ICIAP 2023 – 22nd International Conference On Image Analysis and Processing

FireNetAGM: A Deep Learning Approach for Real-Time Fire Detection from Videos Acquired by Fixed CCTV cameras using MobileNetV2 and LSTM

Group 09: Andrea Vincenzo Ricciardi(2009), Giulia Minichiello(2127) and Martina Iannaccone(2112)

Dept. of Information Eng., Electrical Eng. and Applied Mathematics
University of Salerno, Italy
{a.ricciardi38, g.minichiello9, m.iannaccone43}@studenti.unisa.it

Abstract. Real-time fire detection from image sequences is a crucial requirement in video surveillance applications, enabling the prevention of environmental disasters and ensuring continuous monitoring of urban and forest areas. To meet this demand, there is a growing interest in deploying "intelligent cameras" equipped with onboard video-analytics algorithms capable of detecting fires, such as flames and smoke, in real-time. These cameras are typically deployed in remote locations and need to be self-sufficient in terms of computational resources, with a small embedded system capable of processing the image sequence using a fire detection algorithm. In this context, our work presents a novel network model called **FireNetAGM**, which leverages the MobileNetV2 architecture for feature extraction from a sequence of video frames and utilizes an LSTM architecture to handle the temporal dynamics of the video.

Keywords: Convolutional Neural Networks, Deep Learning (DL), Recurrent Networks, Computer Vision, Image Processing, Fire Detection.

1 Introduction

Fires pose a significant threat to the safety of human lives, property, and the environment. Detecting fires early and accurately is crucial to prevent the devastating consequences they can cause. Conventional fire detection methods heavily rely on specialized sensors such as temperature detectors, smoke detectors, and flame detectors. However, these sensors have limitations, such as restricted coverage areas, delayed response times, and limited accessibility for the general public.

In recent years, remarkable advancements in image processing and computer vision have brought about a paradigm shift in fire detection. By harnessing the power of deep learning algorithms and leveraging computer vision techniques, we can now detect fires using cameras as the primary equipment for fire detection. This innovative approach offers a wide range of advantages over traditional ground-based tools.

1.1 Contribution

In this context, our work aims to address the challenge of real-time fire detection using deep learning techniques. We propose an approach that combines the strengths of MobileNetV2 and LSTM architectures to analyze variations within video frames and capture temporal dependencies.

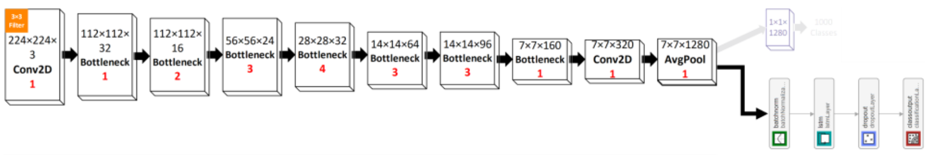


Fig. 1. Architecture of FireNetAGM. The violet-out portion shows the layer removed from the MobileNetV2 model, and the black arrow pointing downwards shows the added layers for our FireNetAGM network. Number in red denote the number of the times the block is repeated. Detailed information about the internals of the different layers (except the ones added in this proposal), expansion factors, and strides, can be obtained from [5].

2 Proposed Approach

The **FireNetAGM** model, designed for mobile and embedded applications, draws inspiration from the **MobileNetV2** architecture [1], as depicted in Fig. 1.

To optimize fire detection performance, the FireNetAGM model adopts a two-step approach. Initially, we freeze the parameters of all the layers in the **MobileNetV2** backbone. This step ensures that the pre-trained backbone focuses solely on capturing spatial information from input images and remains unchanged during training.

To enhance the model's capability to analyze video sequences and capture temporal patterns, we introduce additional layers after modifying the MobileNetV2 backbone. First, we add a **Batch Normalization** layer to normalize input data, adjusting and scaling the activations, which leads to stable and efficient training. Next, we incorporate an **LSTM** (Long Short-Term Memory) layer to analyzes the sequential nature of video frames, enabling the model to make predictions based on the historical observations. To prevent overfitting and improve generalization, a **dropout layer** is introduced after the LSTM. Dropout randomly deactivates a portion of input units during training, encouraging the model to learn more robust representations and reducing reliance on specific features. Finally, the model includes a **classifier layer** for frame-level classification, generating predictions for each frame in the video sequence to identify the presence or absence of fire. Additionally, the model produces a sequence-level prediction based on the final frame, providing a holistic understanding of the entire video sequence.

