Signal, Image and Video Processing (2020) 14:675–682
https://doi.org/10.1007/s11760-019-01600-7

ORIGINAL PAPER

Additive neural network for forest fire detection

Hongyi Pan[1] · Diaa Badawi[1] · Xi Zhang[1] · Ahmet Enis Cetin[1,2]

Received: 27 July 2019 / Revised: 12 September 2019 / Accepted: 8 November 2019 / Published online: 18 November 2019

Abstract
In this paper, we introduce a video-based wildfire detection scheme based on a computationally efficient additive deep neural network, which we call AddNet. This AddNet is based on a multiplication-free vector operator, which performs only addition and sign manipulation operations. In this regard, we construct a dot product-like operation from the mf-operator and use it to define dense and convolutional feed-forwarding passes in AddNet. We train AddNet on images taken from forestry surveillance cameras. Our experiments show that AddNet can achieve a time-saving by 12.4% when compared to an equivalent regular convolutional neural network (CNN). Furthermore, the smoke recognition performance of AddNet is as good as regular CNNs and substantially better than binary-weight neural networks.

Keywords Computationally efficient · Neural network · Additive neural network · Real-time · Forest fire detection

1 Introduction

Despite recent advances in weather forecasting and firefighting technology, devastating forest fires occur throughout the world. For example, in November 2018, the "Camp Fire" in California burned about 153,336 acres and resulted in a death toll of more than 85 people. Early detection of wildfire is critical to minimizing environmental and human losses. In recent years, there has been significant interest in developing real-time algorithms to detect wildfires using regular video-based surveillance systems [1–18]. Video-based forest fire detection can be used to replace traditional point-sensor-type detectors since a single camera can monitor a very large area from a distance and can detect wildfire smoke immediately after fire eruption as long as the smoke is within the viewing range of the camera.

Deep neural networks (DNNs) are widely used in image recognition tasks due to their highly powerful recognition capabilities. In this paper, our goal is to implement a DNN-based wildfire smoke detection system using a simple processor. Such a system can be deployed on drones or remote monitoring towers where the aim is to consume as little computation as possible. Reducing the amount of computation reduces the energy consumption. Therefore, a drone or a remote station that uses a more energy-efficient algorithm can operate for longer time. A typical neuron (or perceptron) in feed-forwarding neural networks carries out three main tasks to produce an output: (i) an inner product operation which involves multiplication of inputs by weights, (ii) additions of the resulting multiplication operations (accumulation) and bias addition and (iii) applying nonlinear activation over the result of the affine transformation. The multiplication operations are the most computationally consuming operation in a typical processor [19]. In this paper, we describe an addition-based efficient neural network and use it in video-based wildfire detection system. In our system, we use an l_1 norm-based neural network, called additive neural network (AddNet), which replaces the regular multiplication operator with a new computationally efficient operator called multiplication-free (mf)-operator. Afrasiyabi et al. show that mf-operator-based neural networks perform as well as regular neural networks on MNIST dataset and CIFAR-10 dataset [20]. Instead of multiplications, the mf-operator performs sign multiplications and addition operations in a typical neu-

✉ Hongyi Pan
hpan21@uic.edu

Diaa Badawi
dbadaw2@uic.edu

Xi Zhang
xzhan62@uic.edu

Ahmet Enis Cetin
aecyy@uic.edu

[1] University of Illinois at Chicago, Chicago, USA

[2] Bilkent University, Ankara, Turkey

ron. The sign multiplication of two real numbers is a simple bit operation.

Other solutions to computationally efficient neural networks require dedicated software for a specific hardware, such as neuromorphic devices [21] and Movidius Myraid architecture [22]. Although these approaches reduce computational consumption and computation amount, they rely on extra special hardware [23]. AddNets, on the other hand, can be implemented in ordinary microprocessors and digital signal processors, which are portable and low cost. The system needs no other special hardware to deploy and can implement a well-trained neural network.

In the next section, we review the mf-operator and describe the mf-operator-based convolutional neural networks. In Sect. 3, we describe the AddNet algorithm. We compare it with a regular convolutional neural network (CNN) and a binarized weight convolutional neural network.

2 Multiplication-free vector product

Let α and β be two real-valued scalars. We define our multiplication-free operator as follows:

$$\alpha \oplus \beta = \mathrm{sign}(\alpha \cdot \beta)(|\alpha| + |\beta|) \tag{1}$$

where sign(.) is the signum function, defined as follows:

$$\mathrm{sign}(x) = \begin{cases} 1, & x > 0 \\ -1, & x \leqslant 0 \end{cases} \tag{2}$$

The multiplication-free operator can also be represented as follows:

$$\alpha \oplus \beta = \mathrm{sign}(\alpha) \cdot \beta + \alpha \cdot \mathrm{sign}(\beta) \tag{3}$$

Note that bit-wise operation is very efficient in computation, and the highest bit of a variable is its sign bit (0 for positive and 1 for negative), we can achieve the $\alpha \cdot \mathrm{sign}(\beta)$ with XOR operation, which is much more efficient compare to the multiplication.

