Scientific Research Publishing

Open Journal of Applied Sciences, 2021, 11, 1240-1255
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

Optimal Flame Detection of Fires in Videos Based on Deep Learning and the Use of Various Optimizers

Tidiane Fofana[1,2], Sié Ouattara[1,2,3*], Alain Clement[4]

[1]Laboratoire des Sciences et Technologies de la Communication et de l'Information (LSTCI), Yamoussoukro, Côte d'Ivoire
[2]Institut National Polytechnique Houphouët Boigny (INPHB), Yamoussoukro, Côte d'Ivoire
[3]Ecole Supérieure des Technologies de l'Information et de la Communication (ESATIC), Abidjan, Côte d'Ivoire
[4]LARIS, SFR MATHSTIC, Université d'Angers, Angers, France
Email: *sie_ouat@yahoo.fr

How to cite this paper: Fofana, T., Ouattara, S. and Clement, A. (2021) Optimal Flame Detection of Fires in Videos Based on Deep Learning and the Use of Various Optimizers. *Open Journal of Applied Sciences*, **11**, 1240-1255.
https://doi.org/10.4236/ojapps.2021.1111094

Received: October 14, 2021
Accepted: November 27, 2021
Published: November 30, 2021

Abstract

Deep learning has recently attracted a lot of attention with the aim of developing a fast, automatic and accurate system for image identification and classification. In this work, the focus was on transfer learning and evaluation of state-of-the-art VGG16 and 19 deep convolutional neural networks for fire image classification from fire images. In this study, five different approaches (Adagrad, Adam, AdaMax, Nadam and Rmsprop) based on the gradient descent methods used in parameter updating were studied. By selecting specific learning rates, training image base proportions, number of recursion (epochs), the advantages and disadvantages of each approach are compared with each other in order to achieve the minimum cost function. The results of the comparison are presented in the tables. In our experiment, Adam optimizers with the VGG16 architecture with 300 and 500 epochs tend to steadily improve their accuracy with increasing number of epochs without deteriorating performance. The optimizers were evaluated on the basis of their AUC of the ROC curve. It achieves a test accuracy of 96%, which puts it ahead of other architectures.

Keywords

Image Classification, Optimizers, Transfer Learning, VGG16, VGG19

1. Introduction

The increasing prevalence of surveillance of industry, public spaces and the environment in general with the help of video surveillance systems is necessary for

the security of goods, pollution, fires etc. Indeed, in recent years, many works related to fire detection by analysis and processing of video images have flooded the literature. However, the fire detection problem does not seem to have a universal answer as evidenced by some work [1]-[6]. The major difficulty in our sense of the detection of fires is related to different factors; it is for example the detection of smoke often confused with clouds and fog, the detection of flames confused with tricolor lights, vehicle lights as well as fireworks and fluorescence phenomena. The work of [7], a direct precursor to this study, explored fire detection and localization based on both full-format binary fire detection and superpixels based on similar experimentally defined CNN architectural variants derived from inceptionV1 [8] and AlexNet architectures [9]. InceptionV1-On Fire achieved 89% detection accuracy for superpixel-based detection, while Fire-Net achieved 93% for full frame binary fire detection [7]. In this paper, we show that it is possible to obtain fire detection results comparable to recent work on time dependence [10] [11] [12], by moving beyond the earlier non-temporal approach of Chenebert *et al.* [6] and using a CNN model. The main objective of this work is to implement a classification method to detect the presence of a fire (or presence of fire) in order to avoid damage caused by fires on a large scale and with precision. Thus, in this work, we implemented a classification method based on convolutional neural networks which are VGGnet (VGG16 and 19) using transfer learning. We have classified our image learning base into two classes, namely the fire and non-fire class. For this purpose, different parameters of convolutional neural networks have been dynamically explored in order to optimize the classification or detection of fire. These include the learning rate, the proportions of the training image base, the number of recursions (epochs) and the optimization algorithms (Adagrad, Adam, Adamax, Nadam and RMSProp). We used the AUC of the ROC curve to evaluate our classification methods which resulted in a better classification rate of 96%.

2. Methodology

2.1. Dataset and Operating Protocol

The work of this paper being based on fire detection we have set up a database of images divided into three parts (train, test, validation). The image base used for our study consists of 20,000 images; the image base contains 10,000 fire images and 10,000 non-fire images. Our images have dimensions of 150 × 150 pixels for VGGnet networks. Data normalization is performed by dividing all pixel values by 255 to make them compatible with the initial network values. The percentages used in this work are 60% and 40%; 70% and 30%; 80% and 20%. The choice of split ratio is based on previous work: 60% and 40% [13], 70% and 30% [14] [15] [16] and 80% and 20% [17].

The dataset (**Figure 1**) was tested on VGG networks with five gradient descent optimizers. The VGG16 and VGG19 networks have been used in many research works [18] [19] [20] [21] and have given excellent results. Several works

Figure 1. Example of images available in the dataset.

have been devoted to the study of different optimizers [22] [23] to evaluate the performance and convergence of the models created. In the following sections we will explain the different architectures of VGGnet networks and how the different optimizers used work.

