Contents

Introduction

The aim of this project is to detect fire in videos acquired by video surveillance (static) cameras. The fire can occur at any moment in the video and the frame, in the presence of fire, must be selected.
The aim is to design a network for binary classification: the network must classify a sequence of frames (images) as either positive (fire) or negative (no fire).
Videos are acquired by different cameras at different resolutions/frame rate. May contain over-imposed text. Videos can be acquired during daylight/night, indoor/outdoor and by different distances. In addition, at a distance, the smoke only may be visible so positive frames can include detection of smoke, not only fire. Furthermore, video can contain fire-like (yellowish or reddish) or smoke-like (fog or clouds) objects.

Training set

The training dataset is composed of 330 videos:

- 246 fire;
- 84 no fire.

In addition to this dataset, another 54 videos are used as "Test set" divided in:

- 27 fire;
- 27 no fire;

Model Evaluation

Defined:

- g_i, the first frame in which the fire visible;
- p_i, the first frame in which the fire is detected;
- Δt, a guard time equals 5 seconds;
- TP, as True Positive all the detections in positive videos for which $p \geq max(0, g - \Delta t)$;
- FP, as False Positive all the detections occurring at any time in negative videos or in positive videos for which $P < max(0, g - \Delta t)$;
- FN, as False Negative the set of positive videos for which no fire detection occurs;

the final score is obtained by combining two different contributions:

1. detection performance metrics, described as precision (P), recall (R) and the delay (D) between the fire ignition and detection;
2. complexity, described as the processing capabilities in terms of frame rate and the GPU memory required.

Formulas used for the final score are:

$P = \frac{|TP|}{|TP|+|FP|}$ for the precision,

$R = \frac{|TP|}{|TP|+|FN|}$ for the recall,

$D = \frac{\Sigma_{i=1}^{|TP|} d_i}{|TP|}$, where $d_i = |p_i - g_i|$, for the delay.

In addition, we can define: $D_n = \frac{max(0,60-D)}{60}$, having sixty as maximum delay permissible.
The complexity contributors are defined as follow:

$$PFR = \frac{1}{\frac{\sum_{i=1}^{|N|} t_i}{|N|}},$$

for the processing frame rate, using it we can then define the PFR_{delta} as

$$PFR_{delta} = max\left(0; \frac{PFR_{target}}{PFR} - 1\right).$$

Memory occupancy is instead defined as:

$$MEM_{delta} = max\left(0; \frac{MEM}{MEM_{target}} - 1\right).$$

The final score is computed as follow:

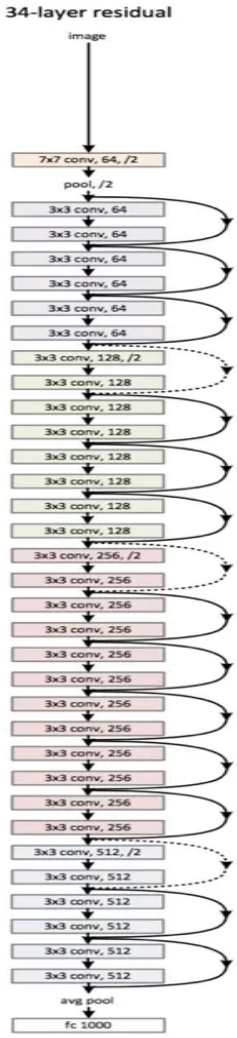
$$F = \frac{P \cdot R \cdot D_n}{\left(1+PFR_{delta}\right) \cdot \left(1+MEM_{delta}\right)}.$$

1. Analysed architectures

Listed below are all the Convolutional Neural Network (CNN) architectures analysed and the reasons why it was decided not to use them.

1.1 ResNet

The Residual Neural Network architecture, or 'ResNet', was the first CNN considered. One of the problems that ResNet networks solve is the famous 'Vanishing Gradient'. This is because, when the network is too deep, the gradients from which the loss function is calculated are easily reduced to zero after several applications of the chain-rule. With ResNets, the gradients can flow directly through the jump connections backwards from the successive layers to the initial filters.
The image in Figure 1 shows the general architecture of ResNets, where you can also see the jump between layers that allows the gradient not to reduce to 0 too quickly.