Furthermore, the scalar definition in (1) and (3) can be extended to the case of real vectors in order to construct a dot product-like operation. In this regard, let $\mathbf{x}$ and $\mathbf{w} \in \mathbb{R}^d$. We define the multiplication-free "dot product" as follows:

$$\mathbf{w} \oplus \mathbf{x} = \sum_{i=1}^{d} \mathrm{sign}(w_i x_i)(|w_i| + |x_i|) \tag{4}$$

Similarly, the above relation can be also expressed as follows:

$$\mathbf{w} \oplus \mathbf{x} = \sum_{i=1}^{d} \mathrm{sign}(w_i)x_i + w_i \mathrm{sign}(x_i) \tag{5}$$

As the dot product induces the ℓ_2 norm, the mf-vector operation induces a scaled version of the ℓ_1 norm as follows:

$$\mathbf{x} \oplus \mathbf{x} = \sum_{i=1}^{d} \mathrm{sign}(x_i x_i)(|x_i| + |x_i|) = 2||\mathbf{x}||_1 \tag{6}$$

For convenience, we define the corresponding matrix-vector operation as follows: Let the vector $\mathbf{x} \in \mathbb{R}^d$ and the matrix $\mathbf{W} \in \mathbb{R}^{d\times k}$. We then define the matrix-vector mf-operation as follows:

$$\mathbf{y} = \mathbf{W} \oplus \mathbf{x} = [\mathbf{w_1} \oplus \mathbf{x}\ \mathbf{w_2} \oplus \mathbf{x}\ \ldots\ \mathbf{w_k} \oplus \mathbf{x}]^{\mathrm{T}} \tag{7}$$

where $\mathbf{w_i}$ is the ith column of $\mathbf{W}$ for $i = 1, 2, \ldots, k$ and $\mathbf{y} \in \mathbb{R}^k$ is the resulting vector.

3 Additive neural network with mf-operator (ADDNET)

3.1 Representation of neurons

In regular neural networks, a dense feed-forwarding pass can be expressed as follows:

$$\mathbf{y} = \phi(\mathbf{W}^{\mathrm{T}}\mathbf{x} + \mathbf{b}) \tag{8}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input vector, $\mathbf{W} \in \mathbb{R}^{d\times k}$ is the weights matrix, $\mathbf{b} \in \mathbb{R}^k$ is the bias vector and $\phi(.)$ is the element-wise nonlinear activation. In AddNet, the feed-forwarding pass of a dense layer can be expressed as follows:

$$\mathbf{y} = \phi(\mathbf{a} \odot (\mathbf{W} \oplus \mathbf{x} + \mathbf{b})) \tag{9}$$

where $\odot$ is the element-wise product between the result $\mathbf{W} \oplus \mathbf{x} + \mathbf{b}$ and a vector $\mathbf{a} \in \mathbb{R}^k$. Note that we introduce the vector $\mathbf{a}$ as a scaling parameter in order to control the range of the pre-activations resulting from the mf-operation. It should be pointed out that calculating the element-wise product is inexpensive since we only carry out d multiplications compared to $k \times d$ multiplication operations in the case of $\mathbf{W}^{\mathrm{T}}\mathbf{x}$.

We can construct that AddNet convolutional layers in a straightforward manner by substituting each convolution (dot product) operation with the mf-equivalent operation.

3.2 Training the AddNet

The standard back-propagation algorithm can be used for training the AddNet with the need of small approximations. The partial scalar derivatives of the pre-activation response with respect to are given as follows:

$$\frac{\partial(w \oplus x)}{\partial x} = \operatorname{sign}(w) + 2w\delta(x) \tag{10}$$

$$\frac{\partial(w \oplus x)}{\partial w} = 2x\delta(w) + \operatorname{sign}(x) \tag{11}$$

where $\delta(x)$ is the Dirac-delta function that directly results from the discontinuity of the signum function at $x = 0$. If we omit the delta function from the definitions of the partial derivatives, we end up with binary derivatives ($\operatorname{sign}(w)$ and $\operatorname{sign}(x)$). However, we found that approximating the Dirac-delta function provides better convergence since we end up with smoother derivatives. In this regard, we approximate the derivative of the signum function to be that of a steep hyperbolic tangent, as follows:

$$\frac{d\operatorname{sign}(x)}{dx} \approx \frac{d\tanh(\alpha x)}{dx} = \alpha\big(1 - \tanh^2(\alpha x)\big) \tag{12}$$

for a scalar $\alpha >> 1$. This is reasonable since $\operatorname{sign}(x) = \lim_{\alpha\to\infty} \tanh(\alpha x)$. This way the terms associated with the delta function in the formula (10) and (11) will contribute to the partial derivatives when the arguments are close to zero.

