction in the initialization and the detection times on the Raspberry Pi of 87.84% and 83.73%, respectively. In the case of an even more prohibitive computational budget, a network pruned at $p = 70\%$ using the ℓ_2-norm could be used. This network results in average initialization and detection times basically equivalent to those of the Tiny YOLOv4 network, but with an mAP@0.50 18.24% better on the test set.

Finally, we compare our fire detection network pruned at $p = 70\%$ using ℓ_2-norm technique with a state-of-the-art fire detection method, which also uses a YOLO series algorithm, particularly YOLOv2 for fire and smoke detection [73]. This previous work was deployed in a Jetson Nano [74], an embedded computing board from NVIDIA that has a GPU and costs about \$120. The design metrics of embedded systems used for comparative purposes and their respective values for both detection network are shown in Table 6. In this case, we measure the predictive performance according to the F_1-score calculated in the context of the fire and smoke detection task (similar to Table 2), since both networks are designed for object detection and this task is typically more challenging (as discussed in Sect. 4.1).

Our contributions are quite clear. We implemented a much deeper network (4.58 times more convolutional layers), which also has better predictive performance and faster detections, on a resource-constrained device with no GPU and which costs 3.43 times less than Jetson Nano. However, to maintain this good performance, the Raspberry Pi 4 typically uses 96% of its processing power in the CPU, which implies a higher average system temperature and, therefore, a higher power consumption. This happens because our device is quite computationally limited. If we ran our system on a Jetson Nano, for example, we could possibly improve some of our design metrics, such as our detection times, temperature measurement for the CPU and resource utilization, since now there would be a GPU to do part of the massive processing.

Table 6 Evaluation of the proposed fire detection system and a previous work according to design metrics of embedded systems

Network	Layers*	Device	Price ($)	Power (W)	CPU (%)	GPU (%)	F_1-score	Detection time (s)	Temperature (°C)
ℓ_2-norm ($p = 70\%$)	110	Raspberry Pi 4 Model B	35.00	6.30	96.00	–	0.69	1.34	59.00
YOLOv2 [73]	24	NVIDIA Jetson Nano	120.00	5.19	53.10	99.00	0.67	2.00	53.10

In both cases the devices are connected to a keyboard, a mouse and a monitor

*Only convolutional layers

The results presented in this case study also show that there is a notable opportunity to monitor large green areas efficiently using multiple cameras. In this system, each surveillance camera would be integrated into a low-cost device with limited computing resources responsible for running the proposed fire detection system. Such infrastructure would not require servers with expensive graphics processing units (GPUs) or cloud computing, but rather much cheaper single-board computers or mobile devices.

5 Conclusions

Among the modern approaches developed for automatic fire detection, the best ones are typically based on deep convolutional neural networks. Unlike handcrafted feature extraction methods, this visual perception-inspired topology is able to automatically extract highly relevant patterns directly from the raw image pixels. However, CNNs are computationally expensive to operate successfully in production environments, usually requiring the use of GPUs, which makes current proposals for large-scale fire identification based on distributed local processing very costly.

To address this limitation, this work proposed a computationally efficient automatic fire detection system based on deep convolutional networks. The development pipeline for this system started with the training of the YOLOv4 detection network, whose architecture, designed for two classes (fire and smoke), has about 64 million parameters, with most of them coming from 33,215 convolutional filters arranged in 110 convolutional layers. This network was trained to detect fire and smoke, the main visual indicators of wildfires, on our D-Fire dataset, which contains a variety of real fire events. Next, the network's complex architecture was optimized by pruning the less important convolutional filters. For this purpose, classical techniques found in the literature were investigated with the aim of maximizing network compression while preserving as much predictive performance as possible. Since most filter pruning techniques are used for CNNs in classification tasks, it was necessary to modify some of them so that they could be used in the fire detection problem. We have shown that it is possible to remove network parameters by up to 83.88% and reduce the computational cost (in BFLOPs) by up to 83.60%, resulting in a reduction in the detection time on a Raspberry Pi 4 of up to 83.73% without degrading network performance.

