
Università degli Studi di Salerno

.DIEM

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Corso di Laurea in Ingegneria Informatica

# Machine Learning 2022/2023

# Canale I-Z – Team 24

Project Work



| Cognome e Nome | Matricola | e-mail |
|---|---|---|
| Massaro Sara | 0622702015 | s.massaro3@studenti.unisa.it |
| Scala Felice | 0622702002 | f.scala14@studenti.unisa.it |
| Trotta Prisco | 0622702014 | p.trotta12@studenti.unisa.it |



Anno accademico 2022-2023

# Sommario

# Introduction

## Contextualizing the importance of fire detection

Real-time fire detection using image sequences is a highly sought-after capability in real-world video surveillance systems, as it plays a crucial role in preventing environmental disasters and ensuring continuous monitoring of urban areas and the preservation of forests. In this context, there is a significant demand for deploying "intelligent cameras" equipped with video analytics algorithms that can swiftly identify fires (including flames and/or smoke) in real-time. These cameras are strategically placed in remote locations and must operate independently with minimal computational resources, possibly accompanied by a compact embedded system responsible for processing the image sequence using the fire detection algorithm. Consequently, overly resource-intensive methods are impractical for this application. It is vital to strike a balance between fire detection accuracy, prompt notification generation, and efficient utilization of processing resources.

## Description of the objective of the artificial intelligence project

The objective of this project is to implement an artificial intelligence that will be executable on board of smart cameras or embedded systems, for real-time fire detection from videos acquired by fixed CCTV cameras. To this aim, the performance of the AI will be evaluated in terms of:

- fire detection capabilities and processing resources;
- detection errors and notification speed (i.e., the delay between the manually labelled fire start, either its ignition or appearance on scene, and the fire notification);
- average delay between fire start and notification (over all the true positive videos), the average processing frame rate and the memory usage;
- other metrics that will be explained more fully later.

# Problem Overview

## Discussing the risks and consequences associated with fires

Fires cause significant damage every year in terms of loss of human lives, property damage, destruction of natural habitats, and environmental impact. In Italy alone, in 2022, interventions by the National Fire Corps for fires and explosions exceeded 250,000 incidents. If fires are not detected promptly and get out of control, the consequences can be even more disastrous. Therefore, investing in Fire Detection resources is necessary: timely fire detection can protect human lives and limit environmental damage.

The first fire detection systems used ionization detectors, photoelectric sensors, and carbon dioxide detectors. However, this solution has many limitations, especially when fires occur in large, open areas.

Another approach involves using digital cameras and image processing algorithms to detect smoke and flames through video analysis.

The challenge in recent years has been to achieve ideal performance using computer vision, which has already demonstrated advantages over sensor-based systems.

## Exploring the challenges of fire detection in videos

Fire detection presents several challenges that need to be addressed to achieve acceptable results:

1. Variability of environmental conditions: Fires can occur in different environmental conditions, including variations in lighting, shadows, and other factors. This variability can affect the quality of captured images, making accurate fire detection more challenging.

2. Limited availability of training data: Often, there is a scarcity of available training data for fire detection algorithms, mainly due to privacy and security concerns. This limitation can affect the performance and generalization of the algorithms.
3. High processing speed: Ideally, fire detection algorithms should have high-speed processing capabilities to analyze images rapidly and reliably, enabling timely alerts to authorities if necessary.
4. Strong classification ability: The algorithms should possess a robust classification ability to differentiate between harmless smoke (e.g., from a cigarette) and smoke that could indicate a fire. The goal is to avoid false negatives and false positives, minimizing false alarms.

# Related Work

There are many proposed approaches that rely on modelling fire-coloured pixels. However, in real-world situations, many objects may be similarly coloured; therefore, if we use only the colour information, false detections could occur. For this reason, motion is also taken into consideration, and the background is removed to segment moving objects in a scene. The most relevant works that use this approach are:

1. P.-H. Huang, J.-Y. Su, Z.-M. Lu, and J.-S. Pan, ‘‘A fire-alarming method based on video processing”;
2. J. Chen, Y. He, and J. Wang, ‘‘Multi-feature fusion based fast video flame detection’’;
3. Z. Teng, J.-H. Kim, and D.-J. Kang, ‘‘Fire detection based on hidden Markov models’’.

In other works, attention is also focused on the flickering and random changes in fire illumination. The work by X. Qi and J. Ebert, titled ‘‘A computer vision-based method for fire detection in colour videos,’’ propose a 1-D wavelet transform for temporal colour variation analysis.

Other approaches involve deep learning algorithms based on convolutional neural networks (CNNs). The work by N. M. Dung and S. Ro titled “Algorithm for fire detection using a camera surveillance system” utilizes a CNN image classifier at the last layer of a cascade classification model to distinguish fire-like moving objects from real fires.

In the work 'Image fire detection algorithms based on convolutional neural networks' by P. Li and W. Zhao, the author attempts to detect fire using advanced object detection models such as YOLO, Faster-RCNN, R-FCN, and SSD. These models prove to be complex to train due to the amorphous shape of fire flames. Additionally, determining the optimal decision threshold is challenging.

Another idea is to use the CNN-LSTM combination to improve the accuracy of fire flame detection. In their work 'Real-time long short-term glance-based fire detection using CNN-LSTM neural network,' authors X. T. Pham, H. V. Nguyen, and C. N. The added CNN layers in the front end, followed by LSTM layers with a dense layer at the output. The CNN layer functions as a feature extractor, while LSTM acts as a video classifier for fire and non-fire classification.

M. D. Nguyen etc. suggest in ‘Multistage Real-Time Fire Detection Using Convolutional Neural Networks and Long Short-Term Memory Networks’ to apply the color and flickering analysis to determinate candidate fire regions inside a video and classify them using a CNN-LSTM network.

# Methodology

## Explanation of the artificial intelligence approach used

Given the requirement to classify videos in the shortest time possible, especially on resource-constrained systems like video-surveillance cameras, we explored two different approaches using lightweight models.

In the first approach, we evaluated well-known lightweight CNNs for image classification, including ResNet, MobileNet and EfficientNet. We conducted experiments to assess their performance in detecting flames and smoke from frames captured by various CCTV footages. To efficiently classify entire videos, we devised a criterion that involved analyzing

one frame per second and identifying sequences of frames containing flames. We measured the performance of the implemented and trained models using metrics defined by the contest on some videos using the criterion previously proposed.

Inspired by prior works that leveraged CNN+LSTM models, we explored an alternative approach to classify consecutive sequences of frames rather than individual frames. To ensure the model's efficiency in terms of memory and time complexity, we chose lightweight CNNs previously evaluated, followed by a simple Long-Short Term Memory (LSTM) unit. This LSTM layer would serve as a classifier for fire and non-fire sequences of frames. We also implemented a criterion to classify entire videos by examining predictions from sequences of frames using a sliding window approach. We evaluated the proposed models based on the metrics defined in the project guidelines.

These two approaches allowed us to explore the trade-offs between model complexity and performance in the context of real-time video classification for fire detection. By considering lightweight models and incorporating LSTM units, we aimed to strike a balance between accuracy and computational efficiency, ensuring that the system could be feasibly deployed on embedded systems like video-surveillance cameras.

## Description of the dataset used for model training and testing

The dataset provided to us consists of 322 videos, but a fire (flames and/or smoke) is present in 219 (positive) videos, while the others (negative) do not contain a fire. The videos have been collected from existing datasets (e.g. MIVIA, RISE, KMU, Wildfire, D-Fire), by selecting as positive videos only those that really frame a fire and not flames and smoke in controlled conditions, and as negative videos the ones that contain moving objects that can be confused with flames or smoke. For each positive video, the fire ignition time has been annotated, namely the time instant in which the fire begins. Since the videos are collected in different conditions, the dataset is very heterogeneous in terms of image resolution, illumination, distance from flame or smoke, pixel size of flame or smoke, background activity, scenario (urban or wildfire).