This network was discarded because, despite its 18-, 34- and 50-layer (and more in other two type of ResNet) variants, it is still a network that requires too much memory and computing power, which would have penalised the group in the competition, since some of the requirements for evaluating the network involve the calculation of the necessary memory, the speed of the network itself in giving an output, and the computing power required to generate said output.

Figure 1 - ResNet 34 architecture

1.2 Inception v3

The CNN 'Inception V3' network immediately seemed to be one of the best candidates for the binary problem of image analysis and fire recognition. The problem with this architecture is once again related to the computing power required. It is a network with 42 layers, as can be seen in Figure 2. Theoretically it would be one of the best networks to use, obviously modifying the output layer (a "Softmax" function is not suitable for a binary problem, a "Sigmoid" type function is typically used), with an error rate of 3.58. As anticipated, however, it is an overly expensive and above all slow network compared to other networks analysed.

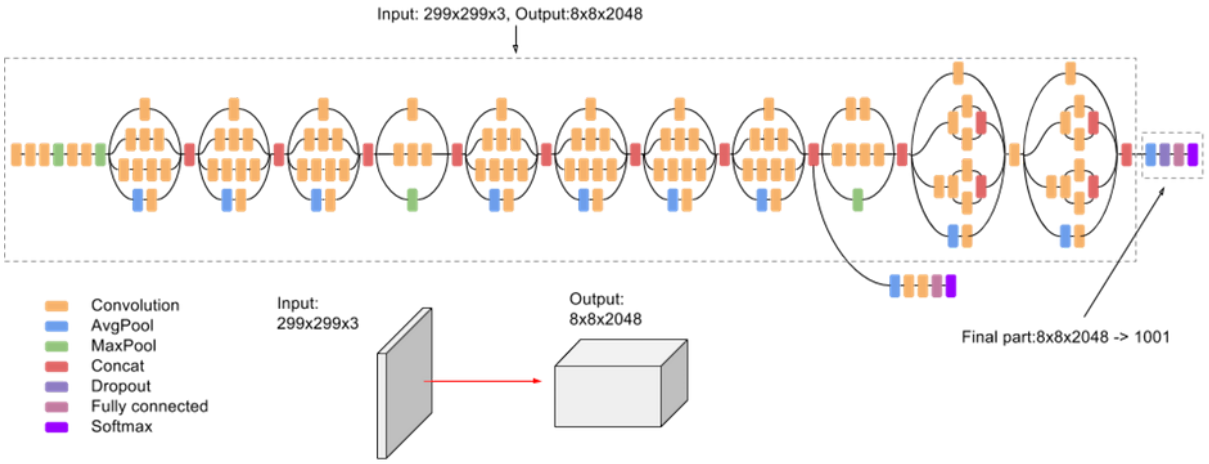


Figure 2 - Inception v3 architecture

1.3 MobileNet

Another type of architecture considered is the MobileNet. This architecture, which exists in three versions, the third of which was the CNN chosen for this project, requires less computing power and memory than its competitors. The three architectures will now be analysed.

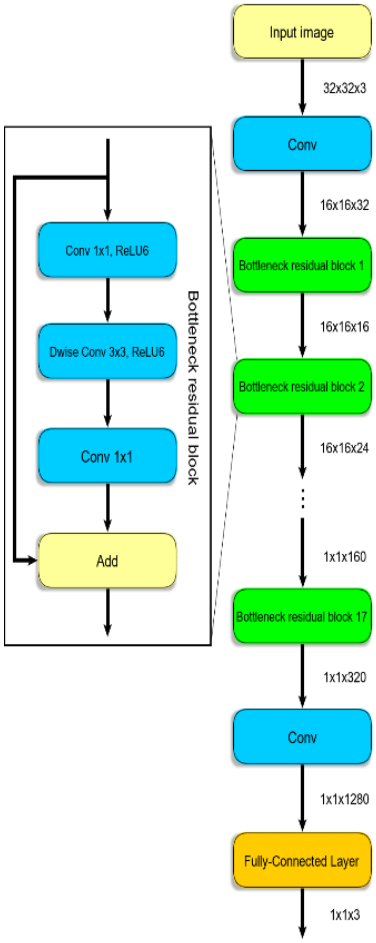
1.3.1 MobileNet v1

This network was not considered, being an outdated, and obsolete, version.

