Fire Detection based on a Two-Dimensional Convolutional Neural Network and Temporal Analysis

Pedro Vinícius A. B. de Venâncio*†, Tamires M. Rezende*†, Adriano C. Lisboa*‡, Adriano V. Barbosa†§
*Gaia, solutions on demand, Technological Park of Belo Horizonte, Belo Horizonte, Brazil
†Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
‡Department of Computer Engineering, Federal Center for Technological Education of Minas Gerais, Belo Horizonte, Brazil
§Department of Electronics Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
*†{pedrovinicius, tamiresrezende}@ufmg.br, *‡adriano.lisboa@gaiasd.com, †§adrianovilela@ufmg.br

2021 IEEE Latin American Conference on Computational Intelligence (LA-CCI) | 978-1-7281-8864-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/LA-CCI48322.2021.9769824

***Abstract*—In the last few years there has been a substantial increase in the success of deep learning, especially with regard to convolutional neural networks for computer vision tasks. These architectures are being widely used in emergency situations, where a fast and accurate response is needed. In environmental monitoring, several works have focused on fire detection, since fires have been increasingly associated with negative consequences such as respiratory diseases, economical losses and the destruction of natural resources. The automatic detection of smoke and fire, however, poses a particularly difficult challenge to computer vision systems, since the variability in the shape, color and texture of these objects makes the process of learning how to detect them much more complicated than for other ordinary objects. As a consequence, the number of false positives may grow high, which is especially problematic for a real-time application that mobilizes human efforts to fight fire. This work presents a robust fire detection tool based on a 2D deep convolutional network capable of suppressing false alarms from clouds, fogs, car lights and other objects that are easily confused with fire and smoke. Our approach integrates an object detector with an object tracker; this makes it possible to analyze the temporal behavior of the object and use that information in the decision process. We also present D-Fire, a public and labeled dataset containing more than 21,000 images, which is used to train and test the proposed system. The experimental results show that the detector reached an mAP@0.50 = 75.91% and that the incorporation of the temporal context resulted in a 60% reduction in the false positive rate at the cost of a 2.86% reduction in true positive rate. In addition, the computational cost added by the proposed approach to the fire detector is negligible, so that real-time detection is still completely feasible.**

***Index Terms*—Two-dimensional convolutional neural network, fire detection, temporal analysis.**

I. Introduction

Since prehistory, humans have used fire to protect themselves, to prepare food and to light environments. As humans improved their ability to control fire, several other functions were assigned to it for their own benefit, e.g, using fire as an energy source or even to prepare the soil for agriculture and livestock. However, the irresponsible use of fire can have irreversible consequences for nature and for the well-being of the local population. Most wildfires worldwide have direct or indirect anthropogenic causes, whether intended or accidental.

According to the National Interagency Fire Center (NIFC), more than 4.16 million hectares of land in the United States were burned in forest fires in 2020, 40% of which in California [1]. Data from Brazil's National Institute for Space Research (INPE) show a very similar situation in South America in 2020 and also in 2021, where there have been more than 32,000 cases in the first 3 months of the year [2]. From this perspective, a more rigorous monitoring of environmental areas becomes necessary, especially if we consider that human observation, a time-consuming and low-efficiency strategy, is one of the most used methods for fire detection [3].

Since the effective surveillance of large swaths of land requires a lot of manpower, systems based on smoke and temperature sensors became relevant to support early fire detection. In order to obtain a high accuracy, a large number of sensors must be properly distributed throughout the environment [4]. However, this sensor network design has several limitations. Some areas can be difficult to access in order to install sensors, and the cost of infrastructure can be quite high.

Alternatively, computer vision-based approaches have been widely applied to this problem. Cameras have the hability to scan large areas in short time spans and can be integrated with a local processing device to run automatic fire detection applications. The first techniques used for this purpose used deterministic rules for identifying fire and smoke from their visual aspects. Several color spaces have been investigated, including RGB [5], CIE Lab [6] and HSI [7]. Subsequently, motion rules were also incorporated into color-based techniques to improve the accuracy of the systems [8] and some studies even used these manually extracted features to train machine learning algorithms [9].