In addition to the dataset provided to us, we collected additional videos to make a dedicated test set for evaluating the various models implemented. Videos were collected from datasets found within scientific articles for artificial networks developed for fire detection: from these datasets, all videos belonging to the training set provided to us were filtered out. Our test set consists of 26 videos, but a fire (flames and/or smoke) is present in 18 (positive) videos, while the others (negative) do not contain a fire. For each positive video, we manually annotated the fire ignition time.

## Transformations applied to videos during training in different models

For each proposed network, during the training phase, we use the Python library Albumentations for image processing, which allows us to implement augmentation. The term "augmentation" refers to the process of transforming images with the aim of increasing data and making them more variable during training to improve the model's generalization.

The transformations used are:

1. HorizontalFlip: This is a data augmentation technique that horizontally flips both the rows and columns of an image matrix. As a result, you will obtain an image flipped horizontally along the y-axis.
2. GaussNoise: This applies Gaussian noise to the input image. This can help make the image more robust to lighting variations.
3. Blur: This allows applying blur to the image. This can be useful for simulating the effect of smoke or heat that may be present around the fire.
4. CLAHE: This reduces the intensity of the image. This can create a rainy atmosphere that reduces visibility.

# Model Architecture

## Description of the Convolutional Neural Network architectures utilized

### MobileNet

MobileNet is a family of convolutional neural networks (CNNs) that are designed to be lightweight and efficient. They are specifically designed for mobile and embedded devices, where computational resources are limited.

MobileNets achieve their efficiency by using several techniques, including:

- Depthwise separable convolutions: These convolutions allow MobileNets to reduce the number of parameters without sacrificing accuracy.
- Squeeze-and-excitation blocks: These blocks help to improve the performance of MobileNets by learning to focus on the most important features in an image.
- Quantization: This technique reduces the precision of the weights in a MobileNet, which further reduces the size of the model.

MobileNets have been shown to be very effective for a variety of mobile vision tasks, including image classification, object detection, and semantic segmentation. They are a popular choice for developers who need to build mobile apps that can perform these tasks with limited computational resources.

Benefits of using MobileNet:

- They are lightweight and efficient, making them ideal for mobile and embedded devices.
- They are accurate, achieving state-of-the-art results on a variety of vision tasks.
- They are easy to use, with pre-trained models available for a variety of tasks.

Limitations of using MobileNet:

- They may not be as accurate as larger CNNs, such as ResNet or VGG.
- They may not be as versatile as larger CNNs, as they are typically designed for specific tasks.

#### *MobileNetV3*

MobileNetV3 is the third generation of the MobileNet family of CNNs. It was introduced in 2019 by Google AI, and it builds on the previous versions of MobileNet by introducing several new features. Key features of MobileNetV3 are:

- Inverted residual blocks: These blocks are a new type of convolution block that is more efficient than the blocks used in previous versions of MobileNet. An Inverted Residual Block, sometimes called an MBConv Block, is a type of residual block used for image models that uses an inverted structure for efficiency reasons. It was originally proposed for the MobileNetV2 CNN architecture. It has since been reused for several mobile-optimized CNNs.

  A traditional Residual Block has a wide -> narrow -> wide structure with the number of channels. The input has a high number of channels, which are compressed with a 1x1 convolution. The number of channels is then increased again with a 1x1 convolution so input and output can be added.

  In contrast, an Inverted Residual Block follows a narrow -> wide -> narrow approach, hence the inversion. We first widen with a 1x1 convolution, then use a 3x3 depthwise convolution (which greatly reduces the number of parameters), then we use a 1x1 convolution to reduce the number of channels so input and output can be added.
- Linear bottlenecks: These bottlenecks are used to reduce the number of parameters in the model without sacrificing accuracy.
- Depthwise separable convolutions: These convolutions are used to further reduce the number of parameters in the model.
- Squeeze-and-excitation blocks: These blocks are used to improve the performance of the model by learning to focus on the most important features in an image. The Squeeze-and-Excitation Block is an architectural unit

designed to improve the representational power of a network by enabling it to perform dynamic channel-wise feature recalibration. The process is:

- The block has a convolutional block as an input.
- Each channel is "squeezed" into a single numeric value using average pooling.
- A dense layer followed by a ReLU adds non-linearity and output channel complexity is reduced by a ratio.
- Another dense layer followed by a sigmoid gives each channel a smooth gating function.
- Finally, we weight each feature map of the convolutional block based on the side network; the "excitation".

MobileNetV3 has been shown to be significantly more efficient than previous versions of MobileNet, while still achieving state-of-the-art accuracy on a variety of vision tasks. Benefit of using MobileNetV3 is that it is more efficient than previous versions of MobileNet.

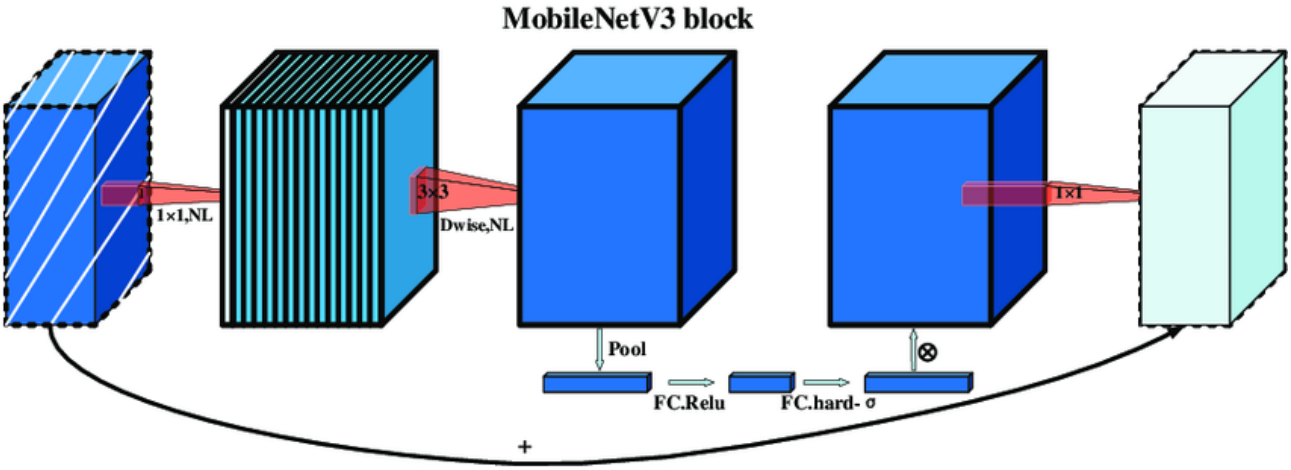



*Figure 1: MobileNetV3 block*

## ResNet

A ResNet (short for Residual Network) is a deep convolutional neural network (CNN) architecture that was introduced by researchers from Microsoft Research in 2015. It has become a popular and influential model in the field of computer vision and has achieved state-of-the-art performance on various visual recognition tasks, such as image classification and object detection.

ResNet was designed to address the problem of vanishing gradients that can occur when training very deep neural networks. As the number of layers in a network increases, it becomes challenging for the network to learn effectively due to the degradation problem. The degradation problem refers to the observation that adding more layers to a neural network can result in higher training error instead of lower error.

To overcome this issue, ResNet introduces the concept of residual learning. Instead of directly learning the desired underlying mapping, ResNet learns residual functions with respect to the input. This is achieved by using skip connections, also known as identity mappings or shortcut connections, that allow the network to skip one or more layers. These skip connections enable the network to directly propagate information from earlier layers to later layers, bypassing the need for each layer to explicitly learn the desired mapping.

The core building block of ResNet is the residual block, which consists of a set of convolutional layers with skip connections. Typically, each residual block contains two or three convolutional layers with batch normalization and ReLU activations. The skip connection directly adds the input of the block to its output, allowing the network to learn the residual mapping.

ResNet architectures come in different depths, such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, among others. The number indicates the total number of layers in the network. Deeper ResNet architectures have shown improved performance at the cost of increased computational complexity.

Overall, ResNet's skip connections and residual learning enable the training of extremely deep neural networks, allowing for better feature extraction and higher accuracy on various visual recognition tasks.