3.3 Computational efficiency

With an input $\mathbf{x} \in \mathbb{R}^d$ and a weight matrix $\mathbf{W} \in \mathbb{R}^{d\times k}$, one needs a total of $k \cdot d$ multiplication operations to realize the matrix-vector product, which is the case for layers in a regular neural network.

On the other hand, AddNet substitutes the multiplication operations in $\mathbf{W}^{\mathrm{T}}\mathbf{x}$ with addition and bit-wise operation, which are more efficient in terms of energy and computation. The realization of the calculation in the mf-operator is inexpensive because it only involves 1-bit operations and can be implemented as a logical XOR operation. So AddNet performs only one multiplication per neuron. The activation function is ReLU. Therefore, **a** determines the slope of the ReLU. Thus, AddNet needs far fewer multiplication operations that does a regular neural network.

4 Experimental results

4.1 Speed test

In this section, we compare the speed of CNN kernel, AddNet kernel and binary CNN kernel by creating two 256 × 256 matrices, two 512 × 512 matrices and two 1024 × 1024 matrices with random entries and then performing kernel operations, respectively. These matrices sizes are widely used in convolutional neural networks design. We compare their speed in matrices operation instead of testing the trained neural networks speed on the dataset is because other operations like images loading, activation functions also take time, and the time required by these operations differs each time due to different computer statuses. It is much clearer to show the acceleration of AddNet without those unrelated operations. Moreover, because we apply same activation function (ReLU) CNN, AddNet and binary CNN models in Sects. 4.3–4.7, we only test the time requirement without any activation function.

For CNN kernel, we do regular element-wise matrices product, for AddNet kernel, we replace the multiplication between elements with our mf-operator, and for binary CNN kernel, we replace it with $\alpha \cdot \operatorname{sign}(\beta)$. As we all know, fire detection is a real-world application. It is important to lower the cost of the equipment as much as possible. However, GPU is powerful but very expensive. We should employ GPU to train the neural networks, but it is unwisely to deploy the well-trained networks also on a device with GPU. Thus, we only interest in CPU speed performance. We use G++ compiler to compile the speed test code with optimization-level O3 and then run the executable file on the Ubuntu system on a single thread. Since background apps may interfere with the timing, we run the operations 10 times and compare the minimum time, respectively.

The speed test results are listed in Table 1. From the speed test on 1024 × 1024 matrices, CNN kernel takes 268.123 ms, AddNet kernel takes 234.825 ms, and binary CNN kernel takes 210.935 ms. So the AddNet kernel saves 33.298 ms (12.419%), while the binary CNN kernel saves 57.188 ms (21.329%). Thus, the AddNet kernel is a trade-off between the binary kernel and the regular CNN kernel.

4.2 Dataset augmentation

We train the neural networks with various wildfire images. We also augment our data during training by shifting and translating wildfire images, and we fill the images by reflect translation. The reason for using such an augmentation scheme is to ensure that the classifier will see examples in which wildfire starts from different locations during training.

4.3 Dataset 1: images from the internet

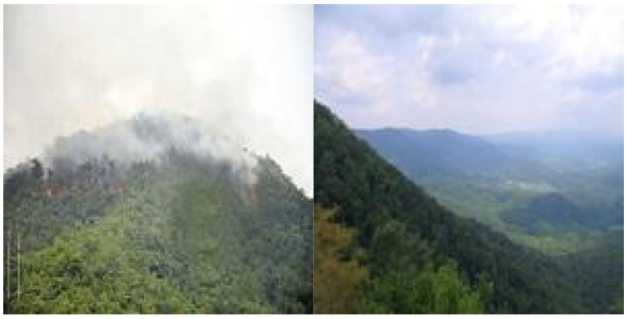
We first build a forest fire dataset with 4000 images (2000 forest images with fire and 2000 forest images without fire) collected from the Internet. They are resized into 150 by 150 pixels, and they are divided into 3000 training images and 1000 test images. In each dataset, the amount of fire images is equal to the amount of no fire images. Figure 1 shows an example of a wild fire image and a no fire image, respectively.

4.4 Training models for dataset 1

Here, we train three neural networks: a regular CNN, an AddNet and a binary CNN all with same architecture for

Table 1 Speed test results

Matrix size	CNN① (ms)	AddNet② (ms)	Binary CNN③ (ms)	①−② (ms)	①−③ (ms)	②'s saving (%)	③'s saving (%)
256	3.672	3.138	2.717	0.534	0.955	14.542	26.008
512	30.914	26.912	23.647	4.002	7.267	12.946	23.507
1024	268.123	234.825	210.935	33.298	57.188	12.419	21.329



(a) A forest image with fire, **(b)** A forest image without fire

Fig. 1 Example images in dataset 1

Table 2 Architecture of the neural network

Layer	Layer specification
Input layer	150 × 150 × 3
Convolutional layer	64 5 × 5 filters, stride = 3
Batch normalization	–
Max pooling	Strides: 2 × 2
Convolutional layer	64 3 × 3 filters
Batch normalization	–
Max pooling	Strides: 2 × 2
Convolutional layer	96 3 × 3 filters
Batch normalization	–
Max pooling	Strides: 2 × 2
Fully connected layer	Output size: 1536
Fully connected layer	Output size: 128
Output layer	Output size: 1

the task of forest fire detection using TensorFlow-Keras deep learning library on a computer with NVIDIA GPU. The architecture is shown in Table 2 and Fig. 2. We use our mf-operator in AddNet in all layers except for the output layer. Similarly, we use binary weights in all layers except for the output layer in binary CNN.

