



A real-time video smoke detection algorithm based on Kalman filter and CNN

Alessio Gagliardi¹ · Francesco de Gioia¹ · Sergio Saponara¹

Received: 25 September 2020 / Accepted: 6 March 2021 / Published online: 19 March 2021
© The Author(s) 2021

Abstract

Smoke detection represents a critical task for avoiding large scale fire disaster in industrial environment and cities. Including intelligent video-based techniques in existing camera infrastructure enables faster response time if compared to traditional analog smoke detectors. In this work presents a hybrid approach to assess the rapid and precise identification of smoke in a video sequence. The algorithm combines a traditional feature detector based on Kalman filtering and motion detection, and a lightweight shallow convolutional neural network. This technique allows the automatic selection of specific regions of interest within the image by the generation of bounding boxes for gray colored moving objects. In the final step the convolutional neural network verifies the actual presence of smoke in the proposed regions of interest. The algorithm provides also an alarm generator that can trigger an alarm signal if the smoke is persistent in a time window of 3 s. The proposed technique has been compared to the state of the art methods available in literature by using several videos of public and non-public dataset showing an improvement in the metrics. Finally, we developed a portable solution for embedded systems and evaluated its performance for the Raspberry Pi 3 and the Nvidia Jetson Nano.

Keywords Video smoke detection · Kalman filter · Convolutional neural network · Deep learning · Raspberry Pi · Nvidia Jetson Nano

1 Introduction

Prevention of fire accidents is an important safety, economic and environmental issue that is constantly addressed in various research fields [1]. Fire protection and prevention systems are available in the majority of public buildings and they are ubiquitous in public and private transportation. Traditional smoke detector devices are able to identify the presence of smoke only in the close proximity of the source of emission, but they lack the ability to signal the presence non-local hazards. Moreover, such devices can be easily damaged by the smoke and high temperatures developed during a fire. To overcome these limitations, video-based fire detection

systems are currently commonly used mainly supported by new emerging image processing and computer vision techniques. These techniques enable cameras and closed-circuit television (CCTV) systems to be used for smoke and fire detection, thus providing remote coverage for wider areas. Vision-based smoke/fire sensors also provide faster reaction times compared to sensors based on photometry, thermal or chemical detection that instead require larger amount of fire/smoke to trigger. Additionally, vision-based detection algorithms can be easily included in existing surveillance systems and deployed in city streets, industrial buildings and in public transportation. Since these algorithms are generally developed for low-cost IoT embedded devices with networking capabilities, they can also be used to provide remote signalling procedures complete with useful information about the location and extension of the fire [2].

New high-performance hardware platforms, such as graphic processing units (GPUs) and general purpose processors (GPPs) with significant computing capabilities with high level of parallelism, have allowed the development of artificial intelligence techniques that have dramatically improved the state-of-the-art in object detection, visual

✉ Alessio Gagliardi
alessio.gagliardi@phd.unipi.it

Francesco de Gioia
francesco.degioia@phd.unipi.it

Sergio Saponara
sergio.saponara@iet.unipi.it

¹ Dip. Ingegneria dell'Informazione, University of Pisa, Via G. Caruso 16, 56122 Pisa, Italy

object recognition, in speech recognition, and many other domains [3].

While traditional video-based smoke detectors need pre-processing and feature extraction steps, such as colour and shape characteristics of smoke, recent deep learning algorithms allow for automatic data-driven feature extraction and classification from raw image streams. Thus, deep learning models may prove to be valid alternatives to traditional visual detectors.

In this paper we present a novel smoke detection pipeline specifically designed for embedded devices with high throughput and with small memory requirements. We compared our algorithm on a state-of-the-art dataset and other non-public videos. We also deployed our processing pipeline as a real-time video processing application on two single board embedded platforms, namely an Nvidia Jetson Nano and a Raspberry Pi 3. Hereafter, the paper is organized as follows: Sects. 1 and 2 deal with introduction and state of art video-based fire/smoke detectors. Section 3 presents the algorithm description and discusses the global Neural Network architecture used in the video processing chains. Section 4 shows the experiment results and discussion. Section 5 presents the algorithm implementation in the embedded systems evaluating their performance. Conclusions are reported in Sect. 6.

2 State of art video-based fire/smoke detectors

Smoke detection systems that use machine vision methods to classify frame sequences are mainly based on static information from single frames, or include dynamic characteristics of the smoke. This kind of smoke detection systems use texture, shape, color, movement, energy, and frequency, flutter or frequency spectrum, as in [4–6].

This class of algorithms may produce poor results when there is small chromatic difference between the background and fire/smoke pixels, potentially generating too many false alarms for the system to be useful in practice. Therefore, in [7], the problem of background estimation and segmentation is directly addressed. In [8] a smoke detector based on Kalman estimator, color analysis, image segmentation, blob labeling, geometrical features analysis and M of N decisor is used to produce an alarm signal within a strict real-time deadline.

New deep learning approaches can automate the feature extraction process thus making the process more effective in image classification and object detection [9]. Moreover, optimization techniques and model approximations can be used to allow the implementation of medium-size models on low-performance embedded devices, Thus allowing the system to be used in different domains.

Consequently, various deep learning methods have been proposed for fire and smoke detection. For example, R-CNN, YOLO and SSD networks are used in [10] in order to detect forest fire in real-time. A fire detection system based on pre-trained VGG16 and Resnet50 models is proposed in [11], whereas in [12] and in [13], a YOLO network [14] is used to perform fire detection and flame detection.

Although deep learning methods may achieve higher results in terms of accuracy, they tend to be more complex than traditional algorithms and may be unsuitable for low-memory embedded devices.

In this paper, we used a mixed approach based on Kalman estimation for background subtraction and a Convolutional Neural Network for region classification. In this work, we specifically targeted low-cost embedded platform for real-time video processing and selected a light-weight network model suitable for this class of devices.

3 Algorithm description

Figure 1 shows the logical flow of the proposed video smoke detection algorithm, which is based on motion detection, color segmentation, bounding boxes extraction, and a prediction from the convolutional neural network. The algorithm is designed as a chain of image and video processing tasks.