*ResNet18 VS ResNet50*

ResNet-18 and ResNet-50 are two of the most popular residual neural network (ResNet) architectures. They are both based on the ResNet V2 architecture, which uses identity connections to address the vanishing gradient problem. However, there are some key differences between the two models.

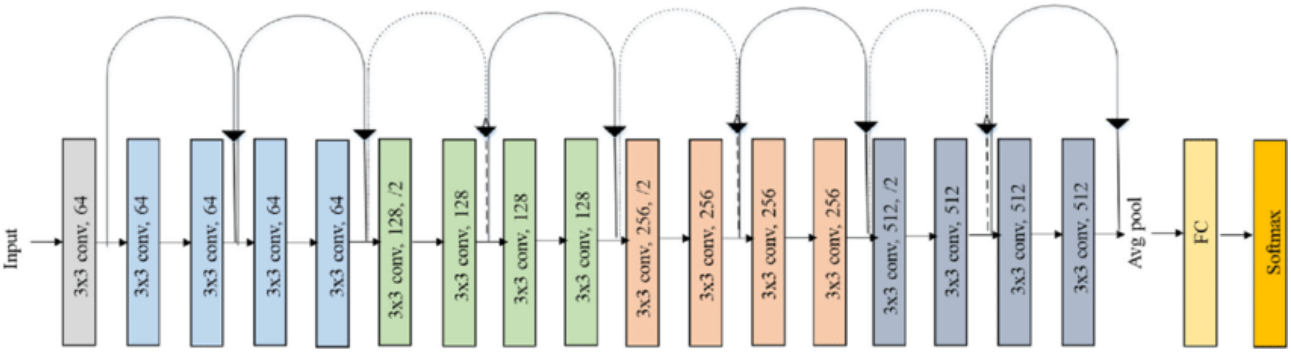



*Figure 2: ResNet18 Architecture*

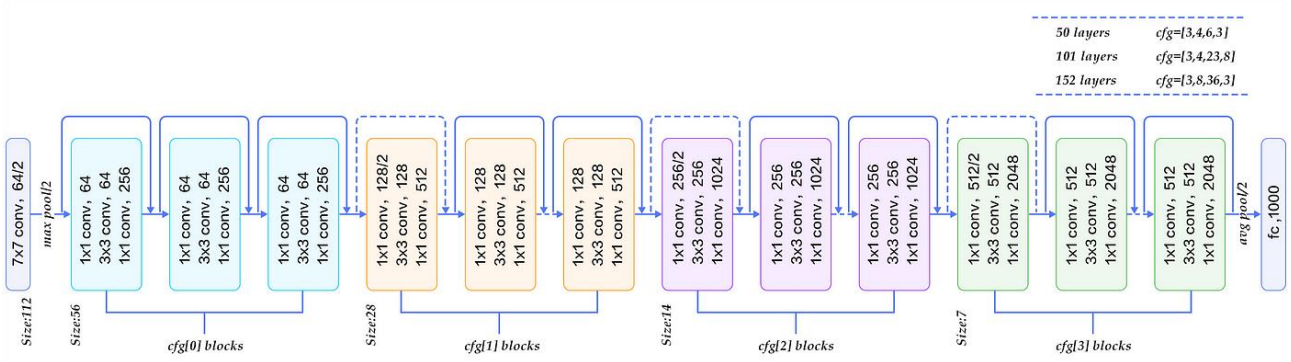



*Figure 3: ResNet50 Architecture*

- Number of layers: ResNet-18 has 18 layers, while ResNet-50 has 50 layers. This means that ResNet-50 is deeper than ResNet-18, which can lead to better performance on some tasks. However, it also means that ResNet-50 is more computationally expensive to train.
- Number of filters: the number of filters in each layer also differs between the two models. ResNet-18 has 64 filters in the first layer, while ResNet-50 has 256 filters. This means that ResNet-50 has a wider receptive field than ResNet-18, which can also lead to better performance on some tasks.

- Performance: ResNet-50 has been shown to outperform ResNet-18 on a number of image classification tasks. For example, on the ImageNet dataset, ResNet-50 achieves a top-5 error rate of 23.9%, while ResNet-18 achieves a top-5 error rate of 30.2%.
- Speed: ResNet-18 is faster to train than ResNet-50. This is because ResNet-50 has more layers and filters, which requires more computation.

The best model to choose depends on your specific needs. If you need the best possible performance, then ResNet-50 is a good choice. However, if you are looking for a faster model, then ResNet-18 is a better option.

Here is a table that summarizes the key differences between ResNet-18 and ResNet-50:

| Feature | ResNet-18 | ResNet-50 |
|---|---|---|
| Number of layers | 18 | 50 |
| Number of filters | 64-512 | 256-2048 |
| Performance | Good | Excellent |
| Speed | Fast | Slow |
| Best for | Small datasets, speed | Large datasets, accuracy |

*Table 1: Comparison between ResNet18 and ResNet50*

## EfficientNet

EfficientNet is a family of convolutional neural networks (CNNs) that are designed to be both accurate and efficient. They were introduced in 2019 by researchers at Google AI, and they have quickly become one of the most popular CNN architectures for a variety of tasks, including image classification, object detection, and semantic segmentation.

EfficientNets achieve their accuracy and efficiency by using a combination of techniques, including:

- Compound scaling: This technique allows EfficientNets to scale the width, depth, and resolution of a CNN in a principled way, which leads to better accuracy without sacrificing efficiency.
- Squeeze-and-excitation blocks: These blocks help to improve the performance of EfficientNets by learning to focus on the most important features in an image.
- Mobile inverted residual bottlenecks: These bottlenecks are a new type of convolution block that is more efficient than the blocks used in previous CNN architectures.

EfficientNets have been shown to be significantly more accurate and efficient than previous CNN architectures, such as ResNet and MobileNet. They have achieved state-of-the-art results on a variety of vision tasks, including image classification on the ImageNet dataset, object detection on the COCO dataset, and semantic segmentation on the Cityscapes dataset.

Benefits of using EfficientNet are:

- They are accurate, achieving state-of-the-art results on a variety of vision tasks.
- They are efficient, making them suitable for a variety of devices, including mobile phones and embedded devices.
- They are easy to use, with pre-trained models available for a variety of tasks.

Limitations of using EfficientNet are:

- They may not be as versatile as larger CNNs, as they are typically designed for specific tasks.
- They may not be as efficient as some other CNN architectures, such as MobileNetV3.

Overall, EfficientNet is a powerful tool for vision-based applications. It is accurate, efficient, and easy to use, making it a good choice for a variety of tasks.

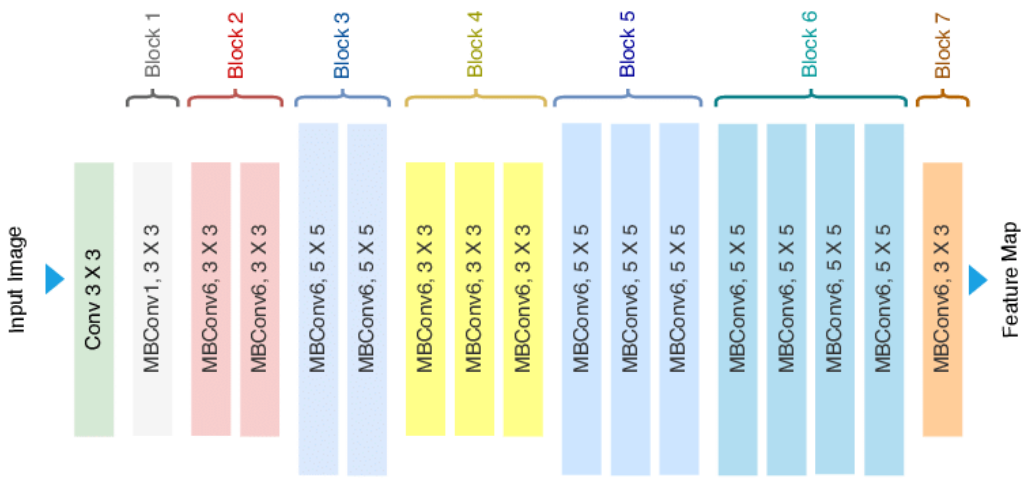



*Figure 4: EffiicientNet architecture*

## Comparisons between the proposed CNN architectures

In the table, we compare architectures we proposed: ResNet18, ResNet50, MobileNetV3 and EfficientNet-B1. The comparison includes the number of parameters (in millions), the number of GFLOPS (giga floating-point operations per second), and the Top-1 and Top-5 accuracies achieved in the ImageNet challenge. From the table, we can observe that MobileNetV3 stands out as an excellent choice due to its good performance, lower number of parameters, and reduced number of operations required. This is achieved through the utilization of depthwise separable convolutions, inverted residual blocks, and squeeze-and-excitation blocks, which optimize the model's efficiency without compromising accuracy.

