Energy Efficient Fire Detection System

Tianyi Chen, Yuanxi Li
e-mail: chen.tianyi1@northeastern.edu, li.yuanxi1@northeastern.edu

Abstract –

Canada has suffered from forest fires over recent years. Immense damage was caused leaving large area of forestry land burned. The early detection of such fires can prevent huge amount of losses. In this report, we present a computer vision based novel method that can detect fires at their early stages. Our approach combines traditional machine learning and deep learning models that is accurate while being energy efficient. It can be easily deployed on low cost computing devices like Raspberry Pi, etc. Methodology (Section II) will introduce the algorithms and models we have used before we explain the workflow of our approach in Section III. We have done experiments (in Section IV) with results and analysis. Conclusion (Section V) follows.

Keywords: Wildfire, Computer Vision, Machine Learning

I. INTRODUCTION

Wildfires, also known as forest fires, are a natural hazard in any forested and grassland region in Canada. The place where we are located, British Columbia, ranks among the highest wildfire occurrence regions in Canada. Wildfires usually occur from May to September and can cause extensive damage and put lives in danger. Both human and natural factors can cause a wildfire. Heat, oxygen and fuel - known as the fire triangle - play an important role in the occurrences and spreadings of a wildfire. Eliminating one or more of above mentioned elements can help to slow down and even put out fires. [1]

In Canada, thousands of forest fires burned millions of hectares of forest areas over the years [2]. Massive damages were caused. Early detection of fire can significantly shorten the reaction time and also reduce the potential damage as well as the cost of fire fighting. Our objective is to detect the fire as fast as possible with early notification to the fire units. [3]

Currently, Natural Resources Canada (NRCan) rely mainly on satellite imagery to detect when wildfires occur. [4] There are some companies who have developed some tools to detect occurrences of wildfires using computer vision technology.

Looking back into history, ever since the inception of CV, researchers had been trying to apply CV techniques to prevent fires. Early efforts focus on specific scenarios such as tunnels, aircraft hangars, ships, etc. [5] Later on researchers tried to figure out some generic solutions. Healey et al. [6] propose to identify fire using only color clues. Based on the fact that fire flickers with a certain range of frequency, [7] add an analysis of temporal variations of fire to improve detection performance. However, such methods relies on handcrafted features and are inferior to deep learning based methods. Khan et al. [8] purposed a fire detector based on CNN with a performance better than all previous traditional methods. A future version [9] focuses on efficiency of the CNN model was also proposed. Fire detectors based on object detection models are also purposed such as YOLOv3 [10] based [11] and Faster R-CNN [12] based [13].

Previous works mainly focus on performance of the detector. Even though efficiency was focused sometimes, whether it's reflected in real world application is questionable. According to [11], their simplified detector could only run at roughly 3 fps on a UAV, which implies high energy consumption and short battery life. Even on platforms that don't care about battery life, computing power might still be a limitation. In our paper, we propose a fire detector which focuses on energy efficiency while still being accurate. We'll describe our methodology in section II and how the system works in section III.

II. METHODOLOGY

We shall achieve energy efficiency through a mixed deep-learning and traditional CV approach. Such an algorithm is intended to have the best of both worlds. Namely, it shall be energy efficient as traditional methods while being accurate as deep learning methods. We'll define the components in this section and combine the components to elaborate the algorithm in the next section.

A. Background Modeling

Since we are working with fixed cameras in untraveled places, we're expected to have a stationary background in most of the frames. For color modeling, it's important to subtract the background pixels and extract only the foreground pixels as a way of data augmentation. The chosen background modeling is similar to some previous works such as [14] and [15]. We'll first introduce the modeling technique which is followed by how we deal with permanent changes in the background.