By combining the power of the pre-trained MobileNetV2 backbone for spatial feature extraction with the trainable LSTM module and classification layer for temporal analysis, the FireNetAGM model achieves a highly effective approach to fire detection. This architecture strikes a balance between utilizing pre-trained knowledge and adapting to the specific fire detection task, making it suitable for resource-constrained environments such as mobile and embedded systems.

The proposed method is discussed below in details.

2.1 Pre-Processing

The first step involves preparing the dataset for our proposed model, which includes essential preprocessing steps to ensure compatibility. We apply the preprocessing operations to all frames, regardless of the set they belong to. These transformations are necessary to meet the specific requirements of our model, which expects input images to be normalized in a particular format. To satisfy the model's criteria, the images should be organized in the form of mini-batches of 3-channel RGB images with dimensions `(3 x H x W)`, where `H` and `W` are expected to be at least `224` pixels. Prior to input, the images have to be loaded within the range of `[0, 1]` and then and normalized using the mean values of `[0.485, 0.456, 0.406]` and standard deviation values of `[0.229, 0.224, 0.225]`. Moreover, specific transformations are applied to the training set to create diverse training examples that capture different variations of fire instances. These transformations include horizontal flipping, random brightness contrast, Gaussian blur, and color jitter.

It's important to highlight that for each video, a subset of frames is selected for training, validation, and testing purposes. Specifically, for each video, the model takes `num_frames × num_frames_per_segment` images. The selection of image subsets for each set involves a different approach in calculating the start indices for the segments. In the training set, the start indices are randomly sampled but evenly distributed across the video frames. This random selection diversifies the training examples and reduces potential bias in the training data. Conversely, for the validation and test sets, the start indices are determined uniformly to ensure comprehensive coverage of the video frames.

2.2 MobileNetV2

MobileNetV2 is a lightweight deep neural network (DNN) architecture introduced by Sandler et al. in 2018 [1]. It was specifically designed for mobile platforms and processors with limited processing capabilities. It is renowned for its high accuracy and small model size, making it an ideal choice for IoT applications, including CCTV cameras [2].

The architecture of MobileNetV2, as depicted in **Fig. 2.** The Architecture of MobileNetV2.Fig. 2, consists of 2D convolutional layers, bottleneck layers and pooling layers. The *bottleneck layer*, in particular, comprises sub-layers such as expansion, normalization, activation, addition, and 3x3 depth-wise convolution. The purpose of the expansion operation is to expand the data space in terms of increasing the number of channels of the input data by a factor that is defined by the model hyperparameters. On the contrary, the projection operations decrease the amount of channels of the input data and thus, they shrink the data space. The normalization, pooling, convolution and activation layers employ the corresponding operations that are necessary for the DNN training and inference process.

The DNN training procedure requires a forward pass of the data through the neural network, which is followed by a backward propagation operation which fine-tunes the model weights according to the value of the loss function. On the contrary, the DNN inference does not include any weight tuning process of the model, and thus, it requires the input data to be propagated through the network once. As a result, the DNN inference process is considered less cost demanding in terms of computational requirements and energy consumption when compared with the DNN training. More specifically, the inference process of a trained MobileNetV2 model requires 13 million multiply-accumulate (MAC) operations and consists of 4.3 million parameters. As a result, the design and implementation of the MAC operation plays a critical role on the execution time of the MobileNetV2 inference.