Test performance is shown in Table 3. All three networks reach very high accuracy, while the AddNet and the binary CNN have a little lower true-detected accuracy than the regular CNN, and the binary CNN has much higher false alarm rate.

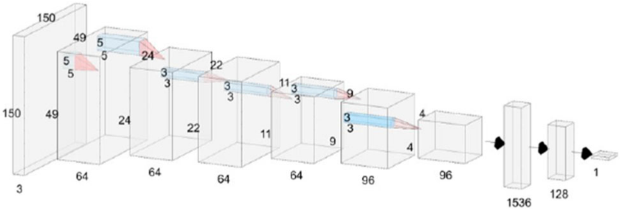


Fig. 2 Architecture of the neural network

4.5 Dataset 2: frames from forestry surveillance videos

Nowadays, with the development of technology, we can obtain forestry surveillance videos in 1080P (1920 × 1080) or higher resolution. However, if we design the input of the neural network in these so high resolutions, the network will be too huge to train. The most common method to solve this problem is to resize and down-sample the images, but if we resize the whole frames into the input size of the neural network, little smoke will be too small to be detected. To overcome this problem, we divide the frames into many tiles as Fig. 3. The videos are in 1080P, and we divided them into many 180 × 180 tiles, and each four tiles (2 × 2) consist a window (360 × 360). The score is the forest fire rate in each window, and composed of the tile has score and its bottom, right and bottom-right tiles, rather than the forest fire rate of each tile. This is the reason that the most right and the most bottom tiles have no score. In this way, if the smoke exists at the edge of one window, it will also exist at the center of its neighbor window. Then, we resize each window into the input size and feed them to the network.

In this section, we use the whole dataset 1, dataset of forestry surveillance videos for the task of smoke detection [18] and some 1080P forestry surveillance videos for training, and then, we test the network on some other 1080P videos. Videos in [18], shown in Fig. 4, are in relatively low resolution and quality, and it has some text on the frames, so we crop out the text part, then resize and label the remains. As for these 1080P videos, we label the frames by windows. Each video corresponds to a different surveillance camera scenery. Some videos have wildfire eruption events, whereas others have no wildfire occurrence. We split the videos into frames at each time instance and constructed our data from RGB images.

Table 3 Test performance of three models

Rate name	CNN (%)	AddNet (%)	Binary CNN (%)
True-detected rate	96.0	95.6	92.4
False alarm rate	0.6	0.6	0.8