The sequence of elaborations starts with a new frame coming from the video camera, or a test set video, and ends with the computation of a smoke alarm signal. The tasks of motion-detection/color-segmentation and the tasks of CNN prediction can be parallelized in a typical implementation on computer platform or embedded systems. The rest of the functions, instead, will be calculated according to a sequential flow of the figure. Hereafter, we report in details the main video processing steps of the workflow of Fig. 1.

3.1 Motion detection

The Kalman filter is used in this work to estimate movement within a series of frames. The motion detection algorithm detects groups of pixels that change their value

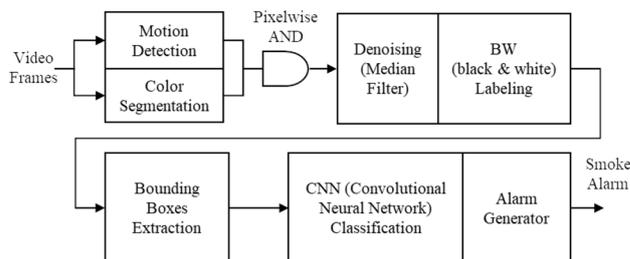


Fig. 1 Signal processing of smoke detection algorithm

over time. We allow the value of the pixel to evolve in time following a linear model. We use the Kalman filter to predict the expected value of a pixel based on its previous state history. If the difference between the predicted value and the actual value is larger than a threshold, a movement is detected and the pixel is marked accordingly. The background prediction is given by Eq. (1), where \widehat{BG}_k is the background prediction of the current frame I , \widetilde{BG}_{k-1} is the background estimation at the previous frame, and $a = A/(1 - \beta)$ is a weighting coefficient for the previous state of the pixel. We allow a to be dependent on the camera frame rate with the relation $\beta = 1/(1 + \tau_\beta \cdot fr)$, where fr is the frame rate in FPS of the processed video and τ_β is a time constant set to 10s, and A is a constant set to 0.618.

$$\begin{bmatrix} \widetilde{BG}_k \\ \widehat{BG}_k \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & a \end{bmatrix} \cdot \begin{bmatrix} \widetilde{BG}_{k-1} \\ \widehat{BG}_{k-1} \end{bmatrix} \quad (1)$$

The background estimation \widetilde{BG}_k of the frame I is obtained from Eq. (2), where \widehat{BG}_k in Eq. (1), and K_1 and K_2 are defined in Eq. (3).

$$\begin{bmatrix} \widetilde{BG}_k \\ \widehat{BG}_k \end{bmatrix} = \begin{bmatrix} \widetilde{BG}_k \\ \widehat{BG}_k \end{bmatrix} + \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} (I - [1 \ 0] \cdot \begin{bmatrix} \widetilde{BG}_k \\ \widehat{BG}_k \end{bmatrix}) \quad (2)$$

$$K_1 = K_2 = A \cdot FG_k + \beta \cdot (1 - FG_k) \quad (3)$$

$$FG_k = |I - \widetilde{BG}_{k-1}| \quad (4)$$

$$FG_k \geq THR_{foreg} \quad (5)$$

In the initialization phase, we set \widetilde{BG}_k equal to initial frame I and \widehat{BG}_k equal to zero. This initialization is done only when the first frame is received. According to 5, we select the pixel of the foreground FG_k if their value is higher than the threshold THR_{foreg} . In the above equations, FG_k is the foreground of the frame I , $A = 1/(1 + \tau_\alpha \cdot fr)$, where τ_α is a time constant set to 16 s. The empirical threshold THR_{foreg} is set to 0.08. Compared to [15] where authors used a fixed value for $a = 0.7$, we let the variable a depend on the value of the video frame rate. Specifically, $a = A/(1 - \beta)$ depends on β , which in turn depends on constants A and τ_β and on the frame rate fr . The coefficient $K_1 = K_2$ are derived to quickly absorb in the background the objects that are faster than the smoke (like moving people) and to filter out static objects that are much slower than the smoke. As reported in [8] the value of $A = 0.618$ has been found starting from the value of 0.7 proposed in [15] and refining it to maximize the detection performance.

3.2 Color segmentation and BW labeling

Color segmentation allows to select pixels in the shade of gray, which could potentially represent a smoky cloud. Hence the RGB color frames are converted in HSV scale, where H is the hue, S is the saturation, and V is the value. To select the smoky pixels, we use a saturation threshold THR_{sat} that we set to 0.2 having the input image values in the range [0, 1]. We select only those regions of the scene where $pixel_{sat} < THR_{sat}$. We choose this parameter because the smoke changes color according to the background, so a good way is to use the saturation channel. The threshold value was derived empirically by maximizing smoke detection in the video datasets described in Sect. 4. At this point the pixels of the motion detection mask and the pixels of the color segmentation mask pass through a block that performs the logical AND (pixel-wise). So we select all pixels that are moving and gray colored. A median filter is then applied to this matrix to remove all scattered non-aggregated pixels, as they represent noise within the image.

Agglomerated pixel regions are labeled according to a labeling function and will be supplied later as “blob” to the bounding box extraction function.

3.3 Bounding boxes extraction

The bounding boxes (BB) extraction selects the moving gray object in the scene. Such task is performed considering the total area of each selected blob that corresponds to the number of pixels of such region. Hence, we select the blobs for BB extraction that satisfy the condition of Eq. (6):

$$Area_{blob} > W \cdot H/250 \quad (6)$$

The area threshold is calculated so that for a VGA frame ($W = 640 \times H = 480$) the area is about 1.2 k pixels, which is a bit less than half of the input size of the convolutional neural network. Selection of blobs with area smaller than the minimum threshold may induce the network to produce erroneous predictions. The function returns four coordinates as output representing the position and size of the smallest box containing the region. We also introduced an extra function to check the ratio between the sides of the bounding boxes. This function accepts regions where the ratio between the sides is greater than or equal to 0.6. This step is especially important as it avoids wrong predictions from the neural network that expects square images as input. Images that are too narrow or too wide can lead to image distortion and therefore to errors, thus they are simply discarded.