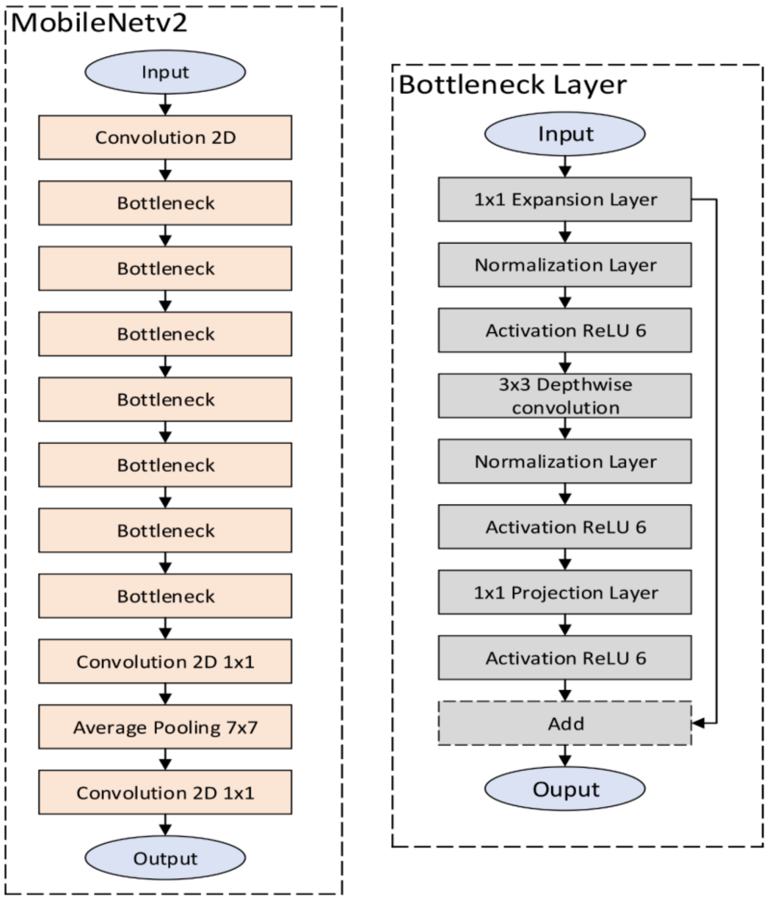


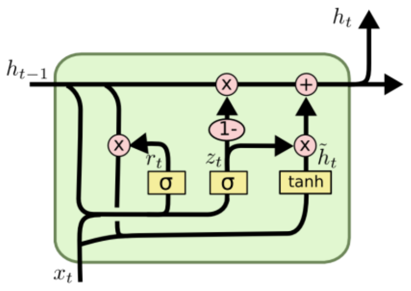
Fig. 2. The Architecture of MobileNetV2. [3]

Feature Extraction using MobileNetV2.

Firstly, the input tensor `x` is expected to have dimensions `(batch_size, timesteps, C, H, W)` where *timesteps* denotes the length of the frames' sequence. This tensor is reshaped to have the shape `(batch_size × timesteps, C, H, W)` to facilitate feature extraction. Next, the reshaped tensor is processed through the MobileNetV2 backbone, which extracts spatial features from each individual frame. Following the feature extraction, Global Average Pooling (GAP) is applied to the output tensor in order to reduce the spatial dimensions of the features while preserving the channel-wise information.

2.3 Recurrent Neural Network

The **Long Short-Term Memory** (**LSTM**) module plays a crucial role in the FireNetAGM model for the fire detection task. In point of fact, LSTM is a type of recurrent neural network (RNN) layer that is designed to capture temporal dependencies and patterns in sequential data, such as video frames.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fig. 3. Long Short-Term Memory [4]

Feature Extraction using LSTM

In the FireNetAGM model, after the spatial features are extracted by the MobileNetV2 backbone and processed through Global Average Pooling, the resulting tensor is reshaped to have the shape `(batch_size, timesteps, C, H, W)`. This reshaped tensor is then passed through the LSTM module, which consists of multiple memory cells that store information over time and perform computations based on the current input x_t and the information stored in the previous time steps h_{t-1}.

The LSTM module plays a vital role in understanding the temporal dynamics of fire-related patterns in video sequences. By analyzing the sequential nature of the data, it allows the model to capture the evolution of flames or smoke over time.