1.3.2 MobileNet v2

This network was the CNN initially chosen for the project: it requires little computing power, compared to the networks previously analysed, and is significantly faster (especially when compared to Inception v3). The architecture of this network is presented in Figure 3 (note that the architecture in the image was used for an image analysis problem like the problem we addressed, but with a ternary output instead of binary: it differentiates between fire, smoke and no-fire).
A prototype network was developed, following the work described in the paper "Fire Detection Using Deep Transfer Learning on Surveillance Videos" [1], written by Abdul Bari, Tapas Saini and Anoop Kumar, by removing the final layer and adding the list of layers in the aforementioned paper, modifying the final activation function, our network being a binary rather than ternary classification.
The aim of these modifications was to have an already trained network and to fine-tune the final layers with the datasets we had available, and additional datasets if deemed necessary.

The changes made to the network were as follows:



- removal of the final 'Classifier' layer;
- Addition of an 'AveragePooling2D' layer, with input and output values of 1280, 7x7 kernel, stride 1, padding 3 and ReLU activation function;
- addition of a 'Flatten' layer;
- addition of a Dense layer with 128 output values;
- addition of a Dropout layer;
- addition of the final layer, with 'Sigmoid' activation function for binary selection.

Figure 3 - MobileNet v2 general architecture

The next version of MobileNet, i.e. version 3, was therefore taken into consideration.
This architecture, being the version used in the project, will be described in the next chapter.

2. Used architecture

In this chapter the architecture of the MobileNetV3 will be explained. From the general, standard, architecture to the implementation for the fine-tuning.

2.1. MobileNet v3 architecture

As previously mentioned, this is the architecture used in the first trained network. Again, a pre-trained network was chosen, to which the last layer was modified, with the addition of additional layers. We then proceeded with a fine-tuning of these additional layers, before designing the next network, namely the Recurrent Neural Network, or RNN.
The architecture used is the MobileNet V3-Large, described in the next figure.

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Figure 4 - MobileNet v3 Large architecture

The fields in the table correspond to:

- **SE**: indicates whether the 'Squeeze-and-Excite' parameter is present in that block;
- **NL**: indicates the type of non-linearity used:
  - **HS**: 'h-swish' activation function;
  - **RE**: 'ReLu' activation function;
- **NBN**: indicates the absence of a 'Batch-Normalisation';
- **s**: indicates the stride used;
- **#out**: indicates the number of channels in output from the layer;

2.2 Layers description

The architecture is divided between 3 main layers: Convolutional 2D, Bottleneck layer and Pooling layer. A brief explanation of what these layers are composed and how they work, will be provided in this section.

2.2.1 Conv2d layer

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. A convolution converts all the pixels in its receptive field into a single value. For example, applying a convolution to an image, will decrease the image size as well as bringing all the information in the field together into a single pixel.
In the MobileNet v3 the first layer is a conv2d, as in all the other CNN's architecture, without a kernel. The fourth to last layer is a conv2d with a 3x3 kernel, and the last two layers are conv2d as well, both of which use a 1x1 kernel and the "No-Batch Normalization".
All these layers utilize the "*h-swish*" activation function, except for the last one, being the output layer.

2.2.2 Bottleneck layer

The bottleneck in a neural network is a layer with fewer neurons than the layer below or above it. Having such a layer encourages the network to compress feature representations (of salient features for the target variable) to best fit in the available space. Improvements to compression occur due to the goal of reducing the cost function, as for all weight updates.
In MobileNet v3 there are fifteen bottleneck layers, with a kernel size varying between 3x3 and 5x5. These layers indeed decrease the size of the input, passing from an input of $112^2 \cdot 16$ to a final output, of the last bottleneck layer, of $7^2 \cdot 960$.

In Figure 5 is reported the difference, based on a ResNet layer, between a normal layer and a bottleneck layer.

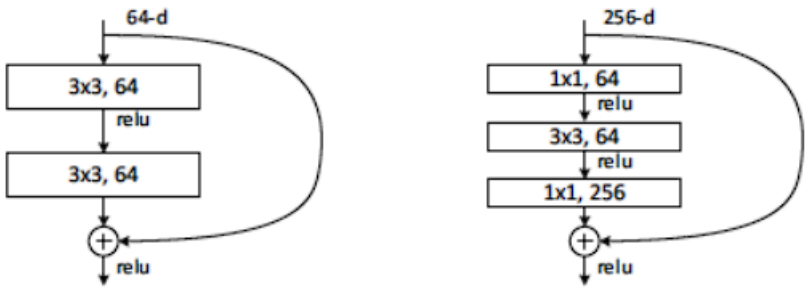


Figure 5 - Typical ResNet layer (left) vs Bottleneck ResNet layer (right)

First six bottleneck layers uses ReLU activation function, then all the other layers utilise the "h-swish" activation function.