2.2. VGGnet Model

The VGG network architecture was initially proposed by Simonyan and Zisserman [24]. The 16-layer (VGG16) and 19-layer (VGG19) VGG architecture served as the basis for their submission to the ImageNet Challenge 2014, where the Visual Geometry Group (VGG) team secured first and second place in the location and identification tests.

2.2.1. VGG 16

VGG16 consists of five convolution blocks, where the first block contains two convolution layers, stacked together with 64 filters. The second block consists of two convolution layers stacked with 128 filters, where the second convolution block is separated from the first block by a max pooling layer. The third block consists of three convolution layers, stacked together with 256 filters and separated from the second block by another pool max layer. The fourth and fifth layers have the same architecture, but have 512 filters. The convolution filter used in this network is of size 3×3 and stride 1. Then, a flattening layer is added between the convolution blocks and the dense layers, converting the 3D vector into a 1D vector. The last block consists of three dense layers, each with 4096 neurons, to classify each image. The last layer is a softmax layer, which is used to ensure that the sum of the probabilities of the output is equal to one. ReLU was used as an activation layer throughout the network.

2.2.2. VGG19

The VGG19 architecture is structured starting with five blocks of convolutional layers followed by three fully connected layers. Convolutional layers use 3×3

cores with a stride of 1 and a fill of 1 to ensure that each activation map maintains the same spatial dimensions as the previous layer. An activation by a rectified linear unit (ReLU) [25] is performed immediately after each convolution and a max pooling operation is sometimes used to reduce the spatial dimension. Max pooling layers use 2 × 2 cores with a stride of 2 and no fill to ensure that each spatial dimension of the previous layer's activation map is divided by two. Two fully connected layers with 4096 ReLU enabled units are then used before the final layer of 1000 fully connected softmax layers. Convolutional blocks can be considered as feature extraction layers. The activation maps generated by these layers are called bottleneck features.

2.3. Optimizers

The model learns (trains) on a dataset by comparing the actual label of the input (available in the training set) to the predicted label, thus minimizing the cost function. Hypothetically, if the cost function is zero, the model has learned the dataset correctly. However, an optimization algorithm is needed to reach the minimum of a cost function. The following section discusses the different optimization algorithms introduced in the literature to minimize the cost function.

2.3.1. Adaptive Gradient Descent Optimizers (AdaGrad)

To scale the learning rate of each weight, the AdaGrad optimization algorithm [26] was introduced to set different updates for the different weights. It performs smaller updates for settings associated with features that occur frequently, and larger updates for settings associated with features that occur seldom. For brevity, we use g_t to denote the gradient at time step t. $g_{t,i}$ is then the partial derivative of the objective function with respect to the parameter θ_i at time step t, η is the learning rate, and ∇_θ is the partial derivative of the loss function $J(\theta_i)$.

$$g_{t,i} = \nabla_\theta J(\theta_i)$$

In its update rule, AdaGrad changes the general learning rate η at each time step t for each parameter θ_i based on past gradients for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \varepsilon}} g_{t,i}$$

where G_t is the sum of the squares of the previous gradients with respect to all parameter θ.

2.3.2 Adaptive Moment Estimation (Adam)

Adam can be considered as a combination of RMSprop and stochastic gradient descent with momentum. Adam calculates the adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past square gradients v_t like Adadelta and RMSprop, Adam also maintains an exponentially decaying average of past gradients m_t, similar to momentum.

The hyperparameters $\beta_1, \beta_2 \in [0,1]$ control the exponential decay rates of these moving averages. We compute the decaying averages of the past squared

gradients, m_t and v_t respectively as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. The author found that the first- and second-order momentum is very small during the initial training, close to 0, because the β value is large, so the author recalculates a deviation to correct for it:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

where t represents its t-th power, so at the beginning of training, the learning rate can be corrected by dividing by $(1-\beta)$. When training for several rounds, the denominator part is also close to 1. Back to the original equation, so the final equation for updating the total gradient is:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

The default value of β_1 is 0.9, the default value of β_2 is 0.999, and ε is 10^{-8}. Experience shows that Adam performs very well in practice. It has more advantages than other adaptive learning algorithms [27].

2.3.3. Adaptive Moment Estimation Extension (AdaMax)

The AdaMax algorithm [28] is an extension of the Adam algorithm based on an infinite norm. In Adam's algorithm, the factor v_t in the update rule scales the gradient inversely to the l_2 norm of the past gradients (v_{t-1}) and the current gradient $|g_t|^2$.

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)|g_t|^2$$

The generalization of the l_2 standard to the l_p standard provides:

$$v_t \leftarrow \beta_2^P v_{t-1} + (1-\beta_2^P)|g_t|^P$$

The authors of AdaMax [29] prove that v_t with l_∞ converges to a more stable value.