The output of the LSTM module is a new tensor containing the learned representation of the temporal patterns in the video frames. This tensor has the shape `(batch_size, timesteps, hidden_size)` where the *hidden_size* represents the number of hidden units in the LSTM layer. In our specific implementation, the hidden size is set to 128, indicating that each LSTM layer has 128 hidden units. These layers play a critical role in capturing and preserving information over time. To further enhance the model's capability in capturing complex temporal patterns, we have chosen to stack two LSTM layers, forming a stacked LSTM. This means that the output of the first LSTM layer serves as the input to the second LSTM layer, allowing for a deeper representation of the temporal patterns present in the video frames.

2.4 Image Classification

The final stage of the FireNetAGM model involves the fully connected layer (FC), which is responsible for performing the final classification. To mitigate overfitting, we applied dropout regularization to the output of the LSTM layer. This helps to reduce the risk of the model relying too heavily on specific features in the sequence representation. The resulting tensor, denotated as `x`, represents the processed sequence representation.

The FireNetAGM model performs two types of classification using this processed sequence representation:

1. **Frame-Level Classification**: The tensor `x` is passed trough a classifier that performs frame-level classification for each frame in the input sequence. A sigmoid activation function is applied to the classifier output, producing probabilities indicating the presence or absence of the target class for each frame.
2. **Sequence-Level Classification**: The last hidden state of the LSTM, which corresponds to the representation of the last frame in the input sequence, is extracted from the output tensor `x`. This last frame representation is then passed to the same classifier used for frame-level classification, which performs sequence-level classification, providing an overall prediction for the entire sequence.

During training, the FireNetAGM model is trained to minimize the loss on the output of the sequence-level classification. Specifically, the *Binary Cross Entropy with Logits* loss function is applied, which combines a sigmoid activation and a binary cross-entropy loss. This loss function is well-suited for binary classification tasks, such as fire detection, and helps the model learn to make accurate predictions at sequence level. By minimizing this loss, the model aims to improve its ability to detect and classify fire instances accurately across different sequences.

3 Dataset

The dataset provided consists of 312 videos, out of which 219 videos (positive) depict the presence of a fire (flames and/or smoke), while the remaining 93 videos do not show any sign of fire (negative). The videos in the dataset provided have been carefully selected from various existing datasets, including MIVIA, RISE, KMU, Wildfire and D-Fire. During the selection process, particular attention was given to ensuring that the positive videos truly captured the presence of a fire, rather than merely featuring controlled conditions with flames or smoke. Similarly, negative videos were chosen to include moving objects that might resemble flames or smoke, adding to the challenge of fire detection.

To mitigate the class imbalance present in the dataset, we took measures to address this issue. Instead of relying solely on the provided videos, we introduced a significant number of entirely new videos to create a more balanced dataset. These additional videos were sourced from various online datasets [5][6], as well as from a dataset specifically curated for this task. By including these new videos, we aimed to ensure that the FireNetAGM model is exposed to a wide range of fire and non-fire instances, thereby enhancing its ability to accurately detect fire in diverse real-world scenarios. This approach helps prevent bias towards the majority class and ensures that the model is well-equipped to handle both positive and negative instances effectively.

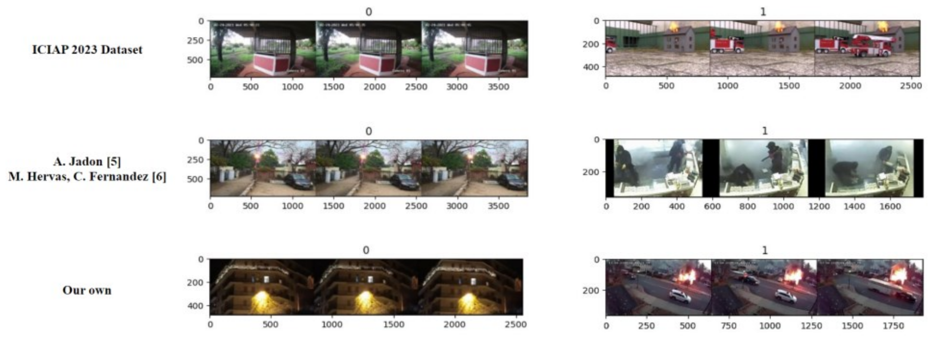


Fig. 4. Sample sequence of frames from the various dataset.