Although more advanced deep learning architectures, such as convolutional neural networks (CNNs), have emerged and substantially improved performance in automatic fire detection [3] [10], identifying the presence of fire or smoke remains a non-trivial task. These are abstract objects that can assume various shapes and colors, which makes the model's task of adapting to new situations quite hard. Thus, the system can set off false alarms (false positives) or even miss real events (false negatives). In order to avoid these issues, a representative and balanced database is essential. This special labeled dataset provides useful information for supervised learning and, consequently, for building a model that can detect fires reliably. However, even with a high accuracy, there is a concern about false positives, since they can generate

recurring alarms to environmental protection agencies and fire departments without any need.

This work proposes a robust fire detection system, where an object tracker is incorporated into an object detector, implemented as a two-dimensional convolutional neural network, with the goal of veryfing if the detections exhibit fire or smoke behavior over time. In addition, we present D-Fire [11], a public and labeled image dataset developed by our research group containing more than 21,000 images, where 11,000 are of real fire situations. These images were used to train two networks, a faster one and a more efficient one, to detect fire and smoke, and their results were compared. Then, the impact of the temporal analysis on both networks is validated on videos from surveillance cameras and from the Internet. We show that it is possible to obtain a significant reduction in the false positive rate at an insignificant computational cost.

This paper is organized as follows. Section II describes the proposed fire detection method. The D-Fire dataset is presented in Section III. Section IV reports the training procedure of the networks and the results obtained for the full system in real situations. Finally, Section V concludes our research with future work possibilities.

II. Proposed Approach

The proposed fire detection system consists of two tasks performed sequentially: (i) detection and (ii) tracking. The first task aims to identify the occurrences of fire and/or smoke in the scene. For this purpose, an object detector based on a 2D CNN is used. The tracking task, in turn, has a supplementary role. It takes the detections made in the previous step and tries to verify if the detected object is in fact fire or smoke based on its temporal behavior along the incoming frames. If the event is confirmed, an alarm is triggered, as show in Figure 1. The integration of the detector with the tracker allows not only to identify fires with greater reliability, but also to determine the fire proportions and, consequently, to estimate the efforts required to fight it.

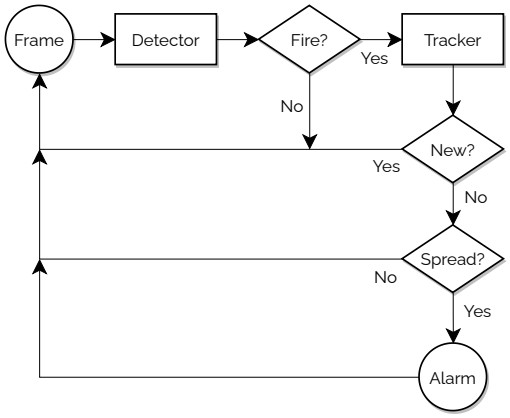

Figure 1. Proposed fire detection system. The fire validation process undergoes multiple confirmation steps in order to reduce false alarms.

A. Fire Detection

YOLO (You Only Look Once) is arguably one of the best single-stage object detection algorithms currently available. It was originally proposed by Joseph Redmon in 2016 [12] and later improved in three other versions, the most recent one (YOLOv4) in 2020 by Alexey Bochkovskiy [13]. As suggested by its name, YOLO uses a single deep convolutional neural network to simultaneously predict class probabilities and bounding boxes directly from images in a single evaluation.