To avoid confusion with Adam, u_t is used to denote the v_t norm at infinity:

$$u_t \leftarrow \beta_2^{\infty} v_{t-1} + \left(1-\beta_2^{\infty}\right)\left|g_t\right|^{\infty}$$

$$u_t \leftarrow \max\left(\beta_2 \cdot v_{t-1}, \left|g_t\right|\right)$$

The resulting AdaMax update rule is as follows:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{u_t}\hat{m}_t$$

We see that u will not be 0, so there is no need to add an ε to the denominator as Adam. Usually, the default parameter size is:

$$\eta = 0.002,\ \ \beta_1 = 0.9,\ \ \beta_2 = 0.999$$

2.3.4. Nesterov-Accelerated Adaptive Moment (Nadam)

Nadam [30] is an extension of Adam's algorithm by combining it with Nesterov's momentum gradient descent. The Nadam gradient update equation is finally obtained as follows:

$$g_t = \frac{\partial(Loss)}{\partial(\theta_t)}$$

$$m_t = \beta_1 m_{t-1} + \left(1-\beta_1\right) g_t$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{\hat{v}_t + \varepsilon}}\left(\beta_1 \hat{m}_t + \frac{\left(1-\beta_1\right) g_t}{1-\beta_1^t}\right)$$

where η is the hyperparameter of the learning rate. While, β_i is used to select the amount of information needed from the previous update, where $\beta_i \in [0,1]$, m_t is the first moment.

2.3.5. Root Mean Square Propagation Algorithm (RMSProp)

The main drawback of AdaGrad is that the learning rate decreases monotonically because each added term is positive. After many epochs, the learning rate is so low that it stops updating the weights. RMSProp was introduced to solve the problem of monotonic learning rate decay [31].

$$E\left[g^2\right]_t = \gamma E\left[g^2\right] t_{-1} + \left(1-\gamma\right) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \varepsilon}} g_t$$

RMSprop also divides the learning rate by an exponentially decreasing average of the gradients squared. Hinton suggests γ should be set to 0.9, while a good default value for the learning rate η is 0.001.

2.4. Overcoming Overfitting

Overfitting usually involves storing the training dataset and usually results in poor performance on the test dataset. This means that the performance on the training set can be excellent, but the performance on the test dataset is quite

poor. The loss of network generalization capability can be due to many factors, such as the capacity of the network or the nature of the training dataset itself. Techniques have been introduced in the literature to overcome overfitting (Dropout, Image Augmentation…).

2.5. Fire Detection Using a Reformed VGGNet Model

This paper uses a fire detection method based on a deep learning convolutional neural network model by proposing a reformed VGG model. The transfer learning method, the pre-trained model parameters optimize the convolution layer model parameters and solve the fire presence classification problem. Therefore, it is proposed to use the GlobalAveragePooling function which is a methodology used for a better representation of our vector instead of the Flatten layer function.

Our model mainly uses transfer learning to transfer the parameters of the VGG-16 and VGG-19 pre-trained models to the convolution layer, pooling layer, and fully connected layer of the fire detection model, and replaces the original with a 2-label Softmax classification layer, and fits a classification model with good accuracy, as shown in Figure 2.

The main operating process is as follows:

1) Enter an example of a fire image. The images are extracted from the fire positive and negative image base as a training sample set for input.

2) Pre-processing: All fire and non-fire images were collected in a dataset and loaded to be scaled to a fixed resolution size of 150 × 150 pixels, to be suitable for further processing in the deep learning pipeline.

3) Build new improved models: using the VGGNet model (VGG16 and VGG19), the FC layers are optimized with reduced parameters, replaced the Softmax classification layer of the original model with a two labels Softmax classifier.

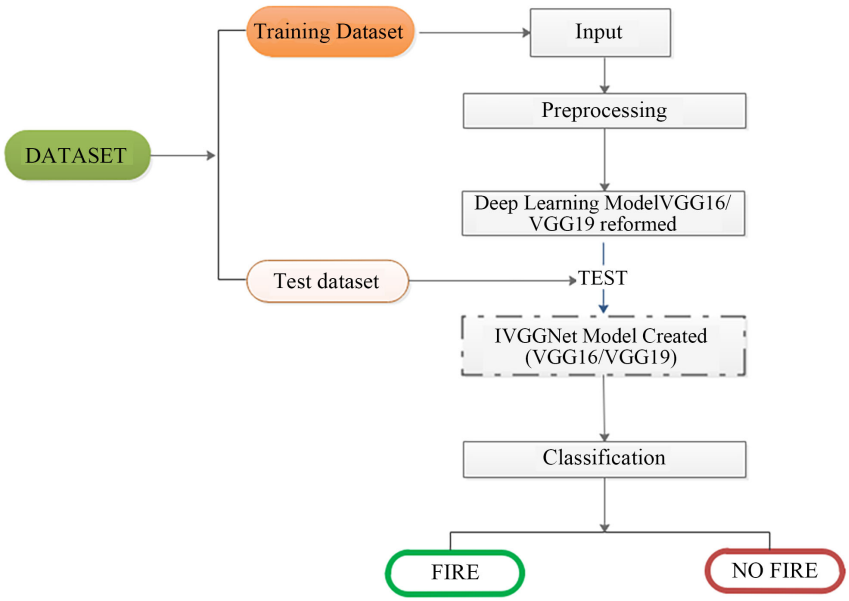

Figure 2. The proposed method.