In total, the dataset used for training our network consists of 255 fire videos and 236 non-fire videos, resulting in a total of 491 videos. The Fig. 4 displays examples of frame sequence from the three types of dataset used in the study. For simplicity, three segments were selected from the video, and one frames was taken for each segment.

To process videos, we extracted frames based on their fps rate. For videos, with an fps rate lower than 10, we sampled frames at the original fps rate of the video. However, for videos with an fps rate equal to or higher than 10, we sampled frames at a fixed rate of 10 fps. After extracting the frames, we divided them into three subsets for training, validation, and testing. Specifically, 70% of the frames were allocated for training, 20% for validation, and the remaining 10% for testing purposes. To create our datasets, we divided each video into nine segments and selected two frames from each segment, resulting in a total of 18 frames per video.

Table 1. Specifications of the trained FireNetAGM model.

Parameter	Value
Batch size	**32**
Epoch	**100**
Loss Function	**BCEWithLogits**
Optimizer	**AdamW**
Learning Rate	**.001**
Weight Decay	**.0001**

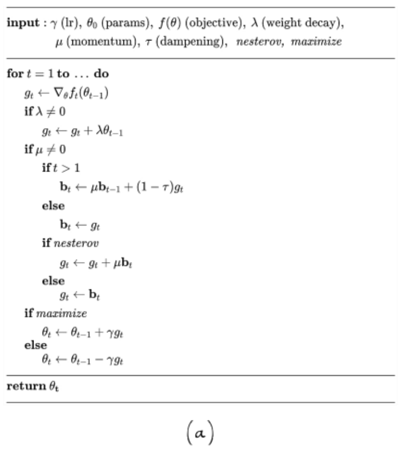
4 Results

The model undergoes training for 100 epochs, utilizing a low learning rate to ensure that the majority of previously learned information is preserved in the network. A detailed list of the various hyperparameters used can be found in Table 1.

To optimize the training process of the FireNetAGM model, we utilized the **AdamW** optimizer [7]. The AdamW optimizer combines the benefits of the Adam optimizer and weight decay regularization. It computes *adaptive learning rates* for each individual parameter by considering the first and second moments of the gradients, which helps accelerate convergence and improve training efficiency.

- The first moment of the gradient $\boldsymbol{m_t}$, also known as the **moving average of the gradients**, tracks the average direction in which the parameter weights are changing. It is computed as a weighted average of the gradients from the previous step, with the weight determined by the term $\boldsymbol{\beta_1}$. The β_1 value controls the influence of the previous gradients on the current update. Typically, β_1 is set to 0.9, meaning that the current update takes into account 90% of the previous gradients.
- The second moment of the gradient $\boldsymbol{v_t}$, also known as the **moving average of the squared gradients**, tracks the variation of gradients over time. It is computed as a weighted average of the squared gradients from the previous step, with the weight determined by the term $\boldsymbol{\beta_2}$. The β_2 value controls the influence of the squared gradients on the current update. Commonly, β_2 is set to 0.999, meaning that the current update considers 99.9% of the squared gradients.

Moreover, the weight decay component aids in mitigating the issue of overfitting by implementing regularization to the model's weights during the optimization process. The Fig. 5 shows the difference between the SGD and AdamW algorithms.

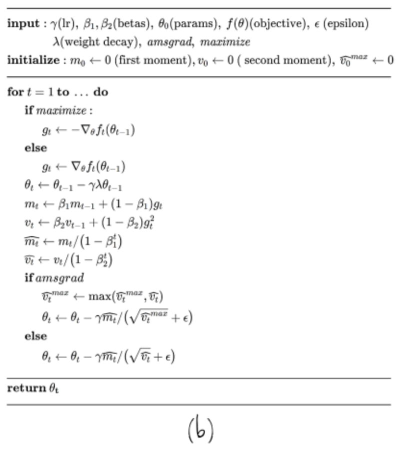


```
input : $\gamma$ (lr), $\theta_0$ (params), $f(\theta)$ (objective), $\lambda$ (weight decay),
        $\mu$ (momentum), $\tau$ (dampening), nesterov, maximize

for $t = 1$ to ... do
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    if $\lambda \neq 0$
        $g_t \leftarrow g_t + \lambda\theta_{t-1}$
    if $\mu \neq 0$
        if $t > 1$
            $\mathbf{b}_t \leftarrow \mu\mathbf{b}_{t-1} + (1-\tau)g_t$
        else
            $\mathbf{b}_t \leftarrow g_t$
        if nesterov
            $g_t \leftarrow g_t + \mu\mathbf{b}_t$
        else
            $g_t \leftarrow \mathbf{b}_t$
    if maximize
        $\theta_t \leftarrow \theta_{t-1} + \gamma g_t$
    else
        $\theta_t \leftarrow \theta_{t-1} - \gamma g_t$

return $\theta_t$
```