The remaining images are scaled to the neural network input dimensions and fed to the network.

3.4 Convolutional neural network

In order to discriminate between the bounding boxes that contain smoke from those that do not, we developed a Convolutional Neural Network classifier. Our network accepts as inputs 50×50 pixel float RGB images with values in range $[0-1]$ and produces as output the prediction value for the input image as a single value in the range $[0-1]$. We report the CNN architecture in Fig. 2. We trained our network on the smoke dataset collected by Yuan et al. [16]. Such dataset contains 14,271 images labelled as “Smoke” or “Non Smoke”.

The training of such network is performed on 11,014 training images and 3257 test images, which represents the 70% and the 30% of the dataset respectively. The training is performed for 100 epochs with batch size 64 achieving 98% accuracy on test data. Since the images of the dataset have a size of 100×100 pixels we perform an image resize to match the input size of $50 \times 50 \times 3$ of the input layer before the training phase. We used binary cross entropy as loss function and Adam algorithm as optimizer. For reference, we performed the same training with Adamax as optimizer, obtaining similar results. Details of training parameters and optimizer are shown in Table 1.

The Fig. 3 instead shows the training process relative accuracy vs number of epochs for both training set and validation set, and the values of the loss function. We limited ourselves in using only convolutional layers, max pooling

Table 1 Training hyper-parameters for CNN

Parameter	Value
Optimizer	Adam
Learning rate	0.01
Beta1	0.9
Beta2	0.999
Epsilon	$1e-7$
Number of epochs	150
Batch size	64
Dropout value	0.5

layers and fully connected layers as these are the most supported layers in tensor processing units (TPUs) for on board AI acceleration. The network consists of 12 levels in total with only Convolution layers for feature extraction. We specifically addressed the problem of overfitting introducing a

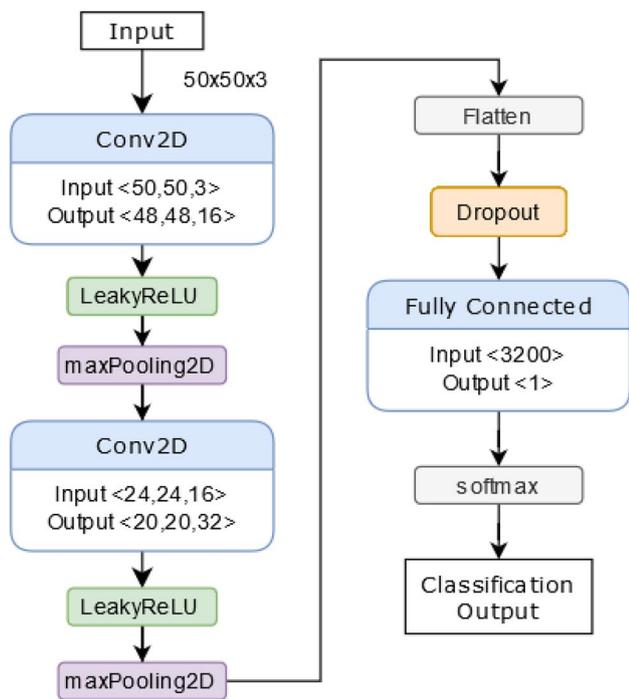


Fig. 2 Convolutional neural network architecture

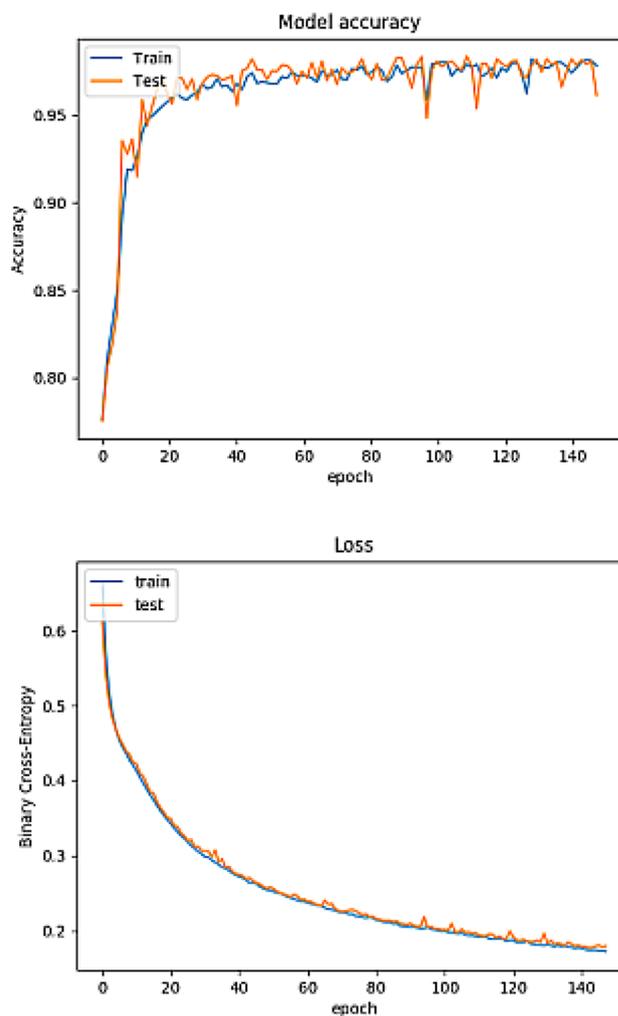


Fig. 3 (Top) training history for accuracy metric on training set and validation set. (Bottom) loss function history on training set and validation set

dropout layer with high (0.5) dropout value. Moreover, to improve training convergence, we used LeakyReLU instead of plain ReLU obtaining marginal training speedup. We intentionally designed a relatively flat CNN as we aimed at low computational power and high processing speed. Given the simplicity of its architecture, the network may sometimes misclassify ambiguous images, however we are able to handle spurious in-frame misclassifications and remove them in a post-processing filtering step. Indeed, we can accept only the region where the score of the prediction is higher than 0.7.