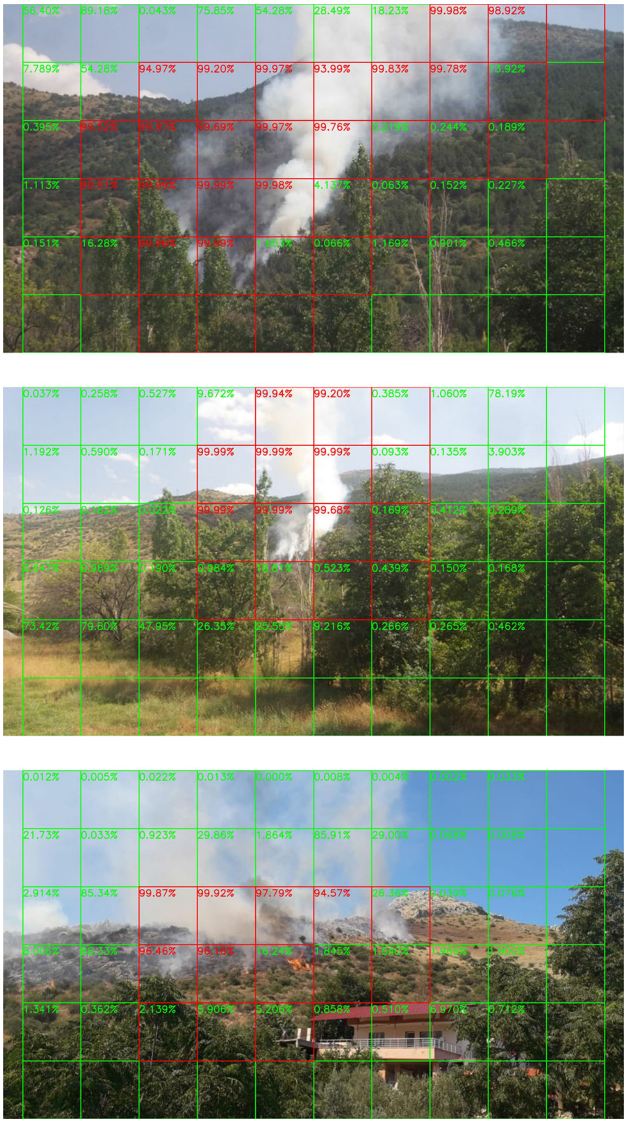
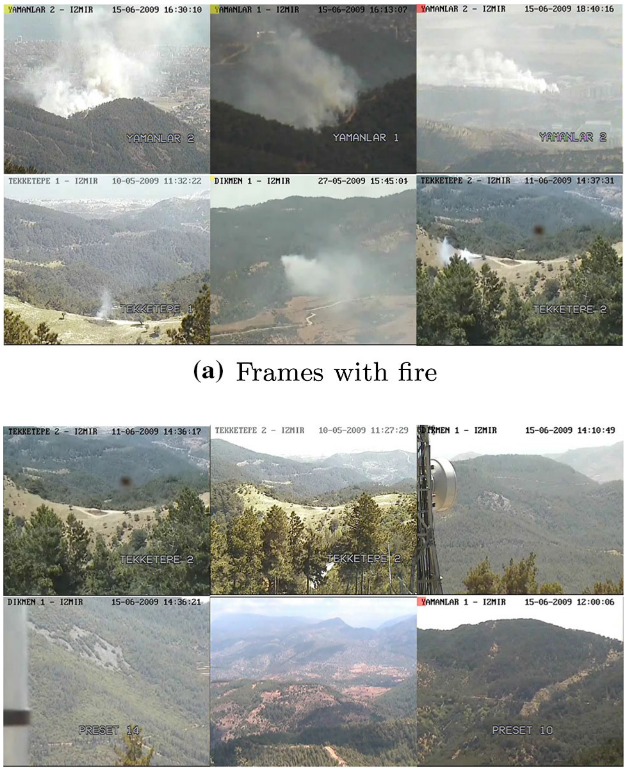


Fig. 3 1080P detection results

4.6 Training models for dataset 2

Because dataset 2 is much more challenging than dataset 1, we use global average pooling to prevent overfitting [24] in this section. The architecture is shown in Table 4 and Fig. 5. Similarly, we used the mf-operator in AddNet and binary weights in binary CNN in all layers except the output layer.

To lower the false alarm rate, we increase the threshold of the fire category. Test accuracy on windows with smoke is shown in Table 5, and false alarm test rate on windows


YAMANLAR 2 - IZMIR 15-06-2009 16:30:10
YAMANLAR 2
YAMANLAR 1 - IZMIR 15-06-2009 16:13:07
YAMANLAR 1
YAMANLAR 2 - IZMIR 15-06-2009 18:40:16
YAMANLAR 2
TEKKETEPE 1 - IZMIR 10-05-2009 11:32:22
TEKKETEPE 1
DIKMEN 1 - IZMIR 27-05-2009 15:45:04
TEKKETEPE 2 - IZMIR 11-06-2009 14:37:31
TEKKETEPE 2
TEKKETEPE 2 - IZMIR 11-06-2009 14:36:17
TEKKETEPE 2
TEKKETEPE 2 - IZMIR 10-05-2009 11:27:29
TEKKETEPE 2
DIKMEN 1 - IZMIR 15-06-2009 14:10:49
DIKMEN 1 - IZMIR 15-06-2009 14:36:21
PRESET 10
YAMANLAR 1 - IZMIR 15-06-2009 12:00:06
PRESET 10

(a) Frames with fire

(b) Frames without fire

Fig. 4 Low-quality frames in training dataset

Table 4 Architecture of the neural network

Layer	Layer specification
Input layer	150 × 150 × 3
Convolutional layer	64 5 × 5 filters, stride = 3
Batch normalization	–
Max pooling	Strides: 2 × 2
Convolutional layer	64 3 × 3 filters
Batch normalization	–
Max pooling	Strides: 2 × 2
Convolutional layer	96 3 × 3 filters
Batch normalization	–
Max pooling	Strides: 2 × 2
Convolutional layer	128 1 × 1 filters
Batch normalization	–
Global average pooling	–
Fully connected layer	Output size: 128
Output layer	Output size: 1