(a)



```
input : $\gamma$(lr), $\beta_1, \beta_2$(betas), $\theta_0$(params), $f(\theta)$(objective), $\epsilon$ (epsilon)
        $\lambda$(weight decay), amsgrad, maximize
initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ ( second moment), $\widehat{v_0}^{max} \leftarrow 0$

for $t = 1$ to ... do
    if maximize :
        $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
    else
        $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)g_t^2$
    $\widehat{m_t} \leftarrow m_t/(1-\beta_1^t)$
    $\widehat{v_t} \leftarrow v_t/(1-\beta_2^t)$
    if amsgrad
        $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
        $\theta_t \leftarrow \theta_t - \gamma\widehat{m_t}/(\sqrt{\widehat{v_t}^{max}} + \epsilon)$
    else
        $\theta_t \leftarrow \theta_t - \gamma\widehat{m_t}/(\sqrt{\widehat{v_t}} + \epsilon)$

return $\theta_t$
```

(b)

Fig. 5. (a) SGD and (b) AdamW algorithm.

In addition to the optimizer, we employed a learning rate scheduler to dynamically adjust the learning rate during training. The scheduler used in our model is the **ReduceLROnPlateau** scheduler. This scheduler monitors a specific metric, in our case, the validation loss, and reduces the learning rate when the monitored metric stops improving. This adaptive adjustment of the learning rate helps the model overcome local minima and improves its ability to converge to the optimal solution. The scheduler operates in the following manner: it monitors the validation loss after each training epoch and checks if the loss has stopped decreasing significantly. If the loss does not show substantial improvement for a certain number of epochs, the scheduler reduces the learning rate by a factor, often denoted as the "*factor*" parameter. This reduction in learning rate allows the model to make smaller updates to the parameters, potentially helping it find a better solution.

By employing the AdamW optimizer with appropriate hyperparameters and the ReduceLROnPlateau scheduler, we aim to achieve fairly efficient and effective training of the FireNetAGM model.