This way we are sure to select only bounding boxes that almost certainly contain smoke and discard everything else.

3.5 Alarm generator

Every bounding box promoted as possible area containing smoke are classified as “smoke” or “non smoke” as output of the Neural Network already presented. To activate the smoke alarm, we want that the classification of “smoke” persists for some certain amount of time, to be sure to do not identify false alarm. For each frame we must have at least one bounding box classified as “smoke” for at least one 1.5 s over a 3 s window.

If these considerations are valid, then an alarm signal will be generated. These time window thresholds can be changed at will in relation to how fast the detection is intended to be performed. Threshold values that are too low can lead to false alarms, whereas high threshold values may hinder the ability to detect fire and exceedingly increase the system reaction time.

These values have been obtained empirically to achieve excellent detection performance and also to comply with the EN50155 [17] standard for onboard train fire-alarm systems.

4 Experimental results

The aim of this work is to present a technique that can be easily deployed to an embedded device in order to finally build a smoke detection unit. For doing so, it becomes inevitable to use a dataset that contains videos and images with smoke–fire emergencies from the real world. For the training phase of the neural network we used the Yuan et al. dataset [16] that contains smoke and non-smoke images captured manually by cameras or from internet.

Figure 4 shows that the used smoke images have a variety of texture, shades and shapes while non-smoke images contain different scenes and objects like cars, plants, people and buildings.

As a result of the training phase we obtained a model of just 126KB requiring low memory space and suitable to be contained in low cost embedded platforms. We can compare



Fig. 4 (Top) Non-smoke images example of the dataset. (Bottom) Smoke images example from dataset

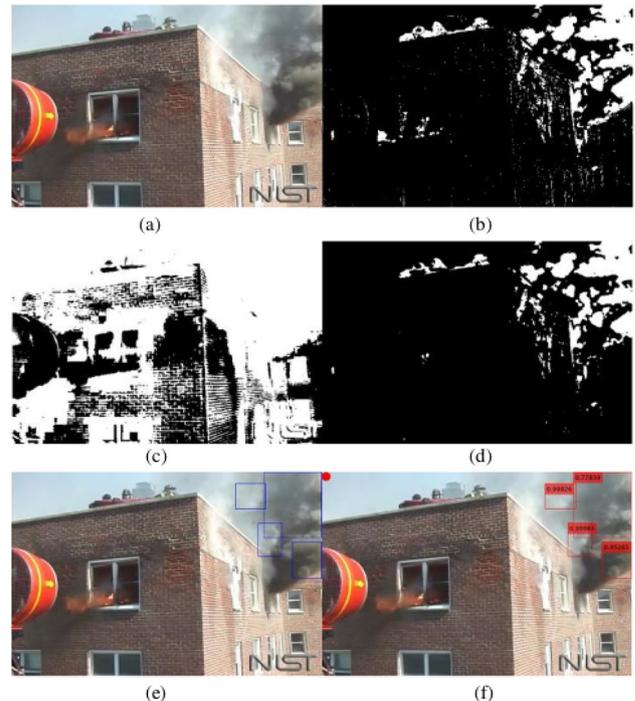


Fig. 5 Detection of smoke on a test video. **a** Current Frame n ; **b** motion detection; **c** color segmentation; **d** logic AND pixelwise and median filter; **e** bounding box extractor; **f** final result at runtime with bounding boxes and accuracy predicted as smoke from the neural network

our neural network with the network designed in [18] on the same image dataset. Although our network consists of only 12 layers compared to the 25 in [18], we obtained only a minor drop in accuracy decreasing from 99% accuracy obtained in [18] to 98% accuracy perhaps we used less trainable parameter.

The neural network model represents only a small piece of the whole processing chain which is instead depicted in Fig. 5. The figure provides an example application of a smart city antifire system showing the output mask of each processing step already described in Sect. 3.

The images of Fig. 5 are obtained by processing a certain number of frames (about 30 frames), to allow Kalman

filter to accurately estimate the moving parts of the scene. In Fig. 5d we notice that the firefighters on top of the roof are also selected in the mask. However, this region is discarded at the next step as shown in Fig. 5e. In the last figure (Fig. 5f) we see the Nth frame in which the bounding boxes are correctly classified as smoke by the neural network and drown into the frame with their prediction scores.

To measure the detection performance of the algorithm we used the metrics reported in Eqs. (7)–(11) where the TP represents the true positive, TN the true negative, FP the false positive and FN the false negative. We define the video as positive if smoke is present, otherwise, if smoke is absent, we define the video as negative. If the algorithm identifies the smoke in a positive video, then we mark the result as true positive (TP). If the algorithm does not identify the smoke in a positive video, then we mark the result as false negative (FN). If the algorithm identifies the smoke in a negative video, then we mark the result as false positive (FP). If the algorithm does not identify the smoke in a negative video, then we mark the result as a true negative (TN).

$$\text{Hit rate} = \text{TP}/(\text{TP} + \text{FN}) \quad (7)$$

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{FN} + \text{TN} + \text{FP}) \quad (8)$$

$$\text{F1 score} = 2 \cdot \text{TP}/(2 \cdot \text{TP} + \text{FP} + \text{FN}) \quad (9)$$

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \quad (10)$$

$$\text{MCC} = \frac{(\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN})}{\sqrt{(\text{TP} + \text{FP}) \cdot (\text{TP} + \text{FN}) \cdot (\text{TN} + \text{FP}) \cdot (\text{TN} + \text{FN})}} \quad (11)$$

The Table 2 shows the detection performance obtained by using two different dataset. The first one is the Firesense [19] public dataset developed within the FP7-ENV-244088 project, and contains videos for testing flame and smoke detection algorithms. The second dataset is provided by [8] for a total 42 videos of which 30 are smoke videos while 12 are of non-smoke videos. Our algorithm has excellent performance in almost all metrics in the first column of Table 2

Table 2 Detection performance for two different dataset

	Firesense [19]	AdViSED [8]
Hit rate	1.0000	0.9049
Precision	0.8667	0.8636
Accuracy	0.9091	0.8438
F1 score	0.9286	0.8837
MCC	0.8210	0.6467
False positives	0.2221	0.2726
False negatives	0.0000	0.0951

for the Firesense dataset. This is certainly attributed to the higher specificity of such video dataset. Indeed, the dataset [8] contains also videos in which the smoke is very far from the scene, so it is more difficult to detect it with the algorithm presented. This would explain a Hit Rate of about 0.9 which corresponds to 2 FN for this dataset, as the algorithm is not able to detect smoke slowly moving and far from the camera frame especially in bad video quality. Nevertheless, the global detection performance remains stable even in both datasets. Regarding the results of the Firesense dataset we observe a Hit Rate of 1 and consequently zero FN. In the same dataset we obtain a precision of about 0.87 due to 2 false positives. However, in this security scenario it is always preferable to have an algorithm that prevents the risk of fire obtaining slightly higher false alarm, but lowering the chances to miss a detection which would be dangerous for human life and the environment.