The YOLO algorithm resizes the input image to $N \times N$ pixels and then splits it into a grid of $S \times S$ non-overlapping cells. Each of the S^2 cells is associated with C class probabilities and B bounding boxes, with a confidence score assigned to each bounding box. Since our problem has $C = 2$ classes of interest (smoke and fire), each cell i is represented by:

$$\boldsymbol{c_i} = \left[c_{i1}, c_{i2}, \boldsymbol{b_{i1}}, ..., \boldsymbol{b_{ij}}, ..., \boldsymbol{b_{iB}}\right] \tag{1}$$

with the j-th bounding box of the i-th cell represented by $b_{ij} = [p_{ij}, x_{ij}, y_{ij}, h_{ij}, w_{ij}]$, where p_{ij} is the probability that bounding box j contains an object, x_{ij} and y_{ij} are the coordinates of the center of the bounding box, h_{ij} and w_{ij} are the height and width of the bounding box, respectively, and c_{i1} and c_{i2} are the probabilities that the object detected in cell i belongs to class 1 (smoke) and 2 (fire), respectively. A grid cell will be responsible for detecting an object if the center of the object falls into that cell [12].

The confidence score is given by $s_{ij} = p_{ij} \times \text{IoU}^{\text{truth}}_{\text{pred}}$, where $\text{IoU}^{\text{truth}}_{\text{pred}}$ is the Intersection over Union (IoU) between the predicted box and the ground truth. During the test, the confidence score and class probabilities are multiplied in order to have class-specific confidence scores for each box. An object is considered to be present in the bounding box if its confidence score is above a given confidence threshold t_{conf}. Since each bounding box can contain only a single object, it will be assigned to the class with the highest class-specific confidence score. However, if the number of cells S^2 is large, an object may span multiple neighboring cells, which will all then be assigned to the same object. In order to avoid this problem, Non-Maximum Suppression technique is used to eliminate boxes whose areas overlap above a threshold t_{IoU}, keeping only the bounding box with the highest $\text{IoU}^{\text{truth}}_{\text{pred}}$ [12].

The network architecture of YOLOv4 consists of three parts: (i) CSPDarknet53 as a backbone, which is a 2D convolutional network pre-trained on ImageNet dataset [14], (ii) Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PAN) as a neck, which are layers generally used to collect feature maps from different levels and (iii) YOLOv3 as a head, which is used to predict, simultaneously, classes and bounding boxes [13]. In total, the YOLOv4 network has 110 convolutional layers, where 107 are followed by a cross-iteration batch normalization and a non-linear activation (of which 35 are Leaky ReLU and 72 are Mish).

This quite robust architecture comes at a cost: each forward propagation pass of the YOLOv4 network on a single 416×416 pixels RGB image requires 59.57 billion floating

point operations (BFLOPs). Since such a high computational cost usually requires the use of Graphics Processing Units (GPUs) for training and detection, the YOLOv4 authors also proposed Tiny YOLOv4, a simplified and, therefore, faster version of YOLO [13]. In order to better suit mobile devices and machines with limited computational power, Tiny YOLOv4 requires only 6.789 BFLOPs in a forward propagation pass on a single 416×416 pixels RGB image, about 88.6% less than YOLOv4. However, this speed comes at the cost of reduced precision. Here, we analyze both architectures for fire detection, discussing the particularities of each one.

B. Temporal Analysis

The proposed tracking algorithm (the *Tracker* block in Figure 1) consists of a multi-step process based on the centroids of detected fire and smoke objects. When such objects are identified by the detector in a video frame, the bounding boxes associated with the detected objects are passed on to the tracker as inputs. The tracker then uses the coordinates of each bounding box to calculate its central point, or centroid.

Centroids are computed for every frame where at least one of the classes is detected. If the detected objects passed to the tracker correspond to the first events since the initialization of the system, a unique identifier is assigned to each occurrence. Otherwise, the objects detected in the current frame Q_t are associated with those that are already being tracked in the previous frame Q_{t-1}. This process is illustrated in Figure 2.