While ResNet18 and EfficientNet-B1 also provide respectable results, MobileNetV3 emerges as a compelling option for tasks that demand a balance between accuracy and computational efficiency, making it well-suited for real-time applications or resource-constrained environments.

| Model | Top-1 Accuracy | Top-5 Accuracy | #Params | GFLOPS |
|---|---|---|---|---|
| ResNet50 | 75.3% | 93.29% | 25M | 3.8 |
| ResNet18 | 72.33% | 91.80% | 11.7M | 1.82 |
| EfficientNet-B1 | 78.8% | 94.4% | 7.8M | 0.7 |
| MobileNetV3 | 74.04% | 91.34% | 5.48M | 0.23 |

*Tabella 2: Comparisons between CNNs*

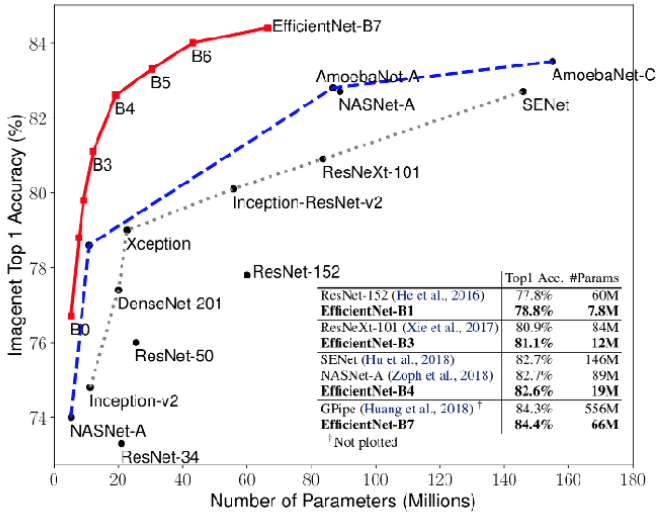



*Figure 5: Comparison of pre-trained models on ImageNet data set [17]*

## Usage of the proposed CNN architectures

To analyze the presence of fire and/or smoke in a single frame, we conducted several experiments where we adapted the classifiers of ResNet18, ResNet50, and MobileNetV3 to suit our binary classification task. As the ImageNet dataset contains 1000 classes, we replaced the 1000-class classifier with a binary fully connected classifier, tailored for our two classes: positive (frames with fire) and negative. In these experiments, we explored different configurations for the binary classifiers, focusing on the number of layers, number of neurons per layer (with a single neuron for the final level), and activation functions. The goal was to find the best configuration that would accurately distinguish frames with fire or smoke from those without.

- For ResNet18 and ResNet50, we substituted the original fully connected layer with a new MLP, testing various setups with one, two, or three layers. We utilized the Sigmoid activation function for these layers.
- Regarding MobileNetV3, we performed experiments where we replaced the last layer of the two-layer MLP with a new layer while keeping the first one with the Hardswish activation function and a Dropout layer. Additionally, we also tried a three-layer MLP with the Hardswish activation function and Dropout layers.

Throughout these experiments, we retained the unchanged extraction part of the network, as we believed that the pre-implemented filters were adept at adapting and extracting the most relevant features from the frames during the training process.

## Description of CNN+LSTM architecture

This architecture has been firstly proposed in 'Long-term Recurrent Convolutional Networks for Visual Recognition and Description' by Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell in 2016.

This model combines a deep hierarchical visual feature extractor (such as a CNN) with a model that can learn to recognize and synthesize temporal dynamics. In our case the model works by passing each visual input (frames of a selected sequence from a video) through a CNN to produce a fixed-length vector representation. The outputs of CNN are then passed into a recurrent sequence learning module, such as a LSTM network. In its most general form, a recurrent model has parameters $W$, and maps an input $x_t$ and a previous time step hidden state $h_{t-1}$ to an output $z_t$ and updated hidden state $h_t$. Therefore, inference must be run sequentially, by computing in order: $h_1 = f_W(x_1, h_0) = f_W(x_1, 0)$, then $h_2 = f_W(x_2, h_1)$, etc., up to $h_T$. To predict a distribution $P(y_t)$ over outcomes $y_t \in C$ (where $C$ is a discrete, finite set of outcomes) at time step $t$, the outputs $z_t \in R^{d_z}$ of the sequential model are passed through a linear

prediction layer $\widehat{y_t} = W_z z_t + b_z$, where $Wz \in R^{|C| \times d_z}$ and $b_z \in R^{|C|}$ are learned parameters. Finally, the predicted distribution is computed by evaluating the activation function on $\widehat{y_t}$.

In the paper, the proposed model was utilized for action recognition in videos using *CaffeNet* as the CNN base followed by an LSTM unit. The model was trained on the UCF101 dataset, where video clips of 16 frames were used for training. During inference, for each video, 16 frames were extracted with a stride of 8 The authors compared their model to a single-frame baseline model, where frames were individually classified by a CNN. The results of several experiments demonstrated that the new network outperformed the single-frame models. This improvement in performance was attributed to the fact that the proposed model not only determined the objects present in the frames, as done by the baseline model, but also considered the motion in the video clips.

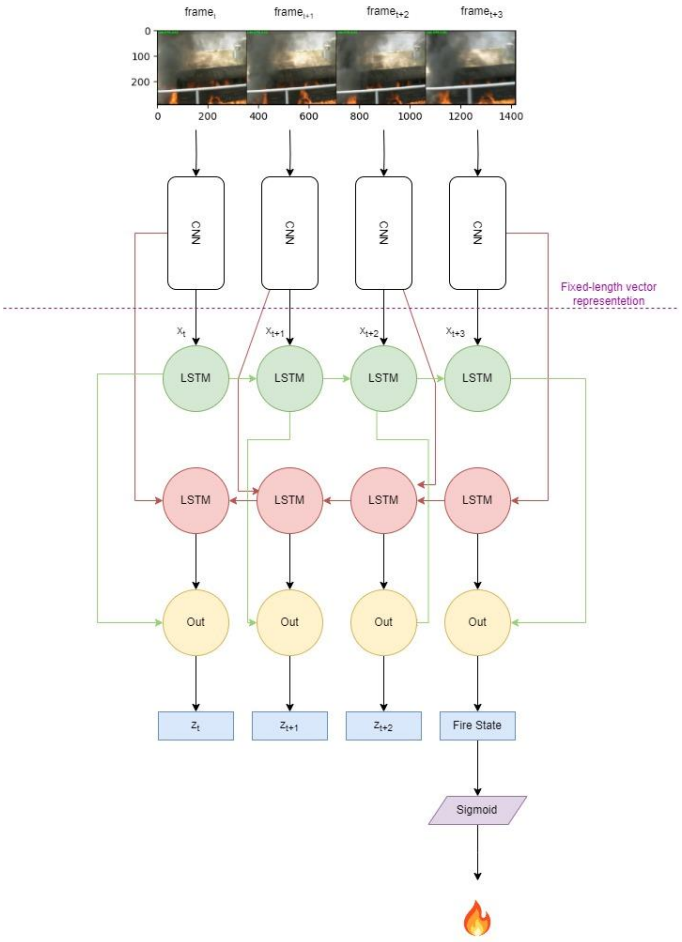



*Figure 6: Network Structure of CNN+LSTM*

By making some adjustments and changes in the organization, the proposed model can be adapted to handle various tasks beyond Action Recognition, including Image Captioning and Video Description.