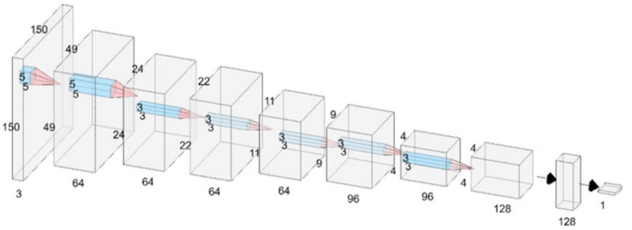


Fig. 5 Architecture of the neural network

Table 5 True-detected rate on windows with smoke

Videos	Total	CNN (%)	AddNet (%)	Binary CNN (%)
Video 1	36	77.78	66.67	66.67
Video 2	32	78.13	68.75	65.63
Video 3	38	55.26	47.37	44.74
Video 4	24	66.67	83.33	91.67
Video 5	21	4.76	47.62	76.19
Video 6	18	77.78	44.44	33.33
Video 7	168	72.62	83.33	86.90
Video 8	121	98.35	99.17	99.17
Video 9	12	58.33	66.67	58.33
Video 10	40	80.00	65.00	65.00
Video 11	30	70.00	43.33	26.67
Video 12	72	69.44	65.28	62.50
Video 13	48	89.58	87.50	77.08
Average		75.61	75.45	75.00

without smoke is shown in Table 6. The average detection true-detected rate of the AddNet is very close to the regular CNN, but the binary CNN performs much worse. It should be noted that the regular CNN has a dismal performance on Video 5. It only detected about 4% of frames containing smoke. On the other hand, the AddNet detected 47% of smoke frames in this difficult video clip.

Table 6 False alarm rate on normal windows

Videos	Total	CNN (%)	AddNet (%)	Binary CNN (%)
Video 0	1377	7.33	6.03	7.55
Video 1	999	9.91	8.21	9.91
Video 2	837	1.79	4.18	2.63
Video 3	747	6.69	10.84	13.25
Video 4	972	1.54	3.60	2.26
Video 5	1125	4.09	3.20	3.64
Video 6	945	6.46	4.44	5.40
Video 7	279	5.73	11.47	5.02
Video 8	729	10.84	8.78	11.25
Video 9	1107	8.94	7.41	8.94
Video 10	567	2.65	6.17	3.88
Video 11	810	1.48	2.59	1.36
Video 12	810	6.17	10.00	12.22
Video 13	1062	8.95	7.72	9.32
Average		6.09	6.40	6.99

4.7 Robustness test

In real-world application, camera lenses cannot always keep clean after a long period usage. Although there exist some methods to recover the images captured by cameras with dirty lenses [25], they require suitable parameters which need to be set manually. It will be helpful if the neural networks can distinguish dirt from smoke from raw images.

In this section, we add camera dirt effect on the test frames, then test and compare three models' robustness. This is because cameras get dirty over time. Jinwei et al. show that an image $I(x, y)$ captured by the camera consists of two components, attenuation where the radiance emitted from the target scene is attenuated by the intermediate layer and intensification where the intermediate layer itself contributes some radiance to the camera, by either scattering light from other directions in the environment or reflecting light from its surface [25]. Suppose $I_0(x, y)$ is the radiance of the target scene, $\alpha \in [0, 1]$ is the attenuation pattern of the intermediate layer, i.e., the fraction of light transmitted (0 means blocked completely and 1 means passed completely), and $I_\alpha(x, y)$ is the intensification term. The final image I is the sum of the attenuated light from the background after the defocus blur $I_0 \cdot (\alpha * k)$ and the light emitted from the intermediate layer itself $I_\alpha * k$.

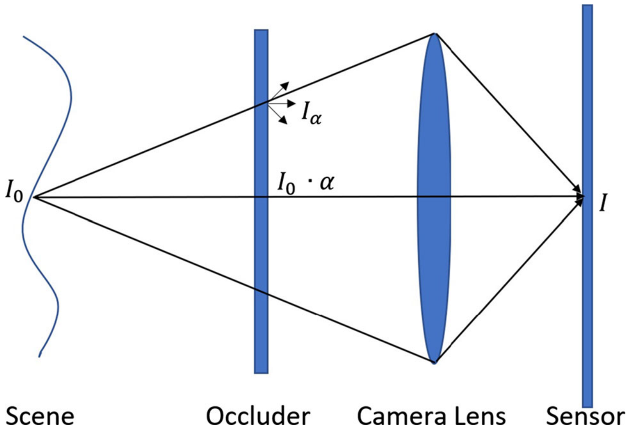
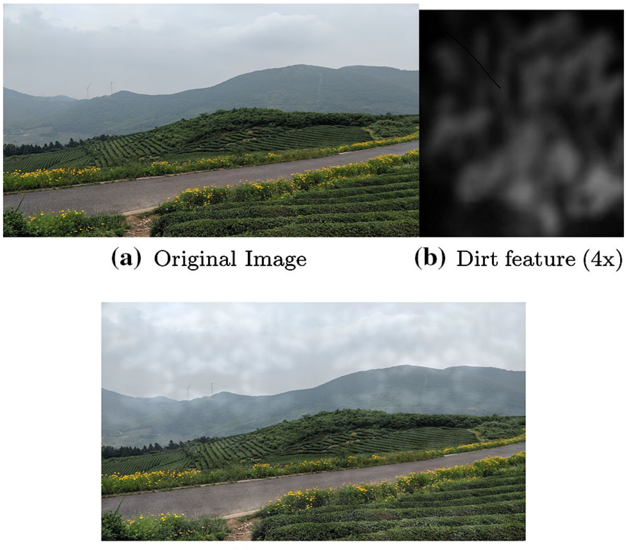


Fig. 6 Image formation model

As shown in Fig. 6, the image formation model we used in this section is:

$$I = I_0 \cdot (\alpha * k) + I_\alpha * k \tag{13}$$

where $k(x, y)$ is the defocus blur kernel for the intermediate layer.