4) Transfer learning: Using the parameters of the 13 (VGG16) and 16 (VGG19) convolutional layers and pooling layers of the VGGNet pre-trained model, the parameters of the detection model were optimized with transfer learning.

5) Model training: In order to start the training phase of one of the two selected and/or tuned deep learning models, the preprocessed dataset is divided into 60% - 40%, 70% - 30% and 80% - 20%. This means that 40%, 30% and 20% of dataset will be used for the test phase. To train and optimize the parameters of the pooling layers and Softmax layers, it is necessary to freeze the parameters of the convolutional layers and their pooling layers and to initialize the parameters of the model using an elaborate approach, to define the different optimizers chosen, the learning rate, the number of epochs in order to obtain a good accuracy with the taking into account of the overfitting.

6) Classification: In the last step of the proposed model, the test data is fed to the deep learning classifier set to classify all image patches into one of two classes: presence of fire or no fire, as shown in Figure 2. At the end of the workflow, the overall performance analysis for each deep learning classifier will be evaluated based on the metrics described in the following section.

2.6. Model Evaluation Criteria

The evaluation of the fire detection model can be assessed from the effect and reliability. These two indicators are generally precision and time to test, while the former includes three indicators: precision, recall rate and error rate. Among them, precision is defined as (1), and recall is defined as (2). The error rate is defined as (3).

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{1}$$

$$\mathrm{recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2}$$

$$\text{Error rate} = \frac{\mathrm{FP} + \mathrm{FN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} = 1 - \mathrm{recall} \tag{3}$$

Among them, TP is the number of positive samples determined by the model to be positive, TN is the number of negative samples determined by the model to be truly negative, and FP is the number of positive samples determined by the model. However, the actual number of negative samples is FN, which is the number of negative samples determined by the model but actually positive, as shown in Table 1.

Table 1. Definition of the parameters TP, TN, FP and FN.

	Positive	Negative
True	TP	TN
False	FP	FN

3. Experimentation and Results of the Proposed Approach

The following section details the results obtained when training the two network architectures using five optimizers selected with three learning rates, namely 10^{-3}, 10^{-4} and 10^{-5}. Many experiments have been carried out on datasets, in order to determine the behavior of each optimizer with each network architecture to determine the best possible combination. The performances of each optimizer with the VGG16 architecture are presented in Table 2, Table 4 and those of VGG19 in Table 3, Table 5. The percentages used for the training and validation data are 60% - 40%, 70% - 30% and 80% - 20% respectively. The AUC of the ROC curve measures performance. Optimizers are ranked on the basis of their validation AUC. Images were held constant at 150 × 150; a batch size of 32 images was used, and with a number of epochs between 300 and 500.

3.1. Result of the VGG16 Architecture for 300 Epochs

Table 2 shows the results of the VGG16 architecture for 300 epochs and taking into account the partition of the different datasets (60%/40%, 70%/30% and 80%/20%), we note that the highest AUC is 96%, was obtained by the Adam optimizer with a ratio of 60%/40% for the highest learning rate 10^{-3}, to achieve convergence. At the same time, the lowest AUC 49.81% was obtained by the AdaGrad optimizer which did not converge at all with the 80%/20% ratio. For the average learning rate 10^{-4}, the RMSprop optimizer obtained the highest AUC with a value of 94.67% with the ratio 60%/40% and the other ratios obtained

Table 2. Results obtained with the VGG16 architecture for 300 epochs, where LR represents the learning rate and the training datasets have the following percentages: 60%/40%, 70%/30% and 80%/20% to train and validate the dataset.

OPTIMIZERS	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
AdaGrad_60_40	85.71%	56.38%	51.21%
Adam_60_40	**96.00%**	93.38%	**89.21%**
Adamax_60_40	95.58%	91.92%	82.67%
Nadam_60_40	94.50%	93.46%	**89.21%**
RMSprop_60_40	95.25%	**94.67%**	87.92%
AdaGrad_70_30	87.39%	59.57%	50.50%
Adam_70_30	**95.71%**	93.89%	87.96%
Adamax_70_30	94.93%	92.39%	85.39%
Nadam_70_30	95.11%	**93.93%**	**89.29%**
RMSprop_70_30	95.50%	93.57%	88.21%
AdaGrad_80_20	49.81%	55.56%	50.44%
Adam_80_20	**95.69%**	94.47%	**89.22%**
Adamax_80_20	94.94%	93.72%	85.09%
Nadam_80_20	95.38%	**94.59%**	88.88%
RMSprop_80_20	94.72%	93.75%	89.00%

Table 3. Results obtained with the VGG19 architecture for 300 epochs, where LR represents the learning rate and the training datasets have the following percentages: 60%/40%, 70%/30% and 80%/20% to train and validate the dataset.