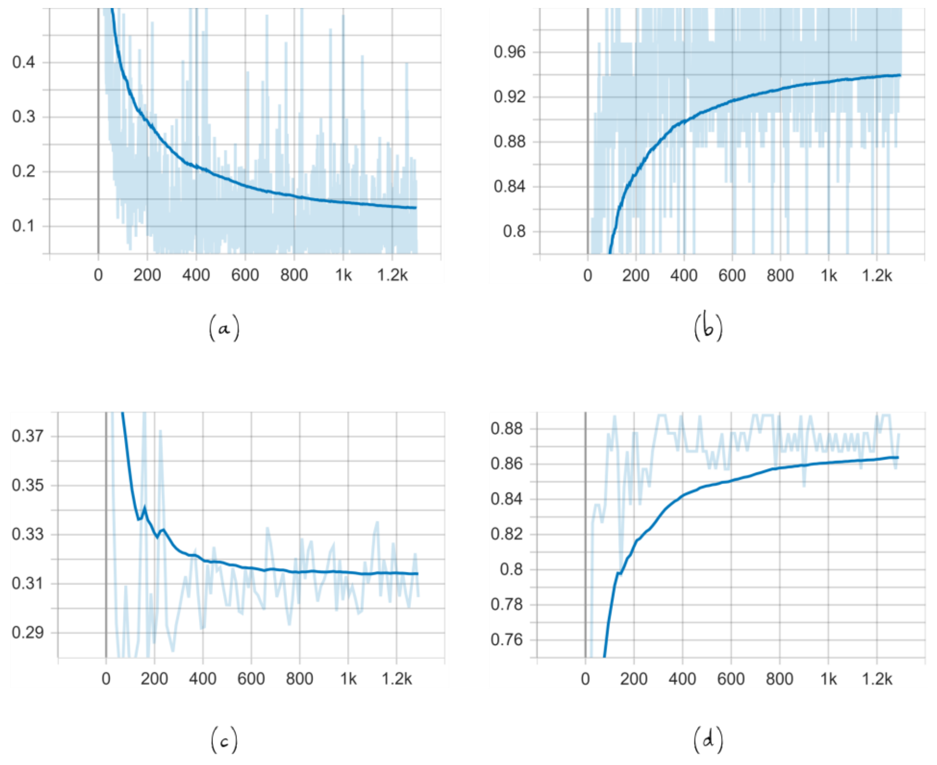


Fig. 6. FireNetAGM: (a) Training Loss and (b) Training Accuracy, (c) Validation Loss and (d) Validation Accuracy.

As depicted in **Fig. 6.** FireNetAGM: (a) Training Loss and (b) Training Accuracy, (c) Validation Loss and (d) Validation AccuracyFig. 6, we observe that the validation loss gradually decreases throughout the training process, eventually stabilizing at a value of 0.32. In contrast, the training loss continues to decrease and reaches a lower value of 0.14. The stable and relatively low validation loss, compared to the training loss, suggests the absence of overfitting in the model.

Considering the number of epochs, it is important to note that reaching a relatively stable loss value of 0.32 after a certain number of epochs indicates that the model has converged to a satisfactory performance level. This suggests that it might be possible to reduce the number of epochs from 100 to a smaller value without significantly compromising the model's performance. However, the decision to decrease the number of epochs should be carefully evaluated, considering the trade-off between training time and achieving optimal results.

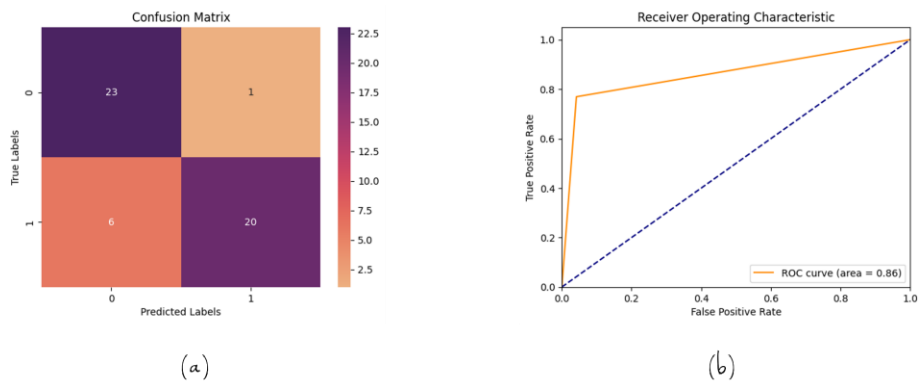


Fig. 7. Confusion Matrix and Roc Curve on Test Set

For the accuracy analysis of the model, we considered only the case where the model is able to predict whether there is fire or not given a sequence of frames, without considering the prediction of the exact frame where the fire starts. In particular, from the Fig. 7, we can derive the following evaluation metrics:

- Test Accuracy:

$$Accuracy = \frac{\#(samples\_correctly\_classified)}{\#(total\_samples)} = 0.86$$

- Test Precision:

$$Precision = \frac{TP}{TP + FP} = 0.95$$

- Test Recall:

$$Recall = \frac{TP}{TP + FN} = 0.75$$

- Test F1 Score:

$$F_1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 0.85$$

5 Improved Model

The model presented for evaluation is not the best model that we have obtained. Unfortunately, due to GPU exhaustion just a few hours before the deadline and a few epochs away from completing the training, we were unable to submit the best model. However, we were able to run the model the following day using the computing unit in Colab. The model obtained, partially on the first run and complete on the second run, achieved improved results compared to the submitted model.