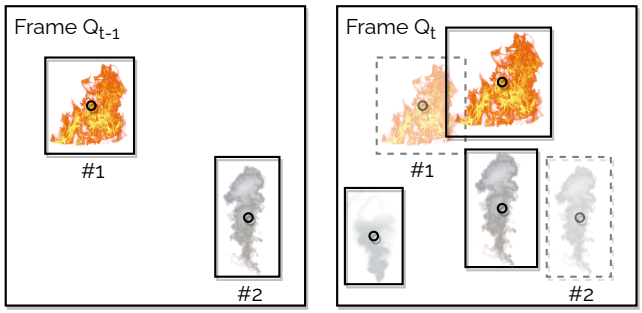

Figure 2. Left: a fire and a smoke are identified in frame Q_{t-1} and labeled $\#1$ and $\#2$, respectively. The centroids of the corresponding bounding boxes are computed. Right: new bounding boxes (solid lines) are identified in frame Q_t. The centroids computed in frame Q_{t-1} are also shown (dashed lines).

Next, the Euclidean distances between the centroids of corresponding objects in frames Q_t and Q_{t-1} are computed. Finally, each object in frame Q_t is associated with the object in frame Q_{t-1} whose centroid is closest. This corresponds to choosing the minimum movement for the fire and smoke objects across consecutive frames. In the cases where there are more detected objects in frame Q_t than in frame Q_{t-1}, i.e., there are more detections than events being tracked, a unique identifier is assigned to the object that was not associated with any object in the previous frame (see Figure 3). On the other hand, if an object disappears from view for a total of m consecutive frames, its record is removed and its numeric identifier is made available for future assignments.

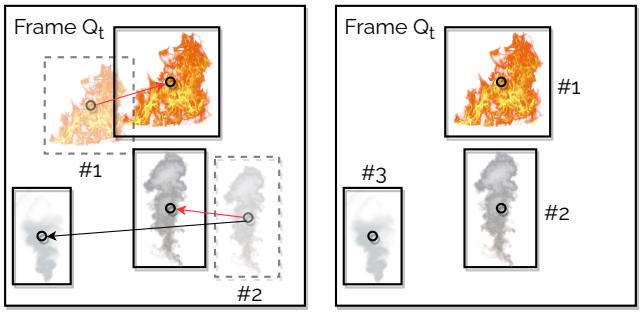

Figure 3. Left: finding correspondences between objects in consecutive frames. Objects in frame Q_t (solid lines) are associated with objects in frame Q_{t-1} (dashed lines) by calculating the Euclidean distances of $\#1$ and $\#2$ to all objects of the same class. The objects' positions in frame Q_t are determined from the shortest distances (red arrows). Right: the object whose centroid has no equivalent is recorded as a new event $\#3$.

The main purpose of the tracker is to determine whether the temporal behavior of the detected objects match that of fire or smoke in their initial stages. This is done by monitoring the size (area) of the detected bounding boxes over time. If the areas grow above a certain threshold t_{area} over a time span of w frames, the fire or smoke detection is confirmed and an alarm is activated.

The measure used to quantify the temporal growth rate of the detected objects is the Coefficient of Variation, which is computed, over all frames in the current time window, as the ratio between the standard deviation and the mean of the detected objects' areas. This strategy allows to minimize the number of false alarms, since the main agents behind the false detections are static or do not show a growth trend over time, such as solar reflections, raindrops and car lights.

III. Dataset

In order to train and test the proposed fire detection system, both an image data base and a video data base are necessary. The former is used to train both two-dimensional deep detection networks, whereas the latter is used to test the efficiency of the proposed temporal analysis.

As an image data base we use D-Fire [11], a dataset we have created specifically for use in all our fire detection experiments based on 2D CNNs. D-Fire contains 21,527 labeled images. Of these, 1,164 contain only fire, 5,867 contain only smoke, 4,658 contain both fire and smoke, and 9,838 contain negative examples (i.e., neither fire nor smoke). Most images have more than one occurrence of a given class. In total, there are 11,865 bounding boxes labeled as smoke and 14,692 labeled as fire.

D-Fire images were capture mainly from the Internet, but also collected from fire tests and surveillance cameras, and synthesized digitally. The dataset is quite diverse in terms of the smoke type (color, shape, texture, density), the fire origin (forest, sea, urban), the camera's view point and confounding elements (clouds, shiny objects).