OPTIMIZERS	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
AdaGrad_60_40	87.12%	57.21%	50.75%
Adam_60	95.13%	94.04%	**89.88%**
Adamax_60	95.21%	92.25%	86.29%
Nadam_60	**95.58%**	**94.25%**	89.04%
RMSprop_60	94.88%	93.25%	89.38%
AdaGrad_70	88.50%	59.68%	51.82%
Adam_70	94.00%	93.93%	90.36%
Adamax_70	94.93%	91.93%	86.25%
Nadam_70	**95.39%**	93.54%	89.29%
RMSprop_70	94.36%	**94.29%**	**90.50%**
AdaGrad_80	88.03%	63.63%	57.13%
Adam_80	94.97%	93.81%	89.78%
Adamax_80	94.22%	92.56%	87.81%
Nadam_80	**95.44%**	**94.69%**	**90.19%**
RMSprop_80	94.91%	94.28%	89.00%

lower values. The AdaGrad optimizer obtained the lowest AUC with a value of 55.56%. For the lowest learning rate 10^{-5}, the Nadam optimizer with the ratio 70%/30% obtained the highest AUC value 89.29% and the AdaGrad optimizer obtained the lowest AUC 50.44%. Overall, the highest learning rate 10^{-3} gave the best results, followed by the medium learning rate 10^{-4}. Throughout our experiments the AdaGrad optimizer achieved the lowest AUC value for all learning rates and ratios used.

3.2. Result of the VGG16 Architecture for 500 Epochs

Table 4 shows the results of the VGG16 architecture for 500 epochs and taking into account the partition of the different datasets (60% - 40%, 70% - 30% and 80% - 20%). For the highest learning rate 10^{-3}, it shows that the highest AUC value is 95.43% was obtained by the Adam optimizer with a ratio of 70%/30%. At the same time, the lowest AUC value of 51.14% was obtained by the AdaGrad optimizer which did not converge at all with the 70%/30% ratio for the learning rate of 10^{-5}. For the average learning rate 10^{-4}, the Nadam optimizer obtained the highest AUC value of 95.04% with the ratio 70%/30%. The AdaGrad optimizer obtained the lowest AUC value 59.25%. For the lowest learning rate 10^{-5}, the Adam optimizer with the 80%/20% ratio obtained the highest AUC 91.44%. Overall, the highest learning rate 10^{-3} gave the best results, followed by the medium learning rate 10^{-4} and the lowest values were obtained with the learning rate 10^{-5}. Throughout our experiments the AdaGrad optimizer achieved the lowest AUC value for all learning rates and ratios used.

Table 4. Results obtained with the VGG16 architecture for 500 epochs, where LR represents the learning rate and the training datasets have the following percentages: 60% - 40%, 70% - 30% and 80% - 20% to train and validate the dataset.

OPTIMIZERS	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
AdaGrad_60	86.83%	66.71%	53.83%
Adam_60	**94.83%**	**94.96%**	**89.33%**
Adamax_60	94.75%	94.00%	85.62%
Nadam_60	94.67%	93.88%	**89.33%**
RMSprop_60	94.33%	94.08%	**89.33%**
AdaGrad_70	85.39%	59.25%	51.14%
Adam_70	**95.43%**	94.07%	**90.43%**
Adamax_70	95.32%	92.86%	87.25%
Nadam_70	94.32%	**95.04%**	90.21%
RMSprop_70	94.93%	94.79%	90.29%
AdaGrad_80	87.63%	65.31%	56.03%
Adam_80	94.88%	94.47%	**91.44%**
Adamax_80	94.41%	92.72%	87.53%
Nadam_80	95.19%	**94.84%**	91.16%
RMSprop_80	**95.34%**	94.34%	91.28%

3.3. Result of the VGG19 Architecture for 300 Epochs

Table 3 shows the results of the VGG19 architecture for 300 epochs and taking into account the partition of the different datasets (60%/40%, 70%/30% and 80%/20%), we note that the highest AUC value is 95.58% was obtained by the Nadam optimizer with the ratio of 60%/40% with the highest learning rate 10^{-3}. Jointly, the lowest AUC value is 50.75% was obtained by the AdaGrad optimizer which did not converge at all with the 60%/40% ratio with the lowest learning rate 10^{-5}. For the average learning rate 10^{-4}, the Nadam optimizer obtained the highest AUC value 94.69% with the ratio 80%/20%. The Adagrad optimizer obtained the lowest AUC 57.21%. For the lowest learning rate 10^{-5}, the RMSprop optimizer obtained the highest AUC value 90.50% for the ratio 70% - 30%. Overall, the highest learning rate 10^{-3} gave the best results, followed by the medium learning rate 10^{-4}. Nadam optimizers gave the highest values with learning rate 10^{-3} for the three ratios in our work. For the 10^{-4} learning rate, the Nadam optimizers obtained the highest AUC values for the 60% - 40% and 80% - 20% partitions. The RMSprop optimizer obtained the high value for the 70% - 30% partition. Throughout our experiments the AdaGrad optimizer obtained the lowest AUC value for all learning rates and ratios used.