2.2.3 Pooling layer

The last layer to be defined is the Pool, being the third-last of the network. MobileNet v3 utilises an Adaptive Average Pooling 2D layer, which applies a 2D adaptive average pooling over an input signal composed of several input planes, with a kernel of dimension 7x7. The difference between a normal average pool layer is that the adaptive one does not need to have specified kernel size, stride or padding. Instead, it only needs to know the output size requested, then it computes the necessary kernel, stride and padding size.

2.3 Squeeze and Excite block

The 'Squeeze and Excite' block is a block implemented inside the layer. It has been proved, with the ResNet architecture, that it improves the accuracy drastically: a ResNet50, with SE block, has almost the same accuracy of a ResNet101, maintaining, at the same time, a lower computational cost. In Figure 6 it is reported the ResNet's layer without SE (on the left) and with SE (on the right).

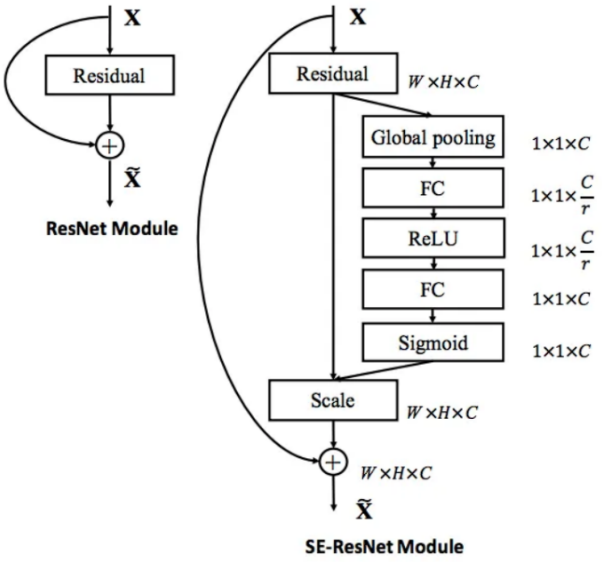


Figure 6 - Normal ResNet50 layer (left) vs ResNet50 layer with SE block implementation (right)

The SE block takes the feature map and number of channels as input. Then a Global Average Pooling is applied that converts each channel to a single numerical value (Squeezing part).
In the following a stack of two Dense blocks transform the 'n values' to 'n weights' for each channel (Excitation). Finally, the output is computed by applying weights to the channels by multiplication.
The first layer of the feed-forward network has a ReLU activation function, and the second (last) layer has a sigmoid activation function acting as a smooth gating function.

2.4 Non-linearity

The 'h-swish', or hard-swish, function is as follows:

$$hswish[x] = x \cdot \frac{ReLU \cdot 6 \cdot (x+3)}{6},$$

which is derived from the 'swish' function, defined as:

$$swish(x) = x \cdot \sigma(x),$$

a function that has been experimentally proven to improve accuracy. However, as the sigmoid function σ is computationally expensive and in this model there is a lot of computational overhead involved, the authors modified with the so-called 'hard-swish', or 'h-swish'. In the next figure, it is shown the difference between these functions.

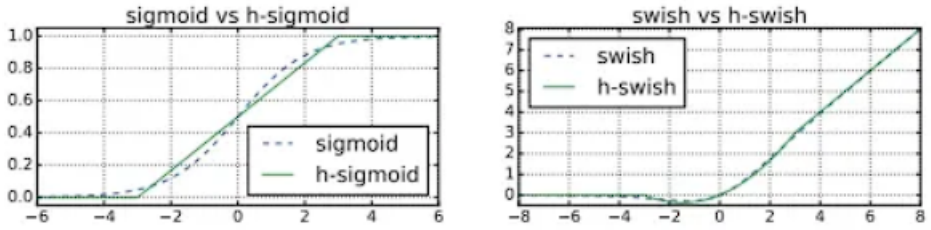


Figure 7 - Differences between Sigmoid, H-Sigmoid, Swish and H-Swish activation function

2.5 Architecture changes

The changes to the architecture were minimal: the last layer was modified to have an output size of one and activation function Sigmoid, since our problem is of binary classification. The model used is pretrained, so only the last layer needed to be trained.
Another change was done in the optimizer part. The SGD was confronted with the AdamW optimizer for computing the gradient descent, resulting in choosing the second optimizer, for better results.
Lastly the Binary Cross Entropy (BCE) function was used for computing the loss.
All these aspects will be discussed in the next chapter.