In order to build a video data base, it was necessary to conduct an exhaustive search on the Internet, since several works use a small number of videos to validate their methodologies [15] [16]. We collected 100 videos, most of which were taken

from environmental surveillance cameras of the "Apaga o Fogo!" (Put out the Fire!) website [17], where our proposed system is running in production. All detections are notified by the website, which sends along a short video of the event so that a human can confirm if the event has been correctly (true positive) or incorrectly (false positive) detected by the system. Half of the 100 collected videos correspond to true positives, whereas the other half correspond to false positives (these contain elements that pose several difficulties to the detector, such as building lights, fog, dust and raindrops). The videos are also available in a public repository[1].

IV. Experimental Results

For training both networks, we used the Mini-batch Gradient Descent as an optimization algorithm over 30,000 iterations with 64 mini-batches and 16 subdivisions. The learning rate was defined at 0.005 for Tiny YOLOv4 and 0.001 for YOLOv4 by the Grid Search method [18], both with reductions in 24,000 and 27,000 iterations by a factor of 0.1. In order to avoid overfitting, the convergence criterion used was the early stopping. The weight decay was set to 0.0005, momentum 0.9, burn-in 1000, image size $N = 416$, grid size $S = {}^{N}/_{32} = 13$ and $B = 3$ bounding boxes for each grid cell [12]. The weights of the network layers were initialized through the transfer learning in the ImageNet dataset [14]. The whole training procedure was performed with the Darknet framework [19] and conducted using a Intel(R) Xeon(R) Gold 6140 of 18 cores and 36 threads operating at 2.30GHz, with 256GB of RAM, RAID 5 with 4 SSDs of 1TB and NVIDIA Quadro P5000 graphics card with 2560 CUDA cores and 16GB of GDDR5X, running Ubuntu 18.04.3 LTS and CUDA compilation tools, release 10.1, V10.1.243.

We split the D-Fire dataset into two independent and identically distributed sets: (i) 80% for training (17,221 images), where data augmentation techniques (e.g, saturation, exposure, hue and mosaic [13]) were randomly applied during the training, and (ii) 20% for testing (4,306 images). Then, we employed 5-fold cross validation to the training set to determine the best confidence thresholds (t_{conf}) for the Tiny YOLOv4 and YOLOv4 networks, where each model was trained on 13,776 images and evaluated on 3,445 images for ten threshold values evenly distributed between 0 and 0.9. The best confidence threshold for each network was defined as the value that, on average over the five models, maximizes the F_1-score, as illustrated in Figure 4. This metric is the harmonic mean of precision and recall. Since for both networks (Tiny YOLOv4 and YOLOv4) there was a tie between two values of t_{conf} that maximized the F_1-score, we used the average of the IoU over all classes, what we call avg IoU, to decide on the best t_{conf} value. As avg IoU increases, the confidence threshold also increases. Thus, the values that maximize this metric were chosen as the best thresholds, i.e., 0.3 for Tiny YOLOv4 (avg IoU = 52.64%) and 0.4 for YOLOv4 (avg IoU = 61.04%).

[1]https://shorturl.at/hvKN6

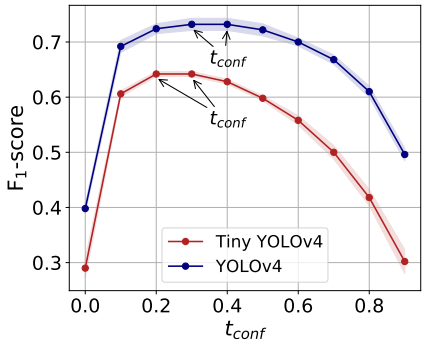

Figure 4. The best average confidence thresholds obtained in the 5-fold cross validation for Tiny YOLOv4 were 0.2 and 0.3 (F_1-score = 0.642) and for YOLOv4 were 0.3 and 0.4 (F_1-score = 0.732). The transparent areas around the curve of each network specify the upper and lower limits based on the respective standard deviations of the F_1-score.

After all network hyperparameters have been chosen, we trained each network on the entire training set (17,221 images) and we evaluated them on the test set (4,306 images), each with its best confidence threshold (0.3 for Tiny YOLOv4 and 0.4 for YOLOv4). The results are shown in Table I which, besides the F_1-score and the avg IoU, also shows the Average Precision (AP), which is given by the area under the precision-recall curve. The AP measure provides a value for each class that is independent of the confidence threshold, i.e., the average of the precision for recall values over all confidence thresholds. The mean value of the AP over all classes is known as the mean Average Precision (mAP). In this work, we use the mAP computed from $t_{\text{IoU}} = 0.50$, referred to as mAP@0.50.

Table I
Performance metrics on the test set.

Network	F_1-Score	avg IoU	AP_{smoke}	AP_{fire}	mAP
Tiny YOLOv4	0.64	52.63%	67.93%	57.36%	62.65%
YOLOv4	0.73	61.06%	82.36%	69.45%	75.91%

From Table I, we see that YOLOv4 performs better than Tiny YOLOv4 on all metrics, which is expected. It is interesting to note that both networks have greater difficulty in detecting fire than smoke; this happens because fire occurrences are usually quite smaller in size. YOLOv4's better performance comes at the cost of higher training and inference computational loads. Training took approximately 8 hours for the Tiny YOLOv4 network and about 50 hours for the YOLOv4 network. Regarding computational cost for inference, YOLOv4 processes 47 fps (frames per second) whereas Tiny YOLOv4 processes 342 fps on our machine. Figures 5 and 6 show some successful detections on the test set, whereas Figures 7 and 8 show some limitations. The values in each bounding box correspond to the network's confidence score on detection.

The area suppression threshold t_{area}, whose purpose is to define a plausible variation limit on the detection area to validate a fire, greatly affects the performance of the temporal

Figure 5. Detection of fire and smoke in a night environment by the YOLOv4 network. Tiny YOLOv4 was also able to identify the occurrences, albeit with slightly lower confidence scores. Note that in addition to low lighting, another challenge is the artificial lights scattered around the scene, which can be confused with saturated fire.

Figure 6. Early detection is very relevant to avoid as much damage as possible. In this example, the YOLOv4 network detects smoke far from the camera at an early stage. Tiny YOLOv4, in contrast, did not detect it.

Figure 7. During sunrise and sunset, some sunlight reflections can affect the cameras and, consequently, hinder detections. This false positive example, which was mistaken for smoke, was detected only by Tiny YOLOv4.

analysis. Very large values of this hyperparameter can suppress many real detections, since greater variations in areas will be expected. In contrast, very small values may not be sufficient to suppress the necessary false positives. In order to tune t_{area}, we use 30% of the collected videos, chosen randomly (15% with false positives and 15% with true positives). The frame window size is set to $w = 20$ frames. In addition, as the videos are about 30 seconds long, we used a number of consecutive frames large enough, i.e., $m = 1{,}000$, that no fire occurrences

Figure 8. False positive generated by both networks. Raindrops are a big challenge to fire detections systems. Depending on their position on the camera lens, they can look very similar to smoke.

were removed from the records during the video.

The choice of the best area suppression thresholds was based on two metrics: (i) true positive rate (TPR), which indicates the proportion of correct detections (i.e., the system detected an occurrence that did happen), and (ii) false positive rate (FPR), which indicates the proportion of incorrect detections (i.e., the system detected an occurrence that did not happen). Therefore, the best threshold will always be the lowest possible threshold, i.e., the threshold that least suppresses detections, but still provides both a high TPR and a low FPR. The Receiver Operating Characteristic (ROC) curve, shown in Figure 9, allows visualizing the influence of area suppression threshold as a function of these rates in a single visual representation.