This technique has been compared with the AdViSED algorithm and its dataset. The results for all metrics are very similar for the two techniques and are shown in Table 3. The difference concerns the presence of 2 FN that, as already presented, are due to bad quality videos and in which the smoke is caught far from the scene. The difference between the two techniques lies in the automatic feature extraction by CNN which it makes easier for the developer to calibrate the algorithm as it is for AdViSED.

In fact, in AdViSED the parameters such as Area, Extent, Eccentricity etc. must be adjusted by hand and this can presents a difficulty in some specific cases. The CNN instead has the task to extract itself the characteristics and no calibration is required if the training phase was successful. Another important parameter is about the speed of the algorithm in detecting the smoke, recognizing it as danger element into the scene caught by the camera. Therefore, in the Table 4 we compare the proposed techniques respect to the state of the art algorithm [8, 20, 21] using the same test video shown in Fig. 6. Such comparison consists of the calculation of the delay, in terms of number of frames, to necessary reveal the presence of smoke.

We achieve as good results as those in [8] for both video n.1 and video n.2. Compared to the latter, we obtain a much better result for video n.3, since our neural network seems to be more robust to the noise generated from the metal

Table 3 Detection Performance respect AdViSED algorithm

	Our algorithm	AdViSED [8]
Hit rate	0.9049	1.0000
Precision	0.8636	0.8890
Accuracy	0.8438	0.8500
F1 score	0.8837	0.8750
MCC	0.6467	0.7035

Table 4 Detection delay in number of frames vs state of the art

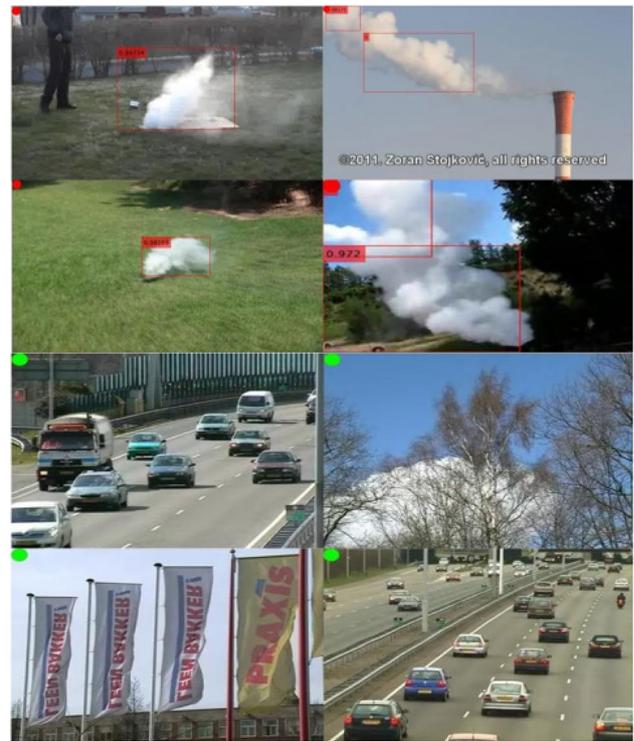
Video in Fig. 6	Duration (in frames)	Delay in smoke detection (in frames)			
		Yu [20]	Toreyin [21]	AdViSED [8]	Our Algorithm
n. 1	900	86	98	9	11
n. 2	244	121	127	19	21
n. 3	630	118	132	120	39

**Fig. 6** Video n.1, video n.2, video n.3 (from left to right)

fence placed in front of the smoke. Note that, although our algorithm is able to detect smoke in a few frames, it waits for the time-threshold of 1.5 s already defined in Sect. 3.5 before triggering an alarm. However, this signal allows us to be about $\times 15$ faster than the EN50155-standardized techniques, which rather have to react in a range of about 60 s. In Fig. 7 we show the output of other test videos as example of a final application of our algorithm. Such challenging dataset contains indoors and outdoors scenes for custom antifire application for Intelligent Transportation System and Smart Cities. On top of Fig. 6 are depicted smoke scenes with fire alarm activated and bounding boxes containing the detected smoke. At the bottom of Fig. 6 are shown videos containing moving objects, clouds and scenarios that could mislead the algorithm. However, no bounding boxes have been drawn in these cases and so the algorithm works properly. Indeed, the alarm is generated when the red circle appears on the top left of the image otherwise the circle displayed is green.

5 Embedded system implementation

We developed a Python [22] implementation from an initial Matlab [23] code using Keras 2.3.0 [24], Numpy 1.18.0 [25], Scikit-image 0.17.2 [26] and openCV 4.3.0 [27]. Such Python implementation has been carried out perhaps in order to have no difference with respect to the initial Matlab source code and in order to obtain the same metrics already described in Sect. 4. We selected two development boards for such implementation as they are ubiquitous low-cost embedded platforms with reasonably low power consumption. We selected an Nvidia Jetson Nano and a Raspberry Pi 3 (RPi 3) considering their ease of use and their popularity on the market. As comparison we also performed a test on

**Fig. 7** (Top) smoke video test with bounding boxes and smoke alarm activated. (Bottom) neutral test video with smoke alarm not activated as expected

a PC equipped with Intel Core i3-4170 CPU, 8GB DDR3 RAM and Windows 10 OS running the same algorithm.