Despite the encouraging results presented in this work, many opportunities for future work are possible. In order to adapt the label representation originally proposed in classification tasks to our problem, the class with the greatest presence in an image was defined as the label of the image. However, this assumption disregards multiple distinct objects in a scene, as well as their locations, and, therefore, impairs the true relationship of convolutional filters with labels in a low-dimensional latent space. A possible alternative is to model the output through class label smoothing, which allows to relax the model's confidence in the true label by defining each label as a class probability rather than just a binary value. Other possibilities include layer pruning, with interesting results already reported in the literature [75, 76], and also iterative pruning, which was not investigated in this work due to the high computational costs involved in the experiments. Evaluating different CNN architectures could also lead to improvements in fire detection, which is substantially worse than smoke detection.

Acknowledgements Financial support for this work was provided by CEMIG-ANEEL (R&D project D0619), by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) to Adriano Chaves Lisboa (Grant 304506/2020-6), by Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) to Adriano Vilela Barbosa (Grant APQ-03701-16), and by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Author Contributions PVABV: Methodology; Formal analysis; Software; Writing—Original Draft, Visualization. ACL: Formal analysis; Writing—Review and Editing; Validation. AVB: Formal analysis; Supervision; Writing—Review and Editing.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Finney MA (2021) The wildland fire system and challenges for engineering. Fire Safety J 120: 103085 (Fire Safety Science: Proceedings of the 13th International Symposium)
2. Joglar F, Mowrer F, Modarres M (2005) A probabilistic model for fire detection with applications. Fire Technol 41(3):151–172
3. Töreyin BU (2018) Smoke detection in compressed video. In: Applications of digital image processing XLI, vol. 10752. International Society for Optics and Photonics, p 1075232
4. Singh A, Singh H (2012) Forest fire detection through wireless sensor network using type-2 fuzzy system. Int J Comput Appl 52(9)
5. Remagnino P, Jones GA, Paragios N, Regazzoni CS (2002) Video-based surveillance systems: computer vision and distributed processing
6. Chen T-H, Wu P-H, Chiou Y-C (2004) An early fire-detection method based on image processing. In: 2004 International conference on image processing, 2004, vol 3. ICIP'04. IEEE, pp 1707–1710
7. Chen T-H, Yin Y-H, Huang S-F, Ye Y-T (2006) The smoke detection for early fire-alarming system base on video processing. In: 2006 international conference on intelligent information hiding and multimedia. IEEE, pp 427–430
8. Yu C, Mei Z, Zhang X (2013) A real-time video fire flame and smoke detection algorithm. Proc Eng 62:891–898
9. Töreyin BU, Dedeoğlu Y, Cetin AE (2005) Wavelet based real-time smoke detection in video. In: 2005 13th European signal processing conference. IEEE, pp 1–4
10. Marbach G, Loepfe M, Brupbacher T (2006) An image processing technique for fire detection in video images. Fire Saf J 41(4):285–289
11. Lascio R.D, Greco A, Saggese A, Vento M (2014) Improving fire detection reliability by a combination of videoanalytics. In: International conference image analysis and recognition. Springer, pp 477–484
12. Celik T, Demirel H, Ozkaramanli H, Uyguroglu M (2007) Fire detection using statistical color model in video sequences. J Vis Commun Image Represent 18(2):176–185
13. Chunyu Y, Jun F, Jinjun W, Yongming Z (2010) Video fire smoke detection using motion and color features. Fire Technol 46(3):651–663
14. Cheng X, Wu J, Yuan X, Zhou H (1999) Principles for a video fire detection system. Fire Saf J 33(1):57–69