In our case, we have evaluated performances of this architecture using some of the lightweight CNNs we briefly discussed, including MobileNetV3 and ResNet18, according to the performances we obtained on the single frame models. To obtain the features for each frame, we utilized the CNN base and replaced the classifier part with an Identity layer, which allowed us to collect the feature vectors. The LSTM unit has been configured to contain two bidirectional layers with hidden size of 100. The output $z_t$ is fed to a linear prediction layer, that is a MLP layer with a single output. At the end, the output of the entire layer is computed by evaluating the Sigmoid function on the excitement layer of the last neuron.

# Model Training and Optimization

## CNN Single-Frame models

### Description of the model training procedure for a CNN model

For each video, we extracted one frame per second and saved them in individual folders corresponding to each video. To load the videos during the training process, we utilized the VideoFrameDataset, which we obtained as supplementary material for this project. The VideoFrameDataset class simplifies and optimizes the loading of video samples from video datasets in PyTorch.

The class offers ease of use since it can be employed with custom datasets with minimal effort and without requiring modifications. It also ensures efficiency by implementing a fast video loading pipeline. This reduces GPU waiting time during training by eliminating input bottlenecks that can significantly slow down the training process. Furthermore, the class incorporates an effective sampling strategy for video frames. Training with the complete sequence of video frames, which can be several hundred frames long, is computationally demanding. To address this, the class employs sparse temporal sampling, evenly selecting frames from the video. This approach allows for representation of every part of the video while accommodating videos of varying lengths within the same dataset.

To apply the K-Cross Validation approach, we divided the dataset into multiple folds. Specifically, we split the training dataset into 6 folds, maintaining an 80/20 proportion for the training subset and the validation dataset. At each iteration k, we designated the kth fold as the validation dataset while using the remaining folds as the training set.

For videos selected for the training phase, we decided that the VideoFrameDataset should uniformly sample a single frame from each video. This choice ensures that the network encounters different frames at each epoch, promoting variation in the training process. For the video designated for the validation phase, we decided that the VideoFrameDataset should consistently select the same frame at each epoch. This allows us to calculate metrics such as accuracy and F-score using the same frames, facilitating comparison of results across epochs within the same iteration.

We configured the training dataset to apply the necessary preprocessing transformations for the selected CNN network in our experiment. These transformations were applied dynamically when loading frames from memory, eliminating the need to preprocess the entire set of images upfront and save them in memory, which would occupy more space. Additionally, we incorporated further transformations for data augmentation in the training dataset, as explained in the previous paragraph. These augmentations were applied probabilistically based on the chosen probabilities. Since these transformations were applied on the fly, we avoided the need to store multiple versions of the same frame with different combinations of transformations, thereby conserving memory. For the validation dataset, we only applied the preprocessing transformations to measure the performance of our network at each epoch consistently, without providing it with different frames in each iteration.

After setting up the video frame dataset and dividing the data into folds, we proceeded with training using the Stochastic Gradient Descent (SGD) optimization algorithm. SGD is a popular and widely used optimization algorithm in machine learning, particularly for deep learning models. SGD works by randomly selecting a subset of samples, known as a mini-batch, from the training set at each epoch. These mini-batches are used to compute the gradient of the loss function with respect to the model parameters. The gradients are then used to update the model weights in the direction that minimizes the loss. Considering the size of each frame and the available memory in the GPUs we utilized, we configured the SGD algorithm to select 50 videos from the training folds for each mini-batch. This option can be adjusted based on the available GPU VRAM and capabilities.

During the training process, we applied SGD to iteratively update the parameters of our model. For each iteration, a mini-batch of video frames was fed into the model, and the loss was computed based on the predictions compared to the ground truth labels. The gradients were then calculated and used to update the model weights.

Fine-tuning is a process in deep learning where a pre-trained neural network is further trained on a new or different dataset to adapt it to a specific task. The idea behind fine-tuning is to utilize the knowledge gained from training the network on a large dataset (often referred to as the pre-training phase) and use it as a starting point for training on a smaller, task-specific dataset. During the fine-tuning process, the modified network is trained using the new dataset, which is specific to the target task. This new dataset is typically smaller than the original pre-training dataset, and the network adjusts its weights based on this new information. The benefit of fine-tuning, widely applied in computer vision, natural language processing, and other deep learning applications, is that it enables the network to leverage knowledge from a related task or domain and quickly adapt to new tasks with smaller datasets. This reduces overfitting problems that might arise from having a limited dataset.

The feature extraction part of each network we selected had previously been trained on the ImageNet dataset. However, since the ImageNet dataset contains images from 1000 classes other than fire, we opted to perform fine-tuning on these networks. This involved initializing the networks with pre-trained weights from the ImageNet dataset and modifying part or the entire classification component to suit our binary classification task. This approach allowed us to start the training process from a non-random initial point, leveraging the knowledge learned from the ImageNet dataset. We made the decision to allow the weights of the convolutional part of the network to be updated during training. This allowed the filters to adapt and focus on capturing the specific characteristics of flames and smoke. By enabling the convolutional weights to change, the network could learn to extract relevant features related to our target classes. To ensure that the network retained its prior knowledge from the ImageNet training, we employed a low learning rate and a high momentum. A low learning rate helped to prevent drastic updates to the network's weights, allowing it to retain the learned features to some extent. Additionally, a high momentum value contributed to the stability of the optimization process, preventing the network from completely forgetting the information acquired during the previous training on the ImageNet dataset. By performing fine-tuning, adapting the network architecture, and carefully selecting the learning rate and momentum values, we aimed to leverage the pre-trained features while allowing the network to specialize in the detection of fire and smoke in our specific binary classification problem.

We limited the training to only 100 epochs due to time constraints we had for the training process. Considering the computational resources and time available imposed by Google Colab, we needed to strike a balance between achieving satisfactory results and completing the training within the given timeframe. By starting from non-random pre-trained weights and leveraging transfer learning, we could benefit from the knowledge already encoded in the network's initial parameters. This initialization allowed us to make efficient use of the available training time without sacrificing performance. Given that the network was already initialized with significant knowledge from the previous training on ImageNet, it was not necessary to train for a large number of epochs to achieve good results. Training for a limited number of epochs allowed us to further consolidate and specialize the network in the specific features of our binary classification problem related to fire and smoke. This decision saved training time while maintaining high performance expectations due to the non-random weight initialization.

Since we were dealing with a binary classification problem, we chose the Binary Cross-Entropy as Loss Function. This is a common choice for classification problems and allowed us to estimate the discrepancy between the model's predictions and the ground truth labels, optimizing the model's ability to accurately classify fire and non-fire examples.

During the training process, our focus was on estimating whether each individual frame contained fire or not. Rather than classifying the entire video as a single entity, our objective was to obtain predictions for each frame and compare them with the corresponding ground truth labels. By training the model to recognize flames and smoke in single images, we aimed to leverage this capability to identify the precise moment of fire emergence in a video. We believed that if the model could successfully detect fire and smoke in a single frame, it would enable us to determine at which second the fire first appeared in the video. This approach would have allowed us to classify the entire video based on the predictions made for each frame, providing a more detailed understanding of the fire occurrence and progression throughout the footage.

## Discussion of evaluation metrics used to measure model performance

During each epoch of the training phase, we computed the loss function as well as the sets of true positives, false positives, and false negatives. This information was used to derive several evaluation metrics including accuracy, precision, recall, and F-score.

Accuracy represents the overall correctness of the model's predictions, measuring the proportion of correctly classified frames. Precision measures the proportion of true positive predictions among all positive predictions, reflecting the model's ability to minimize false positives. Recall, also known as sensitivity, quantifies the proportion of true positive predictions among all actual positive instances, indicating the model's capability to minimize false negatives. F-score is a combined metric that balances precision and recall, providing a single value that represents the model's overall performance in terms of precision and recall. In particular:

$$P = \frac{|TP|}{|TP| + |FP|}$$

$$R = \frac{|TP|}{|TP| + |FN|}$$

$$F - Score \ = \frac{2PR}{P + R}$$

At the end of each epoch, we extended the evaluation of these parameters to the validation dataset specifically chosen for that particular iteration. This allowed us to assess the model's performance on unseen data and gain insights into its generalization capabilities. By evaluating these parameters on the validation dataset, we could determine how well the model was able to generalize and make accurate predictions on new, unseen instances. This information was crucial in making decisions regarding model selection, hyperparameter tuning, and assessing the model's overall effectiveness in real-world scenarios.