5.1 Nvidia Jetson nano

Jetson Nano is a powerful but compact embedded computer with the low cost of approximately \$100 [28]. The board is equipped with a Quad-core ARM A57 at 1.43 GHz CPU, a 128-core Maxwell GPU, 4 GB 64-bit LPDDR4 25.6 GB/s of RAM, and multiple port for connecting peripherals such as 4x USB 3.0, CSI (Camera Serial Interface) connectors etc. We used as testing camera a Raspberry Pi Camera Module V2 with 8M pixel. This camera can work with a camera resolution of 1080p@30 frames per second and is connected on the CSI port. The official operating system for the Jetson Nano is called Linux4Tegra, which is actually a version of Ubuntu 18.04 that's designed to run on Nvidia's hardware.

5.2 Raspberry Pi 3

Raspberry Pi is small, powerful, and low-cost embedded device at about 30\$ per unit. It is a perfect platform for applications like distributed and networked measuring nodes in smart city or intelligent transport systems scenarios. The board is equipped with a Broadcom BCM2837, a System on

Chip (SoC) including a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with a cache L2 of 512 KB and 1GB of DDR2 RAM, Video Core IV GPU, 4 USB 2.0 ports, onboard WiFi @2.4 GHz 802.11n, Bluetooth 4.1 Low Energy, 40 GPIO pins, and many other features [29]. It runs on Raspbian OS, a Debian-based Linux distribution for download [30].

5.3 Test performance

We initially measured the average frame per seconds (fps) for different platform using the same smoke video but changing the pixels resolution. In particular, we tested our algorithm in two cases: 320×240 p and 640×480 p since those are the most used for CCTV systems. We can easily observe in Table 5 that the best performance is achieved on the Jetson Nano board reaching almost 55 fps on a camera resolution of 320×240 and about 18 fps by doubling the size of the input image. This is mainly due to the Nvidia Maxwell GPU integrated on Jetson Nano and therefore specialized in highly parallel floating point computations especially those dealing with neural network tensors. The Raspberry Pi 3 board reached almost 10 fps and 3 fps using the first and the second camera resolution respectively. Our PC instead obtains performance comparable to that of Raspberry Pi. This can be explained because such PC machine does not have a dedicated GPU for accelerating tensor operations, therefore its performance is worse than that of the Jetson Nano board. The Jetson Nano board has been used according to the MAXN power profile with maximum consumption to 10 W, so that the operating system can use the GPU as required.

Performance on each board was also measured in terms of resource usage such as CPU, GPU, Memory and operating temperature. The measurements were carried out for both embedded devices without any peripherals connected and controlled using a static IP address and remote desktop. We monitored 60 min (1 h), of which the first 30 min are in idle state (only background task are running), and in the last 30 the application is running.

In Fig. 8 we reported the CPU usage for RPi 3 and Jetson Nano boards, and the GPU usage for Jetson Nano. The CPU is around 0–3% of usage for both board while there is a jump to 28% of CPU for RPi 3 and 33% for Jetson Nano.

Table 5 Performance in terms of frames per seconds tested on different boards

	GPP \times 64 (fps)	Jetson Nano (fps)	Rasp- berry Pi 3 (fps)
320×240	11.18	54.82	9.37
640×480	6.46	17.49	3.01

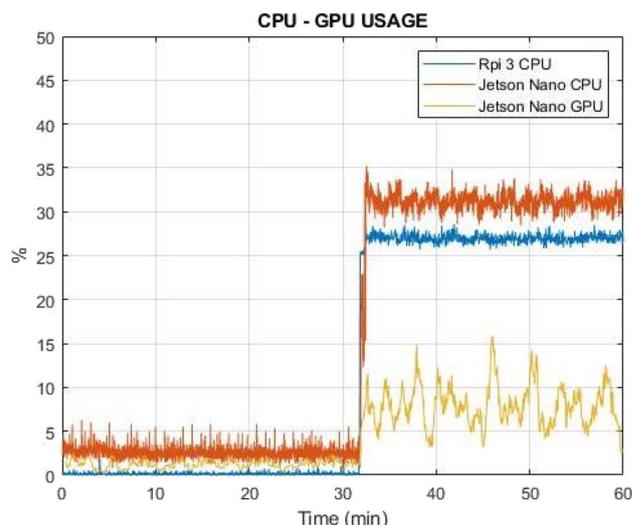


Fig. 8 CPU GPU usage vs execution time

We witnessed more intensive use of the GPU at about 30 min, although it remains around 10% usage. This is mainly due to the small size of the network and therefore not computationally expensive. The memory usage, shown in Fig. 9, increased instead from about 25–40% for RPi 3 and 70% on Jetson Nano board. As reported in [31], temperature can drastically impact the performance of the inference system. We performed an experiment measuring the temperature when the CPU is idle and when the application is running. Thermal results are shown in Fig. 10. We obtained better operating temperature results on the Jetson reaching a maximum temperature of about 49 °C.

The RPi 3 instead reaches the maximum temperature of 69 °C that is below the official operating temperature limit