3.4. Result of the VGG19 Architecture for 500 Epochs

Table 5 shows the results of the VGG19 architecture for 500 epochs, which shows that the highest AUC value is 95.61% which was obtained by the Nadam

Table 5. Results obtained with the VGG19 architecture for 500 epochs, where LR represents the learning rate and the training data sets have the following percentages: 60%/40%, 70%/30% and 80%/20% to train and validate the dataset.

OPTIMIZERS	LR = 10^{-3}	LR = 10^{-4}	LR = 10^{-5}
AdaGrad_60	88.21%	63.88%	48.96%
Adam_60	94.79%	93.50%	91.12%
Adamax_60	89.38%	93.50%	86.71%
Nadam_60	94.79%	93.50%	90.75%
RMSprop_60	94.63%	93.54%	90.25%
AdaGrad_70	88.50%	62.50%	50.25%
Adam_70	94.86%	94.18%	90.79%
Adamax_70	94.08%	92.89%	88.64%
Nadam_70	95.61%	95.11%	90.39%
RMSprop_70	95.18%	94.00%	91.68%
AdaGrad_80	88.25%	65.75%	49.59%
Adam_80	95.11%	94.34%	91.31%
Adamax_80	94.16%	93.09%	89.22%
Nadam_80	94.34%	94.81%	91.53%
RMSprop_80	94.84%	93.78%	90.25%

optimizer with a ratio of 70% - 30% with the highest learning rate 10^{-3}. At the same time, the lowest AUC value 48.96% was obtained by the AdaGrad optimizer which did not converge at all with the 60% - 40% ratio for the lowest learning rate 10^{-5}. For the average learning rate 10^{-4}, the Nadam optimizer obtained the highest AUC value of 95.11% with the ratio 70% - 30%. The AdaGrad optimizer obtained the lowest AUC 62.50% with the ratio 70% - 30%. For the lowest learning rate 10^{-5}, the RMSprop optimizer achieved the highest AUC 91.68% for the ratio 70%/30%. Overall, the highest learning rate 10^{-3} gave the best results, followed by the medium learning rate 10^{-4}. Throughout our experiments the AdaGrad optimizer achieved the lowest AUC value for all learning rates and ratios used.

3.5. Discussion

At the end of our experimental approach and the results obtained, it is possible to draw some interesting analyses on the behavior of the CNNs, the number of epochs and the optimizers studied in this work. Taking into account the choice of optimizer, the number of epochs, the ratios used and the relationship with the learning rate, the experimental results confirm that the choice of learning rate and the ratios used can lead to an unstable behavior of the training process. This is particularly evident, for some of the networks and optimizers considered, when considering the smallest learning rate used in the experiments. As we can see, when LR = 10^{-5}, the learning process of the VGG16 and VGG19 networks

gave low performance value with the optimizers and the ratios studied to see a poor performance of the model. However, when we use a high learning rate, *i.e.* LR = 10^{-3}, we obtain the highest values of our experiment whatever the CNNs used. For the average learning rate LR = 10^{-4}, we obtain lower values and some close to those obtained with the learning rate LR = 10^{-3}. As explained in the previous sections, this can be motivated by the fact that the weights of the network change abruptly from one epoch to another. The transition to higher LR values allows the convergence of the formation process in all studied configurations. Overall, the results do not match the theoretical expectations: a lower LR value allows a smoother convergence, but it requires more time compared to a higher LR value. Another interesting observation concerns the importance of hyperparameters. While this is a topic of fundamental importance in the field of deep learning, it is particularly evident in the results of the experimental phase. In particular, all studied architectures produced comparable performance when the best configuration between the learning rate and the optimizer (which is different for each architecture type) was considered. In other words, it seems that the choice of hyperparameters not only plays a critical role in determining the performance of the model, but the examined CNNs are indistinguishable in terms of performance. Our results confirm those of Sharma and Venugopalan [32]. We think this is an interesting observation that should further emphasize the importance of hyperparameter setting.

Focusing on the optimizers and the different percentages used (60%/40%, 70%/30% and 80%/20%), the Adam optimizer produced the best performance with the 10^{-3} learning rate for VGG16 and Nadam produced also the best performance for VGG19. Conversely, Adam, Nadam and RMSprop obtained the best performance on the considered CNNs when LR = 10^{-5} (except Nadam and RMSprop on the VGG19 architecture, where the best performance is obtained with LR = 10^{-4}). Overall, the best result on the considered dataset was obtained by the Adam optimizer and the VGG16 network. However, the differences in performance between the best configurations of each network are not statistically significant. Overall, each optimizer behaved differently depending on the architecture considered. For example, for VGG16 architectures, Adam outperformed Nadam and RMSprop. For VGG19 architectures, Nadam outperformed Adam and RMSprop.