15. Yuan F, Fang Z, Wu S, Yang Y, Fang Y (2015) Real-time image smoke detection using staircase searching-based dual threshold adaboost and dynamic analysis. IET Image Proc 9(10):849–856
16. Schultze T, Kempka T, Willms I (2006) Audio-video fire-detection of open fires. Fire Saf J 41(4):311–314
17. Morerio P, Marcenaro L, Regazzoni CS, Gera G (2012) Early fire and smoke detection based on colour features and motion analysis. In: 19th International conference on image processing. IEEE, pp 1041–1044
18. Cheng C, Sun F, Zhou X (2011) One fire detection method using neural networks. Tsinghua Sci Technol 16(1):31–35
19. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
20. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Adv Neural Inf Process Syst 25:1097–1105
21. Tao C, Zhang J, Wang P (2016) Smoke detection based on deep convolutional neural networks. In: 2016 International conference on industrial informatics-computing technology, intelligent technology, industrial information integration (ICIICII). IEEE, pp 150–153
22. Yin Z, Wan B, Yuan F, Xia X, Shi J (2017) A deep normalization and convolutional neural network for image smoke detection. IEEE Access 5:18429–18438
23. Muhammad K, Ahmad J, Baik SW (2018) Early fire detection using convolutional neural networks during surveillance for effective disaster management. Neurocomputing 288:30–42
24. Govil K, Welch ML, Ball JT, Pennypacker CR (2020) Preliminary results from a wildfire detection system using deep learning on remote camera images. Remote Sensing 12(1):166
25. Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S et al (2017) Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7310–7311
26. Malach E, Shalev-Shwartz S (2019) Is deeper better only when shallow is good? Adv Neural Inf Process Syst 32:6429–6438
27. Muhammad K, Ahmad J, Lv Z, Bellavista P, Yang P, Baik SW (2018) Efficient deep cnn-based fire detection and localization in video surveillance applications. IEEE Trans Syst Man Cybern Syst 49(7):1419–1434
28. Muhammad K, Ahmad J, Mehmood I, Rho S, Baik SW (2018) Convolutional neural networks based fire detection in surveillance videos. IEEE Access 6:18174–18183
29. Jadon A, Omama M, Varshney A, Ansari MS, Sharma R (2019) Firenet: a specialized lightweight fire & smoke detection model for real-time iot applications. arXiv preprint arXiv:1905.11922
30. Muhammad K, Khan S, Elhoseny M, Ahmed SH, Baik SW (2019) Efficient fire detection for uncertain surveillance environment. IEEE Trans Industr Inf 15(5):3113–3122
31. Jordao A, Akio F, Lie M, Schwartz WR (2021) Stage-wise neural architecture search. In: 2020 25th international conference on pattern recognition (ICPR). IEEE, pp 1985–1992
32. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. Adv Neural Inf Process Syst
33. Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) Xnor-net: Imagenet classification using binary convolutional neural networks. In: European conference on computer vision. Springer, pp 525–542
34. Zhu S, Duong L.H, Liu W (2020) Xor-net: an efficient computation pipeline for binary neural network inference on edge devices. In: 2020 IEEE 26th international conference on parallel and distributed systems (ICPADS). IEEE, pp 124–131
35. Howard A.G, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861
36. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C (2018) Mobilenetv2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4510–4520
37. Buciluǎ C, Caruana R, Niculescu-Mizil A (2006) Model compression. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, pp 535–541
38. Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. arXiv:1503.02531
39. Li H, Kadav A, Durdanovic I, Samet H, Graf H.P (2017) Pruning filters for efficient convnets. In: Proceedings of the international conference for learning representations, pp 1–5
40. He Y, Kang G, Dong X, Fu Y, Yang Y (2018) Soft filter pruning for accelerating deep convolutional neural networks. arXiv:1808.06866