What sets the **FireNetAGM2** model apart from the FireNetAGM model is the use of *data augmentation* to increase the size of the training set. This resulted in a total of 688 training sequences, compared to the 344 sequences in FireNetAGM. Furthermore, the FireNetAGM2 network differs in that it employs the **GRU** (Gated Recurrent Unit) as the recurrent network and utilizes 64 hidden layers instead of 128. Additionally, FireNetAGM2 uses the **ADAM** optimizer with weight decay of 0. Moreover, FireNet-AGM2 is trained for 30 epochs instead of 100.

The main reason for achieving improved performance in FireNetAGM2 compared to FireNetAGM is the use of **K-Cross-Validation**, with K set to 5. This approach helps in reducing the potential bias and variance issues associated with single training/validation splits, resulting in more reliable and accurate performance evaluation.

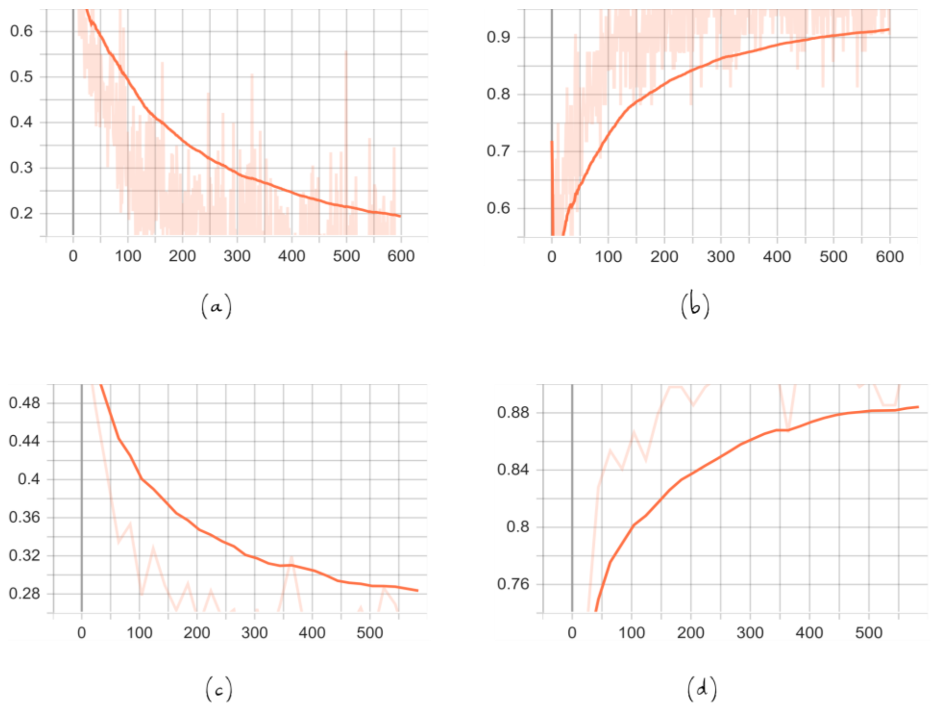


Fig. 8. FireNetAGM2: (a) Training Loss and (b) Training Accuracy, (c) Validation Loss and (d) Validation Accuracy.

References

1. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520.
2. Luo, C.; He, X.; Zhan, J.; Wang, L.; Gao, W.; Dai, J. Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices. arXiv 2020, arXiv:2005.05085.
3. Tragoudaras, A.; Stoikos, P.; Fanaras, K.; Tziouvaras, A.; Floros, G.; Dimitriou, G.; Kolomvatsos, K.; Stamoulis, G. Design Space Exploration of a Sparse MobileNetV2 Using High-Level Synthesis and Sparse Matrix Techniques on FPGAs. Sensors 2022, 22, 4318. https://doi.org/10.3390/s22124318.
4. Olah C.: Understanding LSTM Networks, Colah's blog, August 27, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
5. A. Jadon, A. Varshney, M.S. Ansari, Low-complexity high-performance deep learning model for real-time low-cost embedded fire detection systems, Proc. Comput. Sci. 171 (2020) 418–426.
6. M. Hervas, C. Fernandez: DCSASS Dataset | Kaggle.
7. Loshchilov, I., & Hutter, F. (2017). Fixing Weight Decay Regularization in Adam. ArXiv, abs/1711.05101.