Given a specific network, each optimizer requires a different time to converge (*i.e.* to conclude the defined number of epochs). In particular, Adam was the optimizer that gave the highest AUC value for VGG16 and Nadam also gave a high value for VGG19, whether we use VGG16 or VGG19, the AdaGrad optimizer gave poor convergence results. This result is consistent with that proposed by Lydia and Francis [33], in which the authors studied some alternatives and hyperparameters to improve the performance of Gradient Descent algorithms. The authors explained that good performance can be achieved with optimizers if they are trained on different image data sets. The best performing network obtained with transfer learning (VGG16 architecture, with the Adam optimizer, 60% -

40% ratio and a learning rate of 10^{-3}) was able to obtain an AUC value of 96%.

4. Conclusion

In this paper, a comparative effect of five optimization algorithms on three proportions of image classification datasets and three learning rates using two convolutional neural network architectures (VGG16 and VGG19) has been performed. Our results reveal that the performance of each optimizer varies with each proportion of the dataset used, the learning rate, and the number of recursions (epochs), confirming the effect of hyperparameters on the performance of different optimizers. Based on several experiments conducted, our results show that Adam exhibited superior and robust performance on VGG16 networks for 300 epochs compared to other optimization algorithms. Similarly, Nadam also showed superior performance for VGG19 for 300 epochs. This same result was observed with VGG16 for 500 epochs and VGG19 for 500 epochs. These results show that the Adam and Nadam optimizers are apparently suitable for the different dataset and models examined in this study. In this study, two neural network models and three percentages of image classification data were used to perform all our experiments. It will be interesting to use more than three proportions of data in different domains and experiment with the comparative effects of these optimizers on a number of different CNN architectural designs and deep learning models in order to achieve better generalization. This could be the subject of future projects.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Healey, G., Slater, D., Lin, T., Drda, B. and Goedeke, A. (1993) A System for Real-Time Fire Detection. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, New York, 15-17 June 1993, 605-606. https://doi.org/10.1109/CVPR.1993.341064

[2] Phillips, W., Shah, M. and da Vitoria Lobo, N. (2002) Flame Recognition in Video. *Pattern Recognition Letters*, **23**, 319-327. https://doi.org/10.1016/S0167-8655(01)00135-0

[3] Liu, C. and Ahuja, N. (2004) Vision Based Fire Detection. *Proceedings of International Conference on Pattern Recognition*, Vol. 4, Cambridge, 26 August 2004, 134-137. https://doi.org/10.1109/ICPR.2004.1333722

[4] Toreyin, B., Dedeoglu, Y. and Cetin, A. (2005) Flame Detection in Video Using Hidden Markov Models. *IEEE International Conference on Image Processing* 2005, Genova, 14 September 2005, II-1230. https://doi.org/10.1109/ICIP.2005.1530284

[5] Ko, B.C., Cheong, K.H. and Nam, J.Y. (2009) Fire Detection Based on Vision Sensor and Support Vector Machines. *Fire Safety Journal*, **44**, 322-329. https://doi.org/10.1016/j.firesaf.2008.07.006

[6] Chenebert, A., Breckon, T.P. and Gaszczak, A. (2011) A Non-Temporal Texture

Driven Approach to Real-Time Fire Detection. 2011 18*th IEEE International Conference on Image Processing*, Brussels, 11-14 September 2011, 1741-1744. https://doi.org/10.1109/ICIP.2011.6115796

[7] Dunnings, A.J. and Breckon, T.P. (2018) Experimentally Defined Convolutional Neural Network Architecture Variants for Non-Temporal Real-Time Fire Detection. 2018 25*th IEEE International Conference on Image Processing* (*ICIP*), Athens, 7-10 October 2018, 1558-1562. https://doi.org/10.1109/ICIP.2018.8451657

[8] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) Going Deeper with Convolutions. 2015 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Boston, 7-12 June 2015, 1-9. https://doi.org/10.1109/CVPR.2015.7298594

[9] Krizhevsky, A., Sutskever, I. and Hinton, G. (2012) ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C.J.C., Bottou, L. and Weinberger, K.Q., Eds., *Advances in Neural Information Processing Systems*, Vol. 25, Curran Associates, Inc., Red Hook, 1097-1105.

[10] Bohush, R. and Brouka, N. (2013) Smoke and Flame Detection in Video Sequences Based on Static and Dynamic Features. 2013 *Signal Processing*: *Algorithms*, *Architectures*, *Arrangements*, *and Applications* (*SPA*), Poznan, 26-28 September 2013, 20-25.