41. Jordao A, Yamada F, Schwartz WR (2020) Deep network compression based on partial least squares. Neurocomputing 406:234–243
42. Jordao A, Lie M, Schwartz WR (2020) Discriminative layer pruning for convolutional neural networks. IEEE J Selected Topics Sig Process 14(4):828–837
43. Mittal D, Bhardwaj S, Khapra MM, Ravindran B (2018) Recovering from random pruning: on the plasticity of deep convolutional neural networks. In: 2018 IEEE winter conference on applications of computer vision (WACV). IEEE, pp 848–857
44. Ghosh S, Srinivasa S.K, Amon P, Hutter A, Kaup A (2019) Deep network pruning for object detection. In: 2019 IEEE international conference on image processing (ICIP). IEEE, pp 3915–3919
45. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, vol. 28, pp 91–99
46. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You Only Look Once: unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 779–788
47. Redmon J, Farhadi A (2017) YOLO9000: better, faster, stronger. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7263–7271
48. Redmon J, Farhadi A (2018) YOLOv3: an incremental improvement. arXiv:1804.02767
49. Bochkovskiy A, Wang C.-Y, Liao H-YM (2020) YOLOv4: optimal speed and accuracy of object detection
50. Felzenszwalb PF, Girshick RB, McAllester D, Ramanan D (2009) Object detection with discriminatively trained part-based models. IEEE Trans Pattern Anal Mach Intell 32(9):1627–1645
51. Wang C-Y, Liao H-YM, Wu Y-H, Chen P-Y, Hsieh J-W, Yeh I-H (2020) CSPNet: a new backbone that can enhance learning capability of CNN. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, pp 390–391
52. Deng J, Dong W, Socher R, Li L.-J, Li K, Fei-Fei L (2009) ImageNet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. IEEE, pp 248–255
53. He K, Zhang X, Ren S, Sun J (2015) Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Trans Pattern Anal Mach Intell 37(9):1904–1916
54. Liu S, Qi L, Qin H, Shi J, Jia J (2018) Path aggregation network for instance segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 8759–8768
55. Everingham M, Van Gool L, Williams CK, Winn J, Zisserman A (2010) The PASCAL visual object classes (VOC) challenge. Int J Comput Vision 88(2):303–338
56. Redmon J, Bochkovskiy A (2013) Darknet: open source neural networks in C. https://git.io/JTICL (Downloaded in January, 2021)
57. Venâncio PVAB, Rezende TM, Lisboa AC, Barbosa AV (2021) Fire detection based on two-dimensional convolutional neural network and temporal analysis. In: 2021 IEEE Latin American conference on computational intelligence (LA-CCI). IEEE, pp 1–6
58. Zhang Q-X, Lin G-H, Zhang Y-M, Xu G, Wang J-J (2018) Wildland forest fire smoke detection based on Faster R-CNN using synthetic smoke images. Procedia Eng 211:441–446
59. Gaia (2018) solutions on demand: D-Fire: an image data set for fire detection. https://git.io/JONna (Downloaded in February 2021)
60. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) PyTorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst 32:8026–8037
61. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830
62. de Venâncio PVAB (2021) Pruning techniques of convolutional neural networks implemented in the Darknet framework. https://git.io/JmjNB
63. Bottou L (1998) Online learning and stochastic approximations. Online Learn Neural Netw 17(9):142
64. Polyak BT (1964) Some methods of speeding up the convergence of iteration methods. Ussr Comput Math Math Phys 4(5):1–17
65. Tikhonov A (1963) On solving ill-posed problem and method of regularization. In: Doklady Akademii Nauk USSR, vol 153, pp 501–504
66. Sadeghi MA, Forsyth D (2014) 30hz object detection with dpm v5. In: European conference on computer vision. Springer, pp 65–79
67. Wold H (1985) Partial least squares. Encyclopedia of Statistical Sciences, Wiley, pp 581–591
68. Eckart C (1936) Young G The approximation of one matrix by another of lower rank. Psychometrika 1(3):211–218
69. Abdi H (2010) Partial least squares regression and projection on latent structure regression (pls regression). Wiley Interdisc Rev Comput Stat 2(1):97–106
70. Mehmood T, Liland KH, Snipen L, Sæbø S (2012) A review of variable selection methods in partial least squares regression. Chemom Intell Lab Syst 118:62–69
71. Raspberry Pi (2022) Foundation: Raspberry Pi 4 Model B. https://www.raspberrypi.com/
72. Chino DY, Avalhais LP, Rodrigues JF, Traina AJ (2015) BoW-Fire: detection of fire in still images by integrating pixel color and texture analysis. In: 2015 28th SIBGRAPI conference on graphics, patterns and images. IEEE, pp 95–102
73. Saponara S, Elhanashi A, Gagliardi A (2021) Real-time video fire/smoke detection based on cnn in antifire surveillance systems. J Real-Time Image Proc 18(3):889–900
74. NVIDIA Developer (2022) NVIDIA Jetson Nano. https://developer.nvidia.com/embedded/jetson-nano-developer-kit/
75. Veit A, Belongie S (2018) Convolutional networks with adaptive inference graphs. In: Proceedings of the European conference on computer vision (ECCV), pp 3–18
76. Fan A, Grave E, Joulin A (2020) Reducing transformer depth on demand with structured dropout. In: International conference on learning representations, pp 1–16