There are a few sophisticated background subtraction techniques such as mixture of Gaussian model (MOG). For simplicity, we'll adopt the following one. Assuming the distributions of color channels of each pixel are independent. For each pixel in a channel, its distribution is estimated as:

$$p_i(I_i(X)) = \frac{1}{\sqrt{2\pi}\sigma_i(x)} \exp\left(\frac{(I_1(x)-\mu_i(x))^2}{2\sigma_i^2(x)}\right), i \in \{R,G,B\} \tag{1}$$

where i is a color channel in $\{R,G,B\}$, $I_i(x)$ is the pixel value at spatial location x in i^{th} color channel. $p_i(I_i(x))$ is an approximation for probability density of $I_i(x)$ and p_R, p_G, p_B are assumed to be independent of each other. $\mu_i(x)$ is the mean value of $I_i(x)$, $\sigma_i(x)$ is the standard deviation of $I_i(x)$.

Given N frames for training, $\mu_i(x)$ and $\sigma_i(x)$ could be estimated as follow:

$$\mu_i(x) = \frac{1}{N}\sum_{t=1}^{N} I_i^t(x) \tag{2}$$

$$\sigma_i(x) = \underset{t=1\rightarrow N-1}{argmax} |I_i^{t+1} - I_i^t| \tag{3}$$

where t is the time between frame $1 \rightarrow N$.

Using the model parameter in (2) and (3), a binary change map which indicate which pixels have been changed could be created:

$$CM(x) = \begin{cases} 1\ if\ \left(\sum_{i=\{R,G,B\}} B_i(x)\right) \geq 2 \\ 0\ otherwise \end{cases} \tag{4}$$

$$B_i(x) = \begin{cases} 1\ if\ |\mu_i(x) - I_i(x)| \geq \alpha_i \sigma_i(x) \\ 0\ otherwise \end{cases} \tag{5}$$

$CM(x)$ is the calculated change map. $B_i(x)$ is the change in color channels, and α_u is a hyper parameter which affects the change map. In our definition, if changes take place in more than 1 channel, we'll consider this pixel as being changed.

Since the lighting condition is likely to be changed with time, the model parameters are required to be adapted. Here we update the parameter by a simple formula:

$$\begin{aligned} \mu_i^{t+1}(x) &= \beta_i \mu_i^t(x) + (1-\beta_i) I_i^t(x) \\ \sigma_i^{t+1}(x) &= \beta_i \sigma_i^t(x) + (1-\beta_i)|I_i^t(x) - \mu_i^t(x)| \end{aligned} \tag{6}$$

where μ_i^t, σ_i^t are mean values and standard deviations at time t respectively. β_i is a constant for parameter adaptation for color channel i.

B. Statistical Color Model

The fire in the foreground could be detected by a set of simple yet robust formula:

$$R(x) > R_{mean} \tag{7}$$

$$R(x) > G(x) > B(x) \tag{8}$$

$$R_{mean} = \frac{1}{K}\sum_{i=1}^{K} R(x_i) \tag{9}$$

$$\begin{aligned} 0.25 \leq G(x)/(R(x)+1) \leq 0.65 \\ 0.25 \leq B(x)/(R(x)+1) \leq 0.45 \\ 0.25 \leq B(x)/(G(x)+1) \leq 0.60 \end{aligned} \tag{10}$$

where $R(x), G(x), B(x)$ are Red, Green, Blue values of pixel x, K is the total number of pixels in the image, R_{mean} is the mean of unchanged pixels in Red channel.

C. Deep Fire Detection Model

We choose YOLO as the deep-learning-based fire detector. Although the YOLO model evolved from version 1 to version 5, some basic ideas remain unchanged, which we'll detail in the following paragraphs.

1) Grid: Unlike two-stage detectors like R-CNN, Fast R-CNN, and Faster R-CNN, YOLO only needs one pass for each image. The mechanism behind that is the grids. The image is divided into an $S \times S$ grid, and we assign each object in the image to the grid where its center lies, i.e., this grid is responsible for describing this object's existence, category, and location (represented by a bounding box).

2) Output: The model is learned to predict 3 properties for each grid.