[11] Morerio, P., Marcenaro, L., Regazzoni, C.S. and Gera, G. (2012) Early Fire and Smoke Detection Based on Colour Features and Motion Analysis. 2012 19*th IEEE International Conference on Image Processing*, Orlando, 30 September-3 October 2012, 1041-1044. https://doi.org/10.1109/ICIP.2012.6467041

[12] Luo, Y., Zhao, L., Liu, P. and Huang, D. (2017) Fire Smoke Detection Algorithm Based on Motion Characteristic and Convolutional Neural Networks. *Multimedia Tools and Applications*, **77**, 15075-15092. https://doi.org/10.1007/s11042-017-5090-2

[13] Anjli, N. and Varun, J. (2016) A First Attempt to Develop a Diabetes Prediction Method Based on Different Global Datasets. 2016 4*th International Conference on Parallel*, *Distributed and Grid Computing* (*PDGC*), Waknaghat, 22-24 December 2016, 237-241. https://doi.org/10.1109/PDGC.2016.7913152

[14] Abien, F.M.A. (2018) On Breast Cancer Detection: An Application of Machine Learning Algorithms on the Wisconsin Diagnostic Dataset. *Proceedings of the* 2*nd International Conference on Machine Learning and Soft Computing*, Phu Quoc Island, 2-4 February 2018, 5-9. https://doi.org/10.1145/3184066.3184080

[15] Vaishali, R., Sasikala, R., Ramasubbareddy, S., Remya, S. and Sravani, N. (2017) Genetic Algorithm Based Feature Selection and MOE Fuzzy Classification Algorithm on Pima Indians Diabetes Dataset. 2017 *International Conference on Computing Networking and Informatics* (*ICCNI*), Lagos, 29-31 October 2017, 1-5. https://doi.org/10.1109/ICCNI.2017.8123815

[16] Binh, T.P., Dieu, T.B., Hamid, R.P., Prakash, I. and Dholakia, M.B. (2015) Landslide Susceptibility Assesssment in the Uttarakhand Area (India) Using GIS: A Comparison Study of Prediction Capability of Naïve Bayes, Multilayer Perceptron Neural Networks, and Functional Trees Methods. *Theoretical and Applied Climatology*, **128**, 255-273. https://doi.org/10.1007/s00704-015-1702-9

[17] Ibrahem, K., Mauro, C. and Aleš, P. (2020) Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images. *Journal of Imaging*, **6**, Article No. 92. https://doi.org/10.3390/jimaging6090092

[18] Manali, S. and Meenakshi, P. (2018) Transfer Learning for Image Classification.

Proceedings of the 2*nd International Conference on Electronics*, *Communication and Aerospace Technology* (*ICECA* 2018), Coimbatore, 29-31 March 2018, 656-660. https://doi.org/10.1109/ICECA.2018.8474802

[19] Xiao, J., Wang, J., Cao, S. and Li, B. (2020) Application of a Novel and Improved VGG-19 Network in the Detection of Workers Wearing Masks. *Journal of Physics*: *Conference Series*, **1518**, Article ID: 012041. https://doi.org/10.1088/1742-6596/1518/1/012041

[20] Srikanth, T. (2019) Transfer Learning Using VGG-16 with Deep Convolutional Neural Network for Classifying Images. *International Journal of Scientific and Research Publications*, **9**, 143-150.

[21] Manuel, L. and Elena, G. (2017) Transfer Learning for Illustration Classification. *Spanish Computer Graphics Conference* (*CEIG*), Sevilla, 28-30 June 2017, 1-9.

[22] Saad, H.H. and Adnan, M.A. (2021) Comparison of Optimization Techniques Based on Gradient Descent Algorithm: A Review. *Palarch's Journal of Archaeology of Egypt/Egyptology*, **18**, 2715-2743.

[23] Ersan, Y. and Fatih, T. (2017) Comparison of the Stochastic Gradient Descent Based Optimization Techniques. *International Artificial Intelligence and Data Processing Symposium* (*IDAP*), Malatya, 16-17 September 2017, 1-5.

[24] Simonyan, K. and Zisserman, A. (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.

[25] Nair, V. and Hinton, G.E. (2010) Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the* 27*th International Conference on Machine Learning* (ICML-10), Haifa, 21-24 June 2010, 807-814.

[26] Duchi, J.C., Hazan, E. and Singer, Y. (2011) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, **12**, 2121-2159.

[27] Kingma, D.P. and Ba, J. (2017) Adam: A Method for Stochastic Optimization. arXiv:1412.6980v9.

[28] Kingma, D.P. and Ba, J. (2001) Adam: A Method for Stochastic Optimization. arXiv:1412.6980. https://arxiv.org/abs/1412.6980

[29] Kingma, D.P. and Ba, J. L. (2015) Adam: A Method for Stochastic Optimization. 3*rd International Conference on Learning Representations*, San Diego, 7-9 May 2015, 1-13.

[30] Dozat, T. (2016) Incorporating Nesterov Momentum into Adam. *Proceedings of the* 4*th International Conference on Learning Representations*, *Workshop Track*, San Juan, Puerto Rico, 2-4 May 2016, 1-4.

[31] Hinton, G., Srivastava, N. and Swersky, K. (2020) Neural Networks for Machine Learning, Lecture 6a Overview of Mini-Batch Gradient Descent.

[32] Sharma, B. and Venugopalan, K. (2014) Comparison of Neural Network Training Functions for Hematoma Classification in Brain CT Images. *OSR Journal of Computer Engineering*, **16**, 31-35.

[33] Lydia, A. and Francis, S. (2019) Adagrad—An Optimizer for Stochastic Gradient Descent. *International Journal of Information and Computing Science*, **6**, 566-568.