## Test Results

After completing the model training phase, the testing/validation phase is carried out to verify the ability of a CNN network to correctly classify frames extracted from a video. The dataset used for this phase is the validation dataset associated with the analysed fold. During the testing phase, the network analyses images that are different from those seen during the training phase. The objectives of this phase are:

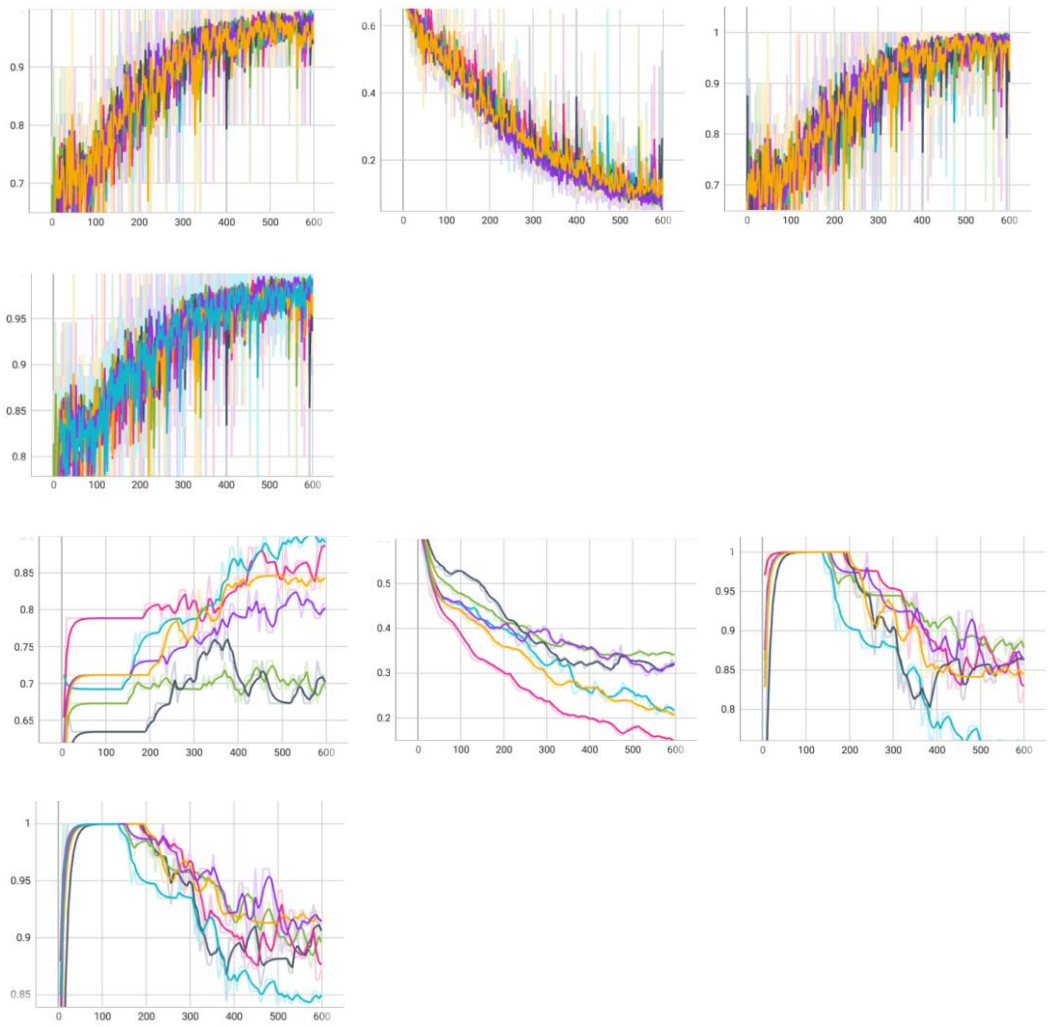
1. Evaluate performance: Assess how the network responds to new data and measure its ability to generalize.
2. Detect anomalous behavior: Identify any clearly incorrect predictions or unexpected behavior from the network.
3. Compare different models: The results obtained in this phase will be used to compare different architectures.
4. Verify reliability: Calculate metrics such as precision, recall, accuracy, loss, and Confusion Matrices to assess the reliability of the network's predictions.

For each experiment, the obtained graphs on TensorBoard are shown in the following order: train/acc, train/loss, train/precision, train/F-Score, val/acc, val/loss, val/precision and val/F-Score.

Considering the ResNet50 architecture, for the different experiments, we obtain:

- 07092023-1:

  For the experiment of 07/09/2023-1, the recall is high and close to 1, both during the training phase and the validation phase. It is possible to notice that as the number of steps increases, the performance improves until it stabilizes around a certain value.



The model incorrectly classifies frames as positive if there is a slight haze. Additionally, it confuses red/orange-colored houses (visible from a distance) as flames of a fire.

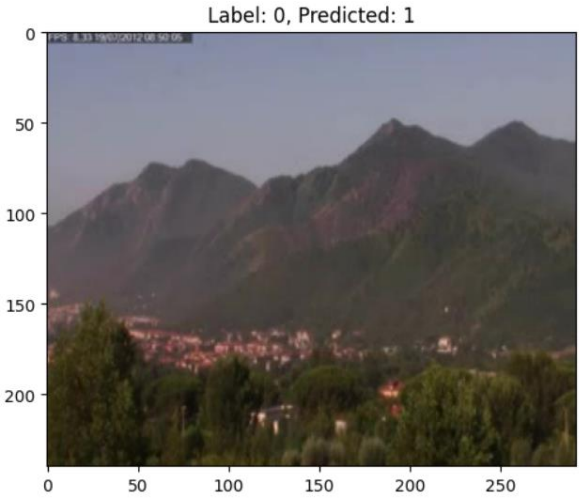
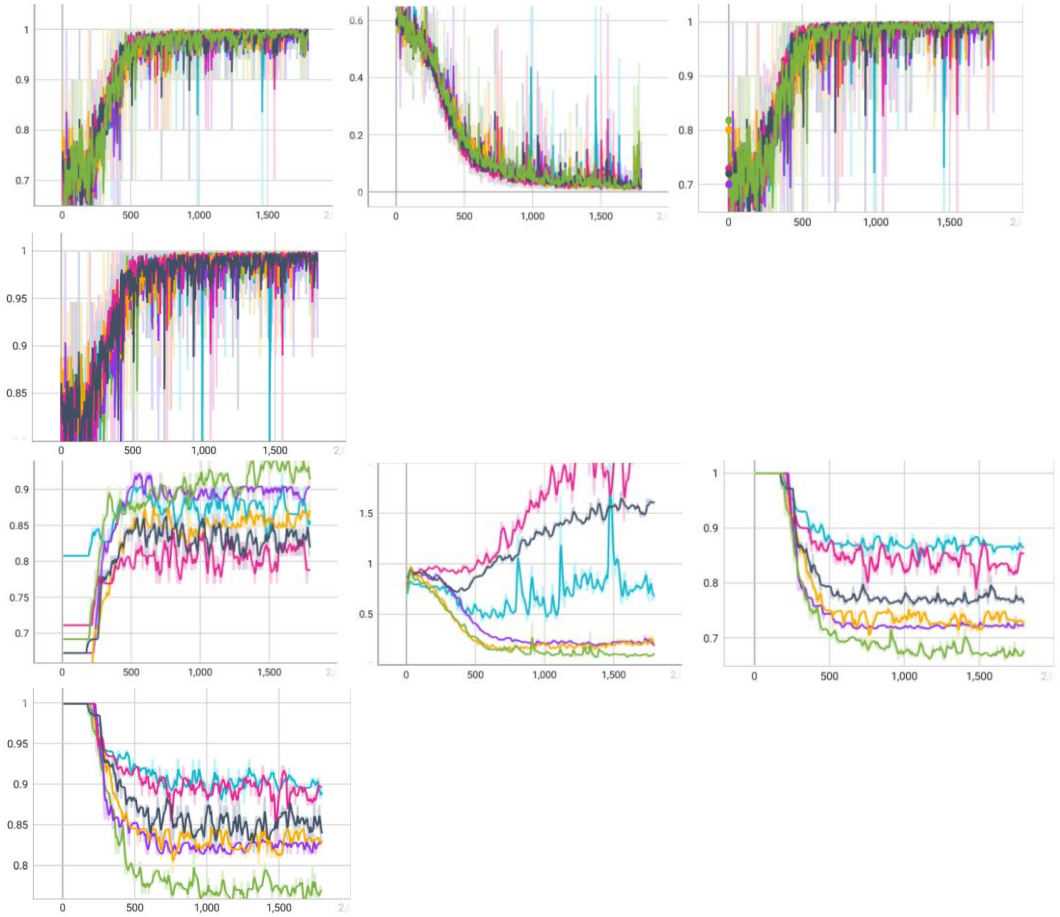