1. The coordinates t_x, t_y, t_w, t_h of the bounding box for the target object. The top left of the bounding box can be converted into an offset from the top left corner of the image.

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \end{aligned} \tag{11}$$

The real size of the bounding box b_w, b_h can be converted from t_w, t_h by formula (12):

$$\begin{aligned} b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{12}$$

where σ is the sigmoid function, c_x, c_y are the top left coordinates, p_w, p_h are the predefined width and height for this bounding box. It's also called an "anchor box".

2. The confidence score P of whether an object centered in this grid as well as how accurate the bounding box is. This is represented by one element in the output vector.
3. The category C of this object. Suppose we have N categories to classify, C will be an N-dim vector. The dimension with the largest value will be considered as the predicted category.

Since there might be multiple objects "belonging" to the same grid, YOLO will make multiple predictions for one grid. In YOLOv3 and later versions, there will be 3 predictions for each grid, and 3 sets of grids at different precision scales. Each prediction has a predefined anchor box, i.e., predefined height and width of the bounding box. The YOLO anchor box is a mechanism learned from Faster R-CNN's region proposal network (RPN). The difference is, YOLO discrads the manual setting apporach and use a K-means clustering to select the proper anchor boxes. The outputs t_w and t_h are used to apply to the corresponding anchor box to adjust it as close to the object as possible.

In our settings, there will be two categories to detect: smoke and fire. The size of output vector for each grid will be $L = 3 \times (4+1+2) = 21$, where 3 represents "3 predictions", 4 represents the coordinates and size of the bounding box, 2 represents "2 categories".

3) Backbone: The backbone in YOLO has been improved in every version. Although in v4 there are many changes compared to v3, the basic structure especially the way the output is generated remains the same. Therefore, the backbone Darknet-53 of YOLOv3 is illustrated for demonstration.

Suppose the input image has a shape of $416 \times 416 \times 3$, the backbone will output the prediction at 3 scales. Each output will be grids of vectors of length L representing 3 predictions. The meaning of which is discussed in previous sections.

Fig. 1. Darknet-53

	Layer	Filter	Strides	Output	
	Conv	3 × 3 × 32	1	416 × 416 × 32	
	Conv	3 × 3 × 64	2	208 × 208 × 64	
	Conv	1 × 1 × 32	1		
1 ×	Conv	3 × 3 × 32	1		
	Res			208 × 208 × 64	
	Conv	3 × 3 × 128	2	104 × 104 × 128	
	Conv	1 × 1 × 64	1		
2 ×	Conv	3 × 3 × 128	1		
	Res			104 × 104 × 128	
	Conv	3 × 3 × 256	2	52 × 52 × 256	
	Conv	1 × 1 × 128	1		
8 ×	Conv	3 × 3 × 256	1		
	Res			52 × 52 × 256	—Conv1×1×L—> 52 × 52 × L
	Conv	3 × 3 × 512	2	26 × 26 × 512	
	Conv	1 × 1 × 256	1		
8 ×	Conv	3 × 3 × 512	1		
	Res			26 × 26 × 512	—Conv1×1×L—> 26 × 26 × L
	Conv	3 × 3 × 1024	2	13 × 13 × 1024	
	Conv	1 × 1 × 512	1		
4 ×	Conv	3 × 3 × 1024	1		
	Res			13 × 13 × 1024	—Conv1×1×L—>13 × 13 × L

YOLOv3 and subsequent versions eliminate fully connected layers. Therefore, the model can accept any image with a height and width being odd multiples of 16.

III. COMBINING THE BIG AND THE LITTLE

Since we've defined the building blocks of the system. We are able to detail the overall detection system.

A. The "Little" Detector

The requirement for the "Little" detector is fast and robust. It doesn't require much data to train yet still apply to most scenarios with a reasonable accuracy. The shape of flames are highly volatile which is mainly affected by the surrounding atmospheric conditions including air, temperature, wind, etc. It's also affected by the burning material. As such, it's not possible to use a handcrafted shape-based classifier just like the related-works in face recognition. The only predictable property is the unpredictability. Thus, the detector will filter out those fire-like pixels, and compare its shape to the previous frame and check if it's changed. If such a change is detected in consecutive frames, then a fire is recognized.