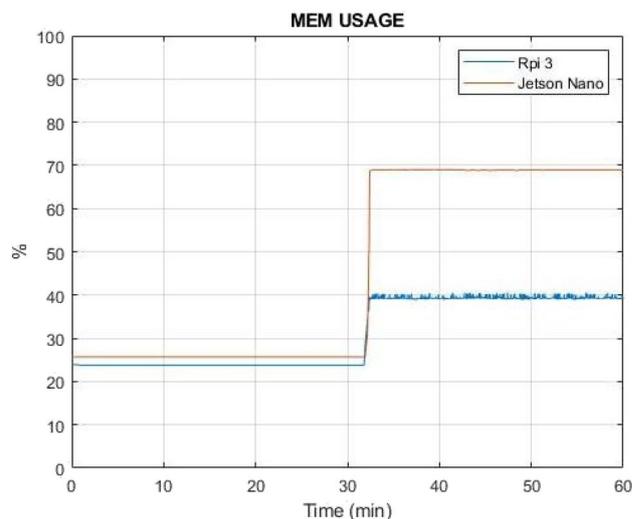


Fig. 9 Memory usage vs execution time

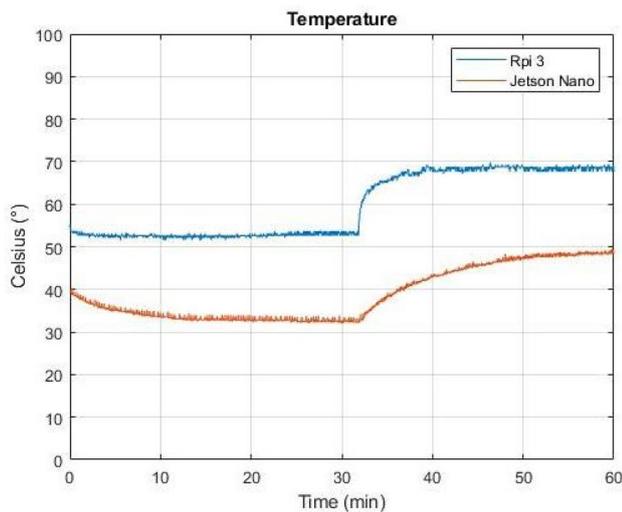


Fig. 10 Operating temperature vs execution time

Table 6 Comparison of different state of the art object detection networks

	Jetson Nano (fps)	Raspberry Pi 3 (fps)	Weights
Proposed [320 × 240]	54.82	9.37	126 KB
Tiny YOLO V3 [416 × 416]	25	0.5	44.9 MB
SSD Mobilenet-V2 [300 × 300]	39	1.0	17.9 MB
SSD Resnet18 [300 × 300]	18	DNR	47.0 MB

of 85 °C, and below the 82 °C of thermal throttle. The measurements are affected by an offset ambient temperature of about 30 °C. The RPi also does not have a heat sink comparable to that of the Jetson, so it heats more easily. We measured also the power consumption in Watts for each board. The total amount of power consumption for Raspberry Pi 3 is around 1.5 W in idle and 2.4 W while the application is running. For the Jetson Nano we measured instead 2.7 W in idle state and 7 W as average running the algorithm.

As final comparison we benchmarked our algorithm respect to the state of the arts object detection artificial neural network, comparing their inference performance and the total memory size in MB or KB. Such benchmarks are partially available from [32] and shown in Table 6.

The purpose of this comparison is to show how an already pre-built network can have much lower performance than a custom built one. Clearly these benchmarks are obtained without making a new training on a specific dataset containing smoke images but we would expect the same outputs. The best results are obtained by our algorithm both in terms of processing speed and in

terms of total amount of memory space occupied. The SSD Mobilenet-V2 [33] network reaches around 40 fps on Jetson Nano and about 1 fps on RPi 3 with about 18 MB of memory occupied on disk. Both the Tiny YOLO V3 [34] and SSD Resnet18 [35] occupy more than 44MB of disk memory and have poor results on RPi 3. The Tiny YOLO V3 barely reaches 1 fps while Resnet18 did not run (DNR) on RPi3. Note that DNR results occurred due to limited memory capacity, unsupported network layers, or hardware/software limitations.

In conclusion, the Nvidia Jetson Nano can be considered as the final platform for the implementation of a full stack IoT antifire application. In fact, while the Raspberry Pi 3 costs about half as much as its competitor, and gets better result in power consumption, at the same time Jetson gets the best results in processing and elaboration in real time. Although power consumption is slightly higher than in RPi 3, it is reasonable to consider it low and suitable for the applications already mentioned.

6 Conclusions

This paper proposed a hybrid technique of smoke detection consisting of both traditional Image Processing and Artificial Intelligence methods. The goal is the development of an object detection algorithm specialized in locating the smoke blobs within a video stream to prevent fire accident.

During the processing, our algorithm selects the regions of interest based on the smoke features while the CNN is used to classify them. The CNN was designed to be light-weight in order to save memory space, and fast in order to accelerate the inference step.

We validated our system on different datasets obtaining good metrics in terms of accuracy and inference time. The results suggest that the system presented in this paper is suitable for the future development of smart IoT devices based on Raspberry Pi or Nvidia Jetson Nano. In particular, on the Nvidia Jetson Nano the system can operate at 55 fps at a resolution of 320 × 240 pixels and be included easily in common CCTV cameras.

The proposed solution can be employed in various anti-fire applications as it achieves low-latency detection when compared to traditional ceiling-mounted smoke detectors.

Acknowledgements This work is partially supported by the H2020 European Processor Initiative project n. 826647 and by the Dipartimento di Eccellenza Crosslab Project by MIUR.

Funding Open access funding provided by Università di Pisa within the CRUI-CARE Agreement.