*Figure 1 - Misclassified frame ResNet50 07092023-1*

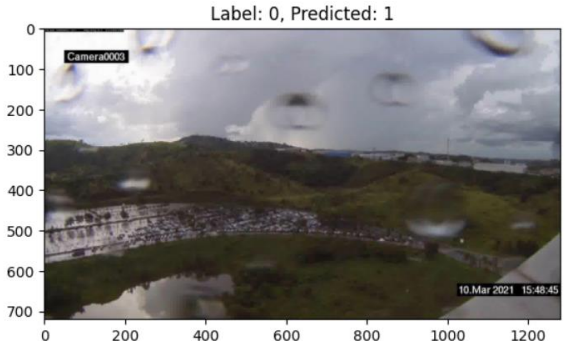
Considering the ResNet18 architecture, for the different experiments, we obtain:

- 06262023-2:

For this experiment, the recall is close to 1 in training phase and the validation phase. With high recall, there are no false negatives.



The model makes mistakes in classifying frames if they contain rain. In fact, they are classified as positive even when they are not.



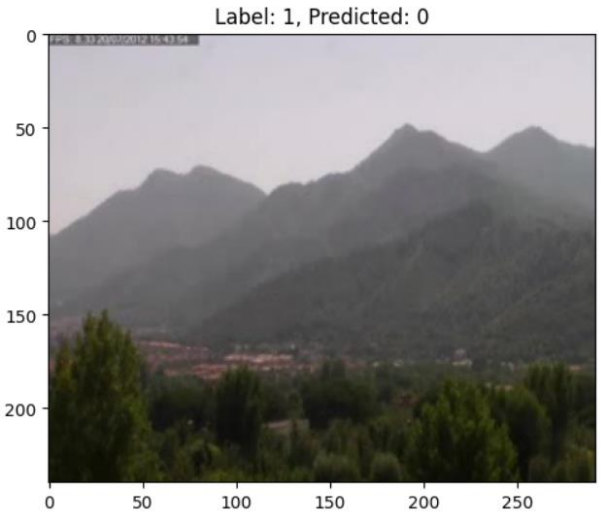



*Figure 2 - Misclassified frame ResNet18 06262023-2*

A particular case is the following: the network classifies a frame as negative where fire is present and where there are houses in red and orange colors.

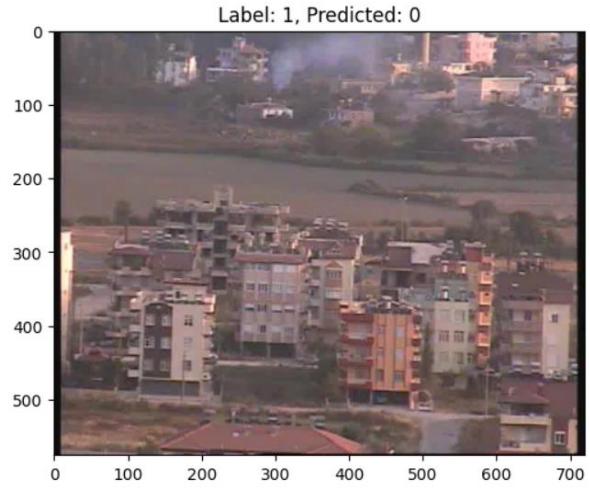
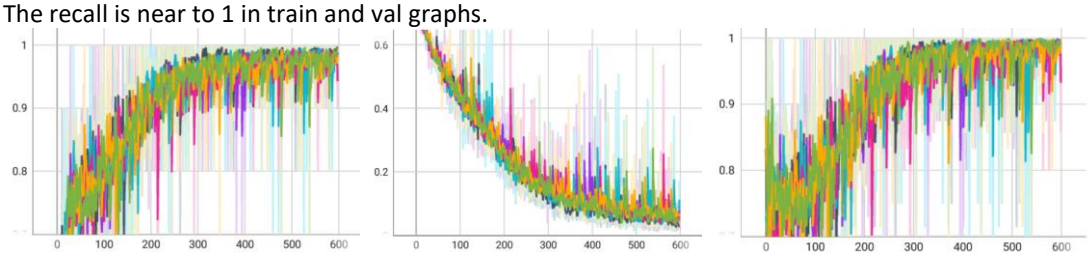
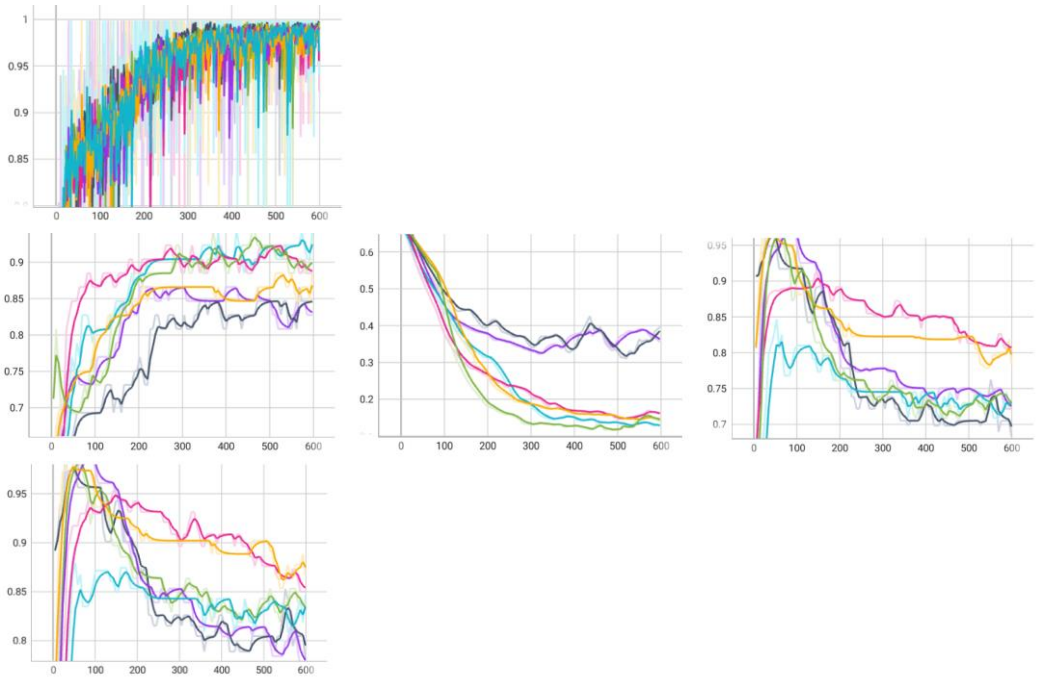



*Figure 3 - Misclassified frame ResNet18 06262023-2*

Considering the MobileNetV3 architectures, for the different experiments, we obtain:

- 07032023-1:
  The recall is near to 1 in train and val graphs.