(a) Original Image **(b)** Dirt feature (4x)

(c) Fused Image

Fig. 7 1080P image, dirt feature and fused image

Table 7 True-detected rate on windows with smoke with dirt feature

Videos	Total	CNN (%)	AddNet (%)	Binary CNN (%)
Video 1	36	66.67	52.78	69.44
Video 2	32	78.13	68.75	68.75
Video 3	38	63.16	47.37	34.21
Video 4	24	83.33	79.17	91.67
Video 5	21	14.29	66.67	95.24
Video 6	18	83.33	55.56	33.33
Video 7	168	70.83	84.52	86.90
Video 8	121	100.00	95.87	99.17
Video 9	12	66.67	66.67	50.00
Video 10	40	77.50	62.50	70.00
Video 11	30	73.33	46.67	33.33
Video 12	72	59.72	63.89	63.89
Video 13	48	87.50	89.58	66.67
Average		75.30	75.15	75.15

Table 8 False alarm rate on normal windows with dirt feature

Videos	Total	CNN (%)	AddNet (%)	Binary CNN (%)
Video 0	1377	9.73	7.70	9.08
Video 1	999	12.11	11.11	12.71
Video 2	837	2.39	5.97	4.18
Video 3	747	8.43	13.52	15.80
Video 4	972	4.63	5.25	3.40
Video 5	1125	8.89	3.29	3.82
Video 6	945	9.31	4.76	5.82
Video 7	279	5.73	16.13	7.89
Video 8	729	11.93	12.76	13.85
Video 9	1107	10.93	10.03	10.75
Video 10	567	4.59	8.99	6.17
Video 11	810	3.21	3.95	1.73
Video 12	810	8.40	12.47	14.69
Video 13	1062	10.92	10.45	11.21
Average		8.34	8.45	8.61

Figure 7 shows a sample image without and with dirt feature. Dirty feature in Fig. 7 is amplified 4 times as its original values to show here for better display, and we fill it into entire frame. We fuse all test frames with the dirt feature in this way and then test the performance of three models on them.

The results are shown in Tables 7 and 8. Compared to Tables 5 and 7, it seems that influence of dirt feature on true-detected rate is slight and uncertain. Some videos (video 3–6, 8–9, 11 for CNN; video 5–7, 11, 13 for AddNet and video 1–2, 5, 10–12 for binary CNN) return higher true-detected rate, while some others (video 1, 7, 10, 12–13 for CNN; video 1, 4, 8, 10, 12 for AddNet and video 3, 9, 13 for binary CNN) return lower true-detected rate. All wildfire events in videos are successfully detected, and therefore, all the methods are robust to camera dirt.

On the other hand, however, compared to Tables 6 and 8, we find that the false alarm rates of all three models on each video increase or remain the same, and all three average false alarm rates rise a little (2.25% for CNN, 2.05% for AddNet and 1.62% for binary CNN). This is because dirt in the clear region makes the region looks like with smoke, which is confusing for the neural networks to distinguish since we have not trained them on images with dirt feature, so these losses are allowable in the real-world application. In spite of the average difference of the binary CNN is smaller than the CNN and the AddNet, its false alarm rate is still the highest.

5 Conclusion

In this paper, we proposed a multiplication-free neural network (AddNet) architecture. We applied AddNet to the problem of wildfire smoke detection task, in which case computationally efficiency and low false alarm rates are critical. We compared the results with those of a regular CNN and a binary CNN. It turns out that AddNet and CNN detect all the wildfires, but binary CNN fails in some cases in our dataset. AddNet can be also applied to three-dimensional wildfire detection schemes such as [6]. Moreover, a 3D extension of the AddNet can be also employed to process videos in time domain like regular CNN does [26], and the gain with the AddNet respect to the regular 3D CNN would be higher in terms of computational time. However, we have no plan about it in this paper due to the time limitation. We also carried our time analysis for inference over a PC. We found that the speed of AddNet is between regular CNN and binary CNN. We conclude that AddNet resembles binary CNN in terms of its computational efficiency because they do not need expensive matrix multiplication.

Acknowledgements The research is supported by National Science Foundation, (No. 1739396) 2017. We also thank NVIDIA company for providing a GPU and the developers of TensorFlow for publishing a so powerful machine learning library.