3. Training

3.1 Dataset

The dataset used for the training part is the “On_fire” dataset, provided by the MIVIA laboratory of Unisa.
As already described in the opening of this paper, it consists of 330 videos divided as:

- 246 fire;
- 84 no fire.

All the models are trained on this dataset and evaluated on the additional dataset of 49 videos, including the provided model.

3.2 Loss Function

The loss function utilised is the Binary Cross Entropy, or BCE for short. The formula is the following:

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log log\left(p\left(y_i\right)\right) + \left(1 - y_i\right) \cdot \log log\left(1 - p\left(y_i\right)\right),$$

where is defined:

- $p(y_i)$, as the probability of 1,
- $1 - p(y_i)$, as the probability of 0.

This is the standard loss function used for binary classification problems.

3.3 Optimizer

Instead of the classic Stochastic Gradient Descent optimizer, SGD for short, for calculating the values of the weights, a better version was used: AdamW.
AdamW optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments with an added method to decay weights per the techniques discussed in the paper, 'Decoupled Weight Decay Regularization' [2] by Loshchilov, Hutter et al., 2019.
According to Kingma et al., 2014, the underlying Adam method is "*computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters*".

3.4 Data processing

As previously mentioned, the dataset provided by Mivia was used for both training and validation procedures. To use it, the frames have first been extracted from all the videos, one frame per second. The VideoFrameDataset class was used as a dataset, through which a single frame per video was loaded. The training dataset was loaded by setting the test_mode parameter to False, thus taking a random frame from each video, while for the validation dataset the test_mode parameter was set to False, to always load the central frame of each video to compare the results obtained from each of the trained models.

For each image the following pre-processing has been applied to use the chosen pre-trained models:

1. resize of the images to 224 x 224 for the MobileNet v3 and a resize to 299 x 299 for the Inception;
2. a normalisation with mean = [0.485, 0.456, 0.406] and std deviation = [0.229, 0.224, 0.225] and max pixel value of 255, valid for both the MobileNet v3 and the Inception V3;
3. each image is transformed into a tensor by the VideoFrameDataset class.

As for augmentation a horizontal flip is performed with probability of 0.5.

3.5 Training procedure

For the training procedure, given the limited data available, the K cross validation technique was used. The folds were created starting from the training dataset, for a total of 6 folds and a model was trained on each fold for 200 epochs, with an early stopping patient of 40. The AdamW optimizer was used with 0.001 of learning rate and 0 of weight decay.
Once the training on each fold is completed the trained models were compared on the validation dataset to determine the correct value of accuracy and loss and the best of the six models was selected.

3.6 Models trained

A fine tuning was performed on two existing models, the MobileNet v3 large and the inception v3. Regarding the MobileNet, the output layer has been replaced with a fully connected layer with input size 1280 and output size 1. Three different fine tunings were performed on this network:

1. A fine tuning of the output layer only;
2. A fine tuning of the last two layer of the network's classifier;
3. A fine tuning of the entire network's classifier.

For Inception v3 instead the output layer has been replaced with a fully connected layer with size 2048 and output size of 1. On this network only the output layer was trained.

3.7 Test procedure

The trained models were tested on external videos collected in the Test set. For each video, the network analyses one frame per second. If a fire is detected for 3 consecutive frames the video is considered positive and on an external text file the second at which fire was detected is saved. Instead, if no fire is detected in the entire video, then the video is considered negative and the relative text file is left empty. The consecutive frame flag was added to avoid many false positives.