Another issue is the fire-like object that might appear in the video, e.g., sun, some lights. This is addressed by two techniques:

1. The shape of the sun/light is not likely to change, which doesn't satisfy the definition of fire in our system.
2. After a few frames, they will become the background which won't appear as potential flames. That's why the "Little" detector needs a background subtraction step.

The algorithm of the "Little" detector is formulated as:

1. Apply background subtraction
2. For the remaining pixels, check if they satisfy (7), (8) and (10)
3. Detect blobs
    (a) Evaluate spatial mean of each detected blob
    (b) Evaluate spatial area of each detected blob
    (c) Mark a guard area (area enclosing the blob) over detected blob
4. If detected blobs' mean and spatial areas change in N consecutive frames in guard area, then a fire is detected.

In the third step the blobs are detected using connected component labeling algorithm. The guard area is a rectangular area that covers each blob and used to observe the behavior of enclosed blob in consecutive frames in order to determine whether it is fire object or not. In each guard area two measures are carried out; the first one is the spatial mean of the blob in guard area which is used to measure the identity of fire which should be changing because the fire has property of swinging, the second measure is spatial area of detected pixels in guard area, which should be either getting larger or smaller in consecutive frames.

B. The "Big" Detector

The YOLO architecture will be adjusted to fit our scenario. Apparently, we won't need the 53-layer version, nor will we need grids at 3 scales. The full size version would be an overkill for this task, and it will be computationally impossible to run on entry level hardware.

The preparation of the model is a bit more complicated than the "Little" detector which barely need any training. The YOLO model doesn't detect fire based on the causal relationship found in a video sequence. It instead only requires annotated images that contain fire or smoke to train.

The training process of the YOLO model can be described as:

1. Data preparation
    (a) Run the k-means algorithm to find K predefined anchor boxes. In the YOLOv3 for COCO settings, K is set to 9.
    (b) For each object, encode its bounding box based on all K anchor boxes.
2. Training
    (a) Randomize the input and collect them into batches. They are then converted into tensors. Let I be the tensor a batch of input.
    (b) Feed I into the backbone. Apply optimization algorithms like SGD, Adam to update the parameters.
    (c) Decrease the learning rate as the iteration increases.
    (d) repeat (a) - (c).
3. Stop training until convergence.

Then we can use the trained model for inference. Since we make at most K predictions for an object, it's expected to see an object being encircled by K or more bounding boxes. We'll use the Non-Max Suppression mechanism to filter out those undesired ones. More specifically:

1. Select the box with highest objectiveness score.
2. Calculate the intersection over union of this box with other boxes. Remove those with a overlap greater than 50
3. Move to the next bounding box with highest objectiveness score.
4. repeat 1-3 until no boxes could be removed.

The trained model will detect possible flaming points in the image by drawing bounding boxes around them.

C. The Unified Detector

The "Big" and the "Little" model will be organized as follows;

1. Preparation: train both detectors according to the aforementioned process.
2. Feed the video stream into the "Little" model. If it detects a potential fire, send the frame to the "Big" model.
3. Randomly trigger the "Big" model at a probability of p every second.

For most of the time, only the "Little" model will be used. The "Big" model will be used as a peer to review the decision made by the "Little" model, eliminating much of the potential false positive cases. It will also wake up occasionally to reduce the chance of false negative. By applying such algorithm, a set of computing hardware designed for one camera could be extended to roughly $floor(1/p)$ cameras.

IV. EXPERIMENTS

We will conduct our simulation in three aspects: accuracy, detection @ threshold and efficiency. Each aspect will include two scenarios. By doing this, we can have a comprehensive comparison between our algorithm and various other solutions. We expect that our algorithm will win the competition in efficiency and have similar performance in accuracy. The experiments are conducted on roughly 18,000 frames and 70 different events. The runtime is a 2 core Colab environment with no tensor accelerating units, e.g., GPUs.