References

1. Töreyin, B.U., Dedeoğlu, Y., Güdükbay, U., Cetin, A.E.: Computer vision based method for real-time fire and flame detection. Pattern Recognit. Lett. **27**(1), 49–58 (2006)
2. Töreyin, B.U., Dedeoğlu, Y., Cetin, A.E.: Wavelet based real-time smoke detection in video. In: 2005 13th European Signal Processing Conference, pp. 1–4. IEEE (2005)
3. Habiboğlu, Y.H., Günay, O., Çetin, A.E.: Covariance matrix-based fire and flame detection method in video. Mach. Vis. Appl. **23**(6), 1103–1113 (2012)
4. Habiboglu, Y.H., Gunay, O., Cetin, A.E.: Real-time wildfire detection using correlation descriptors. In: 2011 19th European Signal Processing Conference, pp. 894–898. IEEE (2011)
5. Töreyin, B.U.: Smoke detection in compressed video. In: Applications of Digital Image Processing XLI, vol. 10752, p. 1075232. International Society for Optics and Photonics (2018)
6. Aslan, S., Güdükbay, U., Töreyin, B.U., Çetin, A.E.: Early wildfire smoke detection based on motion-based geometric image transformation and deep convolutional generative adversarial networks. In: ICASSP 2019: 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8315–8319. IEEE (2019)
7. Borges, P.V.K., Izquierdo, E.: A probabilistic approach for vision-based fire detection in videos. IEEE Trans. Circuits Syst. Video Technol. **20**(5), 721–731 (2010)
8. 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. IEEE (2007)
9. Celik, T., Demirel, H.: Fire detection in video sequences using a generic color model. Fire Saf. J. **44**(2), 147–158 (2009)
10. Yuan, F.: A fast accumulative motion orientation model based on integral image for video smoke detection. Pattern Recognit. Lett. **29**(7), 925–932 (2008)
11. Guillemant, P., Vicente, J.: Real-time identification of smoke images by clustering motions on a fractal curve with a temporal embedding method. Opt. Eng. **40**, 554–563 (2001)
12. Vicente, J., Guillemant, P.: An image processing technique for automatically detecting forest fire. Int. J. Therm. Sci. **41**(12), 1113–1120 (2002)
13. Gomez-Rodriguez, F., Arrue, B.C., Ollero, A.: Smoke monitoring and measurement using image processing: application to forest fires. In: Automatic Target Recognition XIII, vol. 5094, pp. 404–411. International Society for Optics and Photonics (2003)
14. Krstinić, D., Stipaničev, D., Jakovčević, T.: Histogram-based smoke segmentation in forest fire detection system. Inf. Technol. Control **38**(3), 237–244 (2009)
15. Luo, Q., Han, N., Kan, J., Wang, Z.: Effective dynamic object detecting for video-based forest fire smog recognition. In: 2009 2nd International Congress on Image and Signal Processing, pp. 1–5. IEEE (2009)
16. Toreyin, B.U., Cetin, A.E.: Computer vision based forest fire detection. In: 2008 IEEE 16th Signal Processing, Communication and Applications Conference, pp. 1–4. IEEE (2008)
17. Toreyin, B.U., Cetin, A.E.: Wildfire detection using LMS based active learning. In: 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 1461–1464. IEEE (2009)
18. Gunay, O., Toreyin, B.U., Kose, K., Cetin, A.E.: Entropy-functional-based online adaptive decision fusion framework with application to wildfire detection in video. IEEE Trans. Image Process. **21**(5), 2853–2865 (2012)
19. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
20. Afrasiyabi, A., Nasir, B., Yildiz, O., Vural, F.T.Y., Cetin, A.E.: An energy efficient additive neural network. In: 2017 25th Signal Processing and Communications Applications Conference (SIU), pp. 1–4. IEEE (2017)
21. Ionica, M.H., Gregg, D.: The movidius myriad architecture's potential for scientific computing. IEEE Micro **35**(1), 6–14 (2015)
22. Essera, S.K., Merollaa, P.A., Arthura, J.V., Cassidya, A.S., Appuswamya, R., Andreopoulosa, A., Berga, D.J., McKinstrya, J.L., Melanoa, T., Barcha, D.R., di Nolfoa, C.: Convolutional networks for fast energy-efficient neuromorphic computing. Proc. Nat. Acad. Sci. USA **113**(41), 11441–11446 (2016)
23. Painkras, E., Plana, L.A., Garside, J., Temple, S., Galluppi, F., Patterson, C., Lester, D.R., Brown, A.D., Furber, S.B.: SpiNNaker: a 1-W 18-core system-on-chip for massively-parallel neural network simulation. IEEE J. Solid State Circuits **48**(8), 1943–1953 (2013)
24. Lin, M., Chen, Q., Yan, S.: Network in network (2013). arXiv:1312.4400
25. Gu, J., Ramamoorthi, R., Belhumeur, P., Nayar, S.: Removing image artifacts due to dirty camera lenses and thin occluders. ACM Trans. Graph. (TOG) **28**(5), 144 (2009)
26. Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., Toderici, G.: Beyond short snippets: deep networks for video classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4694–4702(2015)