4. Results

In this chapter will be discussed the results of the provided model, and of the other model our team has trained as well, giving a reason to our final decision on the chosen model.
In total six models were trained and evaluated, all of them were pretrained, and only a fine tuning of the last layer was done.
Measures of precision and recall will be computed, for each model, following the formulas described in the introduction.
The "F-Score", of "F-Index", will be computed along the other two measures, for a better understanding of the model behaviour. The formula for the F-Score is:

$$F = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}.$$

4.1 Inception v3

As already stated in the first chapter, Inception v3 is an expensive computational model. Since our team trained different versions of the MobileNet v3, a different model was needed to provide a real confrontation between performance and overall results, hence this architecture was chosen as the opponent. The training process followed the same for all the models: a fine tuning was executed, training only the last layer of the model with the Mivia dataset, and then a test was done on the additional dataset. The results on this network, reported in the next table, are inferior to those of MobileNet v3.

Accuracy	FP on test set	FN on test set
0.84	15/27	0/27

Table 1 - Inception v3 results

The network works better on recognizing the absence of fire or smoke (as per the last value), but has a low accuracy compared to the next three models and much more False Positive detected on both training and test set. As stated previously, the reason for not choosing Inception v3 is for his computational costs. After the various tests done on the network, it was also clear that it needs a much bigger training and test dataset, for a proper training on fire detection: another reason for not using this network. Measure of precision and recall are:

$$P = 0.68,$$
$$R = 1.$$

The F-Score is for this model is:

$$\text{F-Score} = 0.8095.$$

4.2 Standard MobileNet v3

This model, the provided one, is the standard MobileNet v3, on which the output layer has been replaced with a fully connected one with input feature size of 1280 and output feature size of one. A fine tuning of this layer was then performed and, lastly, it was tested with the test set.
The results are reported in the next table.

Accuracy	FP on test set	FN on test set
0.96	8/27	1/27

Table 2 - Standard MobileNet v3 results

It is clear, comparing the two tables, that this network has better results than the previous one. With a greater accuracy and generally better performance on both training and test set, MobileNet v3 has outclassed Inception v3, with the dataset provided. In addition, this network has greater precision in identifying the instant at which the fire is generated, with less wrong values. Precision, recall and the F-Score computed are:

$$P = 0.77,$$
$$R = 0.96,$$
$$\text{F-Score} = 0.8545.$$

4.3 MobileNet v3 with 2 final layers trained

In this version of the MobileNet v3, the last two layers of the classifier were trained and tested. Overall results, reported in the next table, were slightly inferior to the previous one.

Accuracy	FP on test set	FN on test set
0.97	9/27	0/27

Table 3 - MobileNet v3 with 2 classifier layer trained results

Lower accuracy and slightly better performance on both sets, but this network has more difficulty in detecting the instant when the fire is generated in the video, with some values (in seconds) higher than 15, when all the videos in the test set had fire or smoke starting from 0. For this model, precision, recall and F-Score are as follow:

$$P = 0.75,$$
$$R = 1,$$
$$\text{F-Score} = 0.8571.$$

4.4 MobileNet v3 full classifier layer

Lastly, this network had the entire classifier layer trained and tested. Results are reported in the next table.

Accuracy	FP on test set	FN on test set
0.98	8/27	0/27

Table 4 - MobileNet v3 with entire classifier trained results

Average accuracy and slightly better results on training set, this model was nearly identical to the first one analysed. Precision, recall and F-Score for this model are:

$$P = 0.77,$$
$$R = 1,$$
$$\text{F-Score} = 0.87.$$

4.5 Other test on MobileNet v3

In addition to the networks reported above, other two tests were made with the MobileNet v3 architecture:

1. an architecture with two, final, fully connected layers;
2. an architecture with three, final, fully connected layers.

The first one simply added another linear layer to the classifier, with input feature of 512 and output feature of 1. Clearly, the output feature of the last but one linear layer was changed according to the input feature on the last one.
The second architecture added two linear layers to the last classifier level, scaling from 1280 to 512 to 64 to one, gradually, with ReLU activation function between the levels.
Unfortunately, the tests executed on these two architectures were a failure, since in more than half of the folds our team found an accuracy of 1 and a loss less than 0.08. Clearly an overfitting issue has affected these models, so no more tests were done.

4.6 Final considerations

In general the mobile nets have quite similar results. Although the accuracy and F-score values are slightly higher, the choice of the first mobile net was dictated by a lower delay in detecting the output, which is an important evaluation parameter for the task.

Tables and Bibliography

Figures

Table

Bibliography

[1] Fire Detection Using Deep Transfer Learning on Surveillance Videos", Abdul Bari, Tapas Saini and Anoop Kumar
[2] Decoupled Weight Decay Regularization, Loshchilov, Hutter et al., 2019.