A. Accuracy

For accuracy, we will consider two aspects: single frame and the whole event.

1. Single frame will be straightforward. However, our color model can only detect fire in video streams. Thus the single frame test will only be applied for Big models, i.e., the YOLO family and Faster R-CNN.
2. Since a fire is an event that last some moments. The accuracy for a single frame might not reflect the model's performance in real life. For example, in a fire clip where we have 400 frames, even if we can detect 50% of them (i.e., 50% single frame accuracy), we still successfully detect the fire. That's why we also evaluate the accuracy of a whole event. In our settings, an event is defined as a video clip which might or might not contain fire. The result is considered on the whole event basis.

As shown below, YOLOv5m shows the best result across all tested methods. However, we choose the YOLOv5s as the *Big* model since it has relatively good accuracy with far less resource requirement. The faster r-cnn is hard to train in our scenario while being the slowest to use. It's not attending the following tests.

TABLE I
Accuracy @ Frame

	YOLOv5n	YOLOv5s	YOLOv5m	Faster R-CNN
Accuracy	0.36	0.46	0.51	0.39

We test our Big-Little (BnL) model with four sets of parameters as well as the separate little and big model against Accuracy @ Event. The notation "$BnL - a - b$" stands for running the little model for a frames then run the big model for b frames. The BnL models show close if not same accuracy to the big model on an event basis. This is a preliminary proof of the effectiveness.

TABLE II
Accuracy @ Event

	BnL-30-3	BnL-30-2	BnL-30-1	BnL-60-1	Little	Big	Faster R-CNN
Accuracy	0.88	0.9	0.9	0.88	0.74	0.9	0.77

B. Detection @ Threshold

Although accuracy @ event is closer to the real world, it might be too loose as an indicator. The accuracy @ event can't show the responsiveness of the model. Suppose we the fire occurs on Frame 100, and the model detects it on Frame 800. That's the same in terms of accuracy, but timing can make a big difference. The Detection @ Threshold evaluate the model's ability to detect an fire event within the given threshold (t seconds). It's basically a variant of recall.

TABLE III
Detection @ Threshold t Seconds

	BnL-30-3	BnL-30-2	BnL-30-1	BnL-60-1	Little	Big	Faster R-CNN
[0, 1s)	0.8	0.74	0.71	0.66	0.54	0.83	0.66
[0s, 2s)	0.86	0.83	0.77	0.69	0.66	0.86	0.66
[0s, 3s)	0.86	0.86	0.83	0.8	0.71	0.86	0.71
[0s, 4s)	0.89	0.89	0.89	0.86	0.74	0.89	0.77

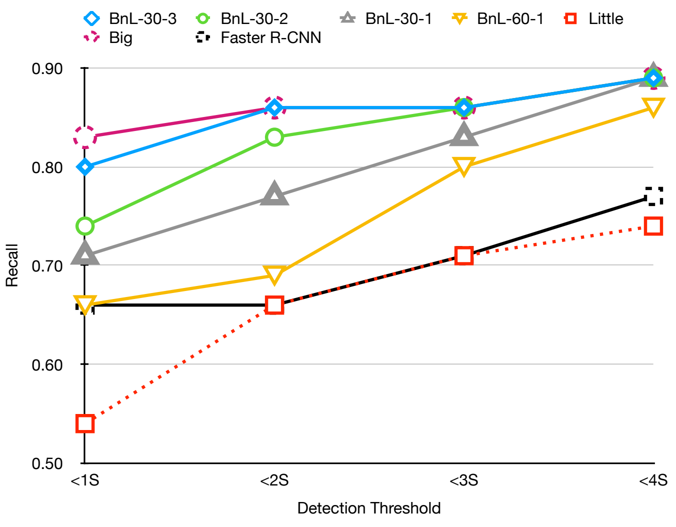
The above is the result and its illustration. It's clear that the big model is always the best meaning that it's the most responsive model. The little model however, keeps being the weakest one. The BnL models lies between the big and the little model. With the given settings, if the threshold is 4 seconds, i.e., we're allowed to detect the fire within the first 4 seconds of its occurrence, we can see that the BnL models have a very close if not same recall rate to the big model. Keep in mind that the big model is the upper bound of our models.