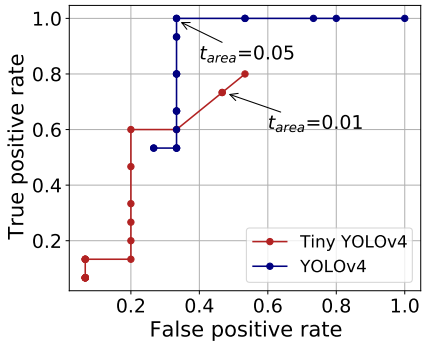

Figure 9. Because all videos in the data base come from detections in the "Put out the Fire!" website, which is running an YOLOv4 detector, the standalone YOLOv4 detector (i.e., without the tracker) has a TPR = 1 and a FPR = 1. As the area suppression thresholds increases, the FPR drops a lot until it reaches about 0.33, with no decrease in the TPR. Thus, the lowest threshold that reaches high TPR and low FPR simultaneously is $t_{\text{area}} = 0.05$. However, Tiny YOLOv4 is very sensitive to the area suppression threshold. A very low value, in spite of decreasing the FPR, also decreases the TPR, which made us choose $t_{\text{area}} = 0.01$, the lowest possible value.

The remaining 70% of the videos were used to estimate the impact of the temporal analysis on the detections made by the trained networks. Table II shows the TPR and the FPR before and after the introduction of the temporal analysis (TA). Also shown is the average delay (number of additional frames) introduced by the temporal analysis.

Table II
IMPACT OF THE TEMPORAL ANALYSIS ON FIRE DETECTION.

Network	TPR	FPR	Delay
Tiny YOLOv4	82.86%	62.86%	-
Tiny YOLOv4 + TA	77.14%	45.71%	2.20
YOLOv4	100.00%	100.00%	-
YOLOv4 + TA	97.14%	40.00%	7.17

From Table II we see that, for both networks, the temporal analysis resulted in a reduction in the false positive rate at the cost of a quite smaller reduction in the true positive rate. This is especially true in the case of the YOLOv4 network, where the reduction in the FPR was quite higher (60.00 p.p) and the reduction in the TPR was quite smaller (2.86 p.p), which is very desirable. On the other hand, Tiny YOLOv4, a less accurate network, experienced a smaller reduction in the FPR (17.15 p.p) at the cost of a greater reduction in the TPR (5.72 p.p). However, suppressing detections from Tiny YOLOv4 may not be very advantageous, as it significantly affects its correct detections, which are already reduced as a result of its architecture with fewer layers. Finally, the detection delay resulting from the post-processing associated with our temporal analysis, i.e., the number of additional frames needed to validate the detections, is quite small for both networks. On average, this implies a delay of 152.55 ms for YOLOv4 and 6.43 ms for Tiny YOLOv4, which does not prevent real time detection in either case.

V. CONCLUSIONS

This work presented a robust fire detection tool based on the YOLO object detection algorithm. Since the learning-based approaches commonly used in the literature tend to have high false positive rates, we incorporate a second stage (a tracker) to the system to confirm detections based on their temporal behavior. This is done by imposing a constraint on the movement of the detections based on the assumption that fires start small and expand over time. This allows the suppression of false positives corresponding to detections which are static or do not follow this pattern. We also presented D-Fire, a data base specifically created for use with fire detection algorithms.

Initially, in order to investigate the trade-off between performance and speed, two topologies were tested: YOLOv4, focused on performance, and Tiny YOLOv4, focused on speed. YOLOv4 resulted in mAP@0.50 = 75.91% on the test set, whereas Tiny YOLOv4 resulted in mAP@0.50 = 62.65%. Next, in order to analyze the influence of the temporal analysis mechanism on the detection rates, both networks were re-evaluated after the incorporation of the tracking stage. This resulted in a reduction in the FPR of 60.00 p.p for YOLOv4 and 17.15 p.p for Tiny YOLOv4. However, this also reduced the TPR in 2.86 p.p and in 5.72 p.p for the YOLOv4 and the Tiny YOLOv4, respectively. In addition, the introduction of the temporal analysis resulted in a delay in the order of milliseconds, which does not preclude real-time detection.

As future work, we plan to keep updating the D-Fire dataset, contributing to research in fire detection. We also plan to create more elaborate models for the movement of bounding boxes by treating them as time series in order to extract meaningful statistics about fire and smoke. We believe this could further reduce FPR without significant impacts on the TPR and the computational cost.

VI. ACKNOWLEDGMENTS

Financial support for this work was provided by CEMIG-ANEEL (R&D project D0619), by CNPq to Adriano Chaves Lisboa (Grant 304506/2020-6) and Tamires Martins Rezende (Grant 167016/2017-2), by FAPEMIG to Adriano Vilela Barbosa (Grant APQ-03701-16), and by CAPES.

REFERENCES

[1] K. Hoover and L. A. Hanson, "Wildfire Statistics," *Congressional Research Service*, 2021, updated January 4, 2021. [Online]. Available: https://crsreports.congress.gov/product/pdf/IF/IF10244

[2] National Institute for Space Research (INPE), "Queimadas Program," http://queimadas.dgi.inpe.br/queimadas/portal-static/estatisticas_paises/, 2021, accessed April 2021.

[3] S. Wu and L. Zhang, "Using popular object detection methods for real time forest fire detection," in *IEEE 11th International Symposium on Computational Intelligence and Design*, vol. 1, 2018, pp. 280–284.

[4] M. Ajith and M. Martínez-Ramón, "Unsupervised segmentation of fire and smoke from infra-red videos," *IEEE Access*, vol. 7, pp. 182 381–182 394, 2019.

[5] T.-H. Chen, P.-H. Wu, and Y.-C. Chiou, "An early fire-detection method based on image processing," in *International Conference on Image Processing (ICIP'04)*, vol. 3. IEEE, 2004, pp. 1707–1710.

[6] T. Celik, "Fast and efficient method for fire detection using image processing," *ETRI journal*, vol. 32, no. 6, pp. 881–890, 2010.

[7] W.-B. Horng, J.-W. Peng, and C.-Y. Chen, "A new image-based real-time flame detection method using color analysis," in *IEEE Networking, Sensing and Control*. IEEE, 2005, pp. 100–105.

[8] T. Celik, H. Demirel, H. Ozkaramanli, and M. Uyguroglu, "Fire detection using statistical color model in video sequences," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 176–185, 2007.

[9] J. Zhao, Z. Zhang, S. Han, C. Qu, Z. Yuan, and D. Zhang, "Svm based forest fire detection using static and dynamic features," *Computer Science and Information Systems*, vol. 8, no. 3, pp. 821–841, 2011.

[10] S. Frizzi, R. Kaabi, M. Bouchouicha, J.-M. Ginoux, E. Moreau, and F. Fnaiech, "Convolutional neural network for video fire and smoke detection," in *42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2016, pp. 877–882.

[11] Gaia, solutions on demand, "D-Fire: an image data set for fire detection," https://git.io/JONna, 2018, downloaded in February, 2021.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[13] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv preprint arXiv:2004.10934*, 2020.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 248–255.

[15] T. Celik and H. Demirel, "Fire detection in video sequences using a generic color model," *Fire Safety Journal*, vol. 44, no. 2, pp. 147–158, 2009.

[16] D. B. Chakrabortya, V. Detania, and S. P. Jigneshkumar, "Fire Threat Detection From Videos with Q-Rough Sets," *arXiv preprint arXiv:2101.08459*, 2021.

[17] CEMIG, UFMG, Gaia, RaroLabs and UFVJM, "Apaga o Fogo!" https://apagaofogo.eco.br/, 2020.

[18] P. V. A. B. Venâncio, "Hyperparameter tuning of Convolutional Neural Networks implemented in the Darknet framework," https://git.io/JLpE8, 2021.

[19] Redmon, J. and Bochkovskiy, A., "Darknet: Open Source Neural Networks in C," https://git.io/JTICL, 2013, downloaded in January, 2021.