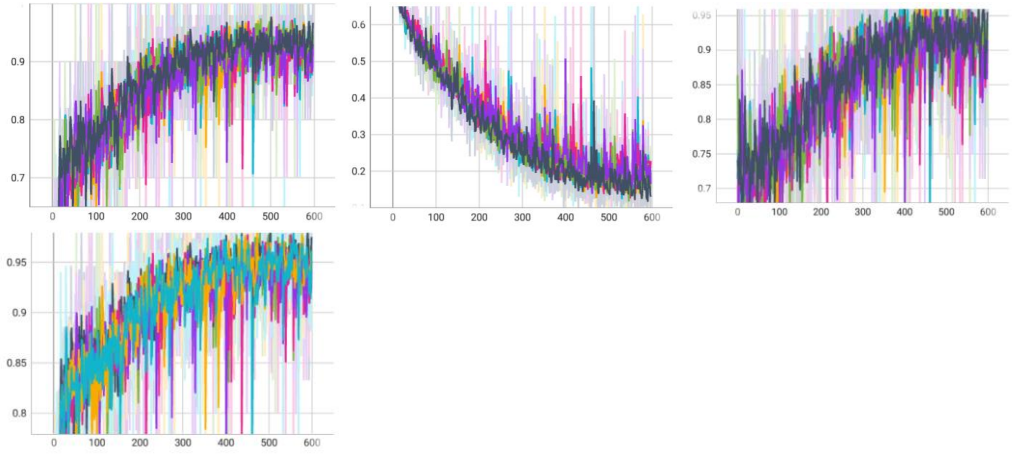


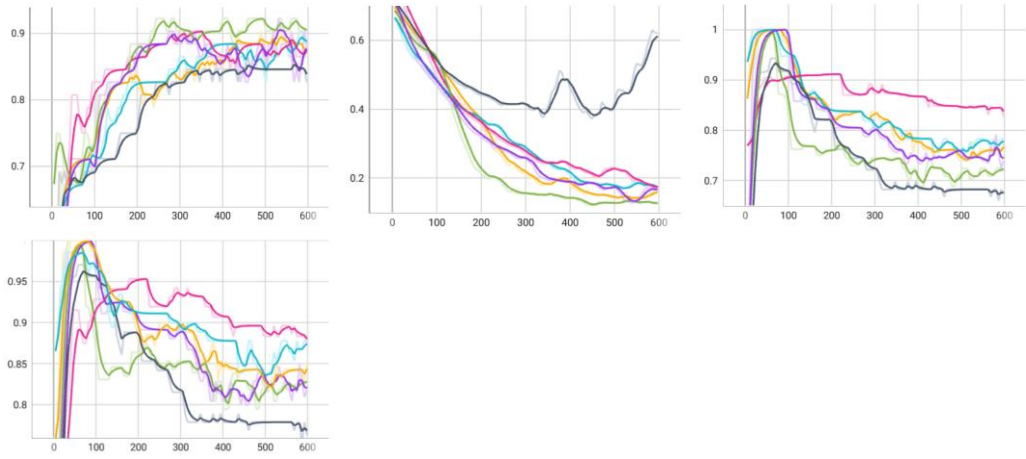


The model classifies a frame as negative if it is raining or if there are distant houses (or object) in red or orange colors, incorrectly classifying them as positive.

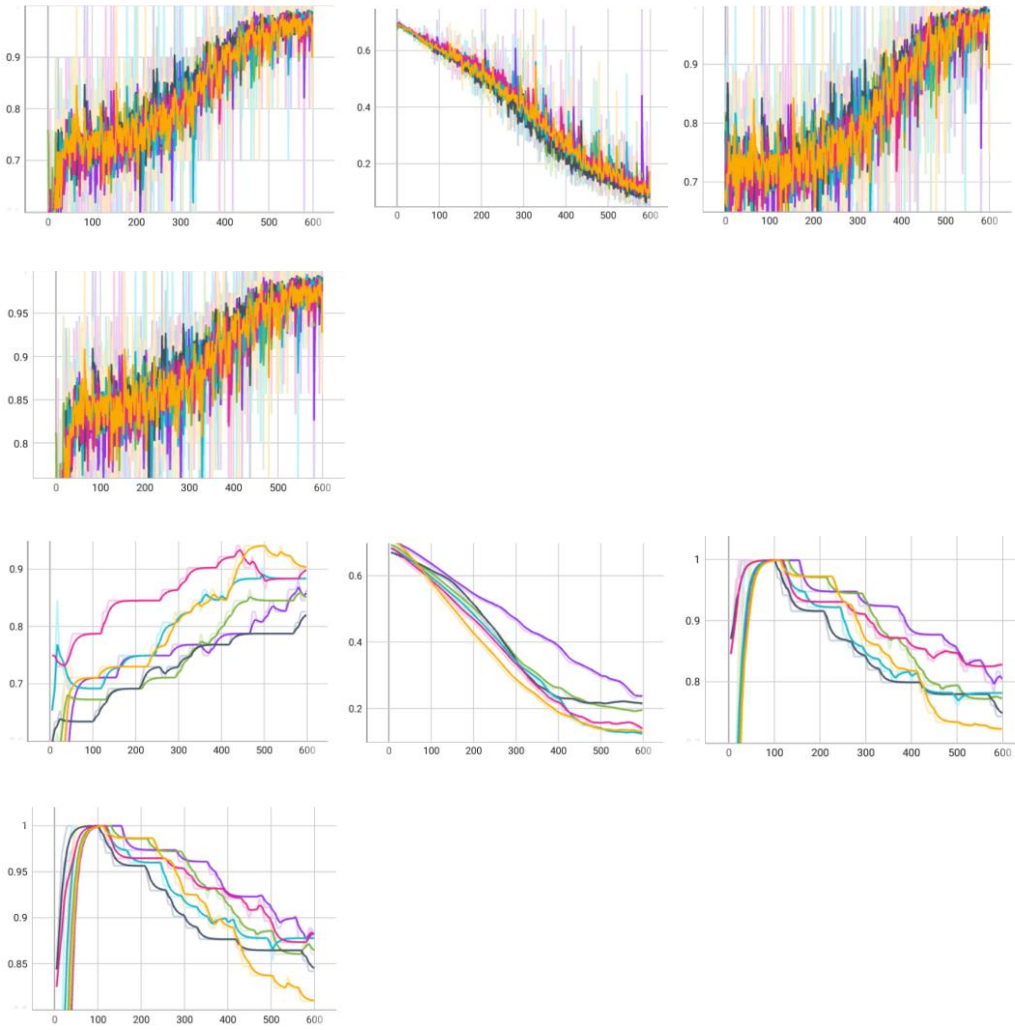

*Figure 4 - Misclassified frame MobileNetV3 07032023-1*

- 07052023-1 and 07082023-2: Both models make mistakes in classifying a frame that contains distant red or orange houses, incorrectly classifying them as positive even though they are negative.
  For these two experiments, the recall is approximately 1.
  For 07052023-1, the TensorBoard graphs are as follows:





For 07082023-2, the TensorBoard graphs are as follows:



## CNN+LSTM models

### Description of the model training procedure for a CNN+LSTM model

To capture movement within video sequences, we extracted two frames per second from each video. This allowed the CNN+LSTM models to capture more detailed information about the dynamic changes occurring within the fed sequence. During the training phase, we utilized the *VideoFrameDataset* to efficiently load the frames. Since our objective was to classify sequences of frames instead of individual frames, we configured the Dataset to uniformly select four consecutive frames for each video, corresponding to a two-second duration. This hyperparameter choice took into account memory constraints and the need to obtain predictions quickly in real-world scenarios. Reducing prediction delay is crucial, particularly in safety-critical environments.

Throughout our experiments, we maintained the use of k-Cross Validation and Stochastic Gradient Descent (SGD), as previously discussed. We continued fine-tuning our networks to optimize their performance. In some experiments, we opted to use the CNN base already pretrained for fire classification. These pre-trained networks are adept at extracting relevant features from images to determine the presence of flames. By leveraging this approach, the LSTM and classifier units could quickly determine the best parameters to minimize the loss function.

We chose to use the Binary-Cross Entropy as the Loss function since we still treated the task as a binary classification problem, focusing on determining the presence or absence of fire in the video sequences.

During the training process, our primary focus was on obtaining the