C. Efficiency

For efficiency, we will simulate two scenarios: single instance and multi-instances respectively. FPS will be chosen as the indicator to evaluate. The two scenarios are:

1. Single instance means we will run 1 model instance and observe the FPS. This measures the performance of our algorithm on low-end devices in unmanned areas. By doing this we can have a knowledge of the efficiency of the algorithm itself. Our model is expected to be comparable to the statistical color model.
2. For multi instance, we will provide relatively strong computational resources. We'll run multiple instances

Fig. 2. Detection @ Threshold

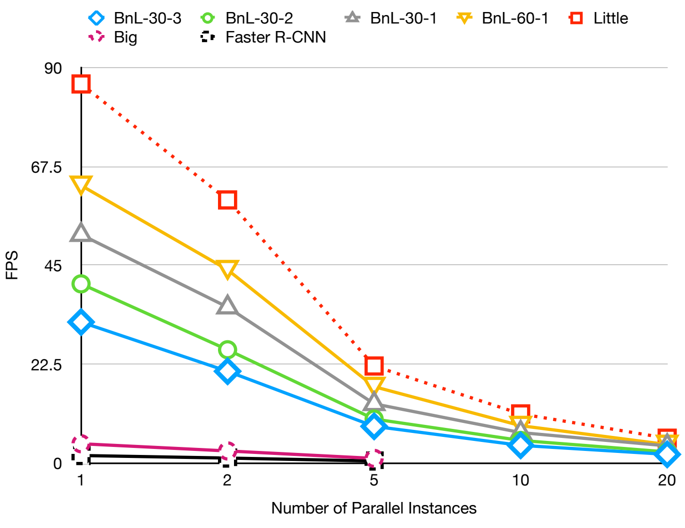


and measure the mean FPS across instances. This simulation can provide us an insight of the model's hardware sharing capability. This simulate the performance of model for multiple indoor security cameras with limited numbers of graphics card. Our model is expected to have a similar FPS decay rate as concurrency increase with the statistical color model.

TABLE IV
FPS @ Number of Instances

	BnL-30-3	BnL-30-2	BnL-30-1	BnL-60-1	Little	Big	Faster R-CNN
@ 1	32.1	40.8	52	63.2	86.3	4.41	1.73
@ 2	20.9	25.8	35.5	44	59.9	2.77	1.15
@ 5	8.37	10.03	13.5	17.4	22.1	1.07	0.48
@ 10	4.04	5.16	6.95	8.52	11.2	N/A	N/A
@ 20	1.99	2.55	3.92	4.23	5.67	N/A	N/A

Fig. 3. FPS @ Number of Instances



Models have a greater difference in efficiency than in performance. The Little model is 20 times faster than the big model. The good thing is, the BnL models are closer to the little model on this aspect. The can run at 30-60 fps on single and more than 2 fps when running 20 of them together on a single machine. We are unable to run more than 5 instances of the big model and Faster R-CNN for 2 reasons, 1) The memory is restricted on our device so we can't run them together, 2) even if we can, that's taking unnecessarily long time to finish.

V. CONCLUSION

The BnL architecture provides a effective way to balance the need for accuracy and efficiency. For example, you can achieve 97% of the detection rate with a threshold of 4 seconds of a YOLO model but being 15 times faster. If you need better precision, you can also find a combination that gives you the same detection rate with a threshold of 2 seconds while still being 8 times faster. You can run 5 instances of that and still being real time on a two core machine. At a cost of 25% slower, you're 50% more accurate.