Declaration

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Hall, J.: The total cost of fire in the united states. National Fire Protection Association, vol. 01 (2014)
- Gagliardi, A., Saponara, S.: Distributed Video Antifire Surveillance System Based on IoT Embedded Computing Nodes, vol. 03, pp. 405–411. Springer, Berlin (2019)
- LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(05), 436–444 (2015)
- Saponara, S., Pilato, L., Fanucci, L.: Early video smoke detection system to improve fire protection in rolling stocks. In: Kehtarnavaz, N., Carlsohn, M.F. (eds.) *Real-Time Image and Video Processing 2014*, vol. 9139, pp. 14–22. International Society for Optics and Photonics, SPIE (2014)
- Çelik, T., Özkaramanli, H., Demirel, H.: Fire and smoke detection without sensors: image processing based approach. In: 2007 15th European Signal Processing Conference, pp. 1794–1798 (2007)
- Rafiee, A., Dianat, R., Jamshidi, M., Tavakoli, R., Abbaspour, S.: Fire and smoke detection using wavelet analysis and disorder characteristics. In: In 2011 3rd International Conference on Computer Research and Development, vol. 3, pp. 262–265. IEEE (2011)
- Vijayalakshmi, S.R., Muruganand, S.: Smoke detection in video images using background subtraction method for early fire alarm system. In: 2017 2nd International Conference on Communication and Electronics Systems (ICCES), pp. 167–171 (2017)
- Gagliardi, A., Saponara, S.: Advised: Advanced video smoke detection for real-time measurements in antifire indoor and outdoor systems. *Energies* **13**(8) (2020). <https://www.mdpi.com/1996-1073/13/8/2098#cite>
- Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. *Neural Inf Process Syst* **25**, 01 (2012)
- Wu, S., Zhang, L.: Using popular object detection methods for real time forest fire detection. In: 2018 11th International Symposium on Computational Intelligence and Design (ISCID), vol. 01, pp. 280–284. IEEE (2018)
- Sharma, J., Granmo, O.-C., Goodwin, M., Fidge, J.T.: Deep convolutional neural networks for fire detection in images. In: *International Conference on Engineering Applications of Neural Networks*, pp. 183–193. Springer (2017)
- Lestari, D.P., Kosasih, R., Handhika, T., Murni, Sari, I., Fahrurrozi, A.: Fire hotspots detection system on cctv videos using you only look once (yolo) method and tiny yolo model for high buildings evacuation. In: 2019 2nd International Conference of Computer and Informatics Engineering (IC2IE), pp. 87–92. IEEE (2019)
- Shen, D., Chen, X., Nguyen, M., Yan, W.Q.: Flame detection using deep learning. In: 2018 4th International Conference on Control, Automation and Robotics (ICCAR), pp. 416–420. IEEE (2018)
- Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788 (2016)
- Ridder, C., Munkelt, O., Kirchner, H.: Adaptive background estimation and foreground detection using kalman-filtering. In: *Proceedings of international conference on recent advances in mechatronics*, pp. 193–199. Citeseer (1995)
- Yuan, F., Shi, J., Xia, X., Fang, Y., Fang, Z., Mei, T.: High-order local ternary patterns with locality preserving projection for smoke detection and image classification. *Inf. Sci.* **372**, 225–240 (2016)
- European standard applies to all electronic equipment for control, regulation, protection, diagnostic, energy supply, etc. installed on rail vehicles. <https://shop.bsigroup.com/ProductDetail/?pid=00000000030282634>. Accessed 14 July 2020
- Tao, C., Zhang, J., Wang, P.: Smoke detection based on deep convolutional neural networks. In: 2016 International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), pp. 150–153. IEEE (2016)
- Dimitropoulos, K., Barmoutis, P., Grammalidis, N.: Spatio-temporal flame modeling and dynamic texture analysis for automatic video-based fire detection. *IEEE Trans. Circuits Syst. Video Technol.* **25**(2), 339–351 (2015)
- Yu, C., Mei, Z., Zhang, X.: A real-time video fire flame and smoke detection algorithm. *Procedia Eng.* **62**, 891–898 (2013)
- Toreyin, B.U., Dedeoglu, Y., Cetin, A.E.: Contour based smoke detection in video using wavelets. In: 2006 14th European Signal Processing Conference, pp. 1–5. IEEE (2006)
- Python programming language. <https://www.python.org/>. Accessed 14 July 2020.
- The MathWorks Inc.: Matlab 9.7.0.1190202 (r2019b) (2018)
- F. Chollet et al.: Keras. <https://keras.io> (2015). Accessed 14 July 2020.
- van der Walt, S., Colbert, S.C., Varoquaux, G.: The numpy array: a structure for efficient numerical computation. *Comput. Sci. Eng.* **13**(2), 22–30 (2011)
- van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T.A.: Scikit-image: image processing in python. *PeerJ* **2**, e453 (2014)
- Bradski, G.: The opencv library (2020). <https://opencv.org/>. Accessed 16 Mar 2020
- Jetson nano developer kit. <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>. Accessed 14 July 2020.
- Raspberry pi foundation. <https://www.raspberrypi.org/>. Accessed 14 July 2020.
- Rpi camera module v.1.3. <https://www.raspberrypi.org/documentation/hardware/camera/>. Accessed 14 July 2020.
- Benoit-Cattin, T., Velasco-Montero, D., Fernández-Berni, J.: Impact of thermal throttling on long-term visual inference in a CPU-based edge device. *Electronics* **9**(12), 2106 (2020)
- Jetson nano: Deep learning inference benchmarks. <https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks>. Accessed 07 Sept 2020.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520 (2018)
- Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7263–7271 (2017)

35. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Alessio Gagliardi received in 2018 the M.Sc. in Robotics and Automation Engineering at University of Pisa, Italy. The master thesis concerned the modelling and analysis of the V2x 802.11p protocol by developing simulation software to study the reliability of vehicles communication in different operating scenarios. He worked in 2016 for Texas Instrument as Application and Support Engineer in Freising, Germany. He is currently pursuing a PhD degree at the Department of Information Engineering focusing his research on Image Processing, Artificial Intelligence, and Embedded Systems for Automotive and Industrial Applications.

Francesco de Gioia received in 2018 the M.Sc. in Embedded Computing Systems with a thesis on software development for RISC-V architecture. He worked for 1 year at IngeniArs, a spin-off company

of University of Pisa as firmware developer for custom IP. In 2018 he enrolled in the Ph.D. program at University of Pisa. His current research activities involve the development of custom firmware and Linux drivers; and the design of AI and Machine Learning systems for embedded devices.

Sergio Saponara received the master's (cum laude) and the Ph.D. degrees in electronic engineering from the University of Pisa, Pisa, Italy. He was a Marie Curie Fellow at IMEC, B, Leuven, Belgium. He is currently a Full Professor of Electronics with the University of Pisa. He is currently the Director of the Summer School "Enabling Technologies for IoT" and the CTO of Ingeniars srl, Pisa. He has co-authored about 300 scientific papers and holds 17 patents. Dr. Saponara is the AE of Canadian Journal of Electrical and Computer Engineering, IEEE Consumer Electronics Magazine, Springer Journal of Real-Time Image Processing, and IET Electronics Letters. He has been a Technical Program Committee member of more than 100 international IEEE and SPIE conferences.