Our work shows the effectiveness of its potential to be applied in real world context. However, due to the lack of forest fires open dataset we can find online, we are testing and training our model with data with fire but not in a forestry context. Furthermore, there is not enough time for us to actually deploy our system to test its performance in a challenging environment. In our experiments, we found that the color model can only detect fires with typical color. If the fire is too bright or too dim, it may miss it. We also found that the models were easily fooled by sunrise and sunset.

To address the aforementioned issues, We will explore more approaches like the work from Mueller et al.'s [16] that uses optical flow. With more time and resources in the future, we will implement and test our system in various environment. We'll also enhance our data set to make the models more robust to complicate scenes.

References

[1] BC Wildfire Service. *Wildfire causes - province of british columbia*. eng. URL: `https://www2.gov.bc.ca/gov/content/safety/wildfire-status/about-bcws/wildfire-response/fire-characteristics/causes` (visited on 04/27/2022).

[2] Natural Resources Canada. *Canadian wildland fire information system | canadian national fire database(Cnfdb)*. eng. URL: `https://cwfis.cfs.nrcan.gc.ca/ha/nfdb` (visited on 04/27/2022).

[3] Ahmad A. A. Alkhatib. "A review on forest fire detection techniques". en. In: *International Journal of Distributed Sensor Networks* 10.3 (Mar. 2014), p. 597368. ISSN: 1550-1477, 1550-1477. DOI: `10.1155/2014/597368`. URL: `http://journals.sagepub.com/doi/10.1155/2014/597368` (visited on 04/27/2022).

[4] Natural Resources Canada. *Canadian wildland fire information system | data sources and methods for daily maps*. eng. URL: `https://cwfis.cfs.nrcan.gc.ca/background/dsm/fm3` (visited on 04/27/2022).

[5] Simon Y. Foo. "A rule-based machine vision system for fire detection in aircraft dry bays and engine compartments". en. In: *Knowledge-Based Systems* 9.8 (Dec. 1996), pp. 531–540. ISSN: 09507051. DOI: `10.1016/S0950-7051(96)00005-6`. URL: `https:`

//linkinghub.elsevier.com/retrieve/pii/S0950705196000056 (visited on 04/27/2022).

[6] G. Healey et al. “A system for real-time fire detection”. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. New York, NY, USA: IEEE Comput. Soc. Press, 1993, pp. 605–606. ISBN: 9780818638800. DOI: 10.1109/CVPR.1993.341064. URL: http://ieeexplore.ieee.org/document/341064/ (visited on 04/27/2022).

[7] Che-Bin Liu and Narendra Ahuja. “Vision based fire detection”. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 4. IEEE. 2004, pp. 134–137.

[8] Khan Muhammad et al. “Convolutional neural networks based fire detection in surveillance videos”. In: *IEEE Access* 6 (2018), pp. 18174–18183.

[9] Khan Muhammad et al. “Efficient fire detection for uncertain surveillance environment”. In: *IEEE Transactions on Industrial Informatics* 15.5 (2019), pp. 3113–3122.

[10] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[11] Zhentian Jiao et al. “A deep learning based forest fire detection approach using UAV and YOLOv3”. In: *2019 1st International conference on industrial artificial intelligence (IAI)*. IEEE. 2019, pp. 1–5.

[12] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[13] Byoungjun Kim and Joonwhoan Lee. “A video-based fire detection using deep learning models”. In: *Applied Sciences* 9.14 (2019), p. 2862.

[14] Christopher Richard Wren et al. “Pfinder: Real-time tracking of the human body”. In: *IEEE Transactions on pattern analysis and machine intelligence* 19.7 (1997), pp. 780–785.

[15] Turgay Celik et al. “Fire detection using statistical color model in video sequences”. In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 176–185.

[16] Martin Mueller et al. “Optical Flow Estimation for Flame Detection in Videos”. In: *IEEE Transactions on Image Processing* 22.7 (2013), pp. 2786–2797. DOI: 10.1109/TIP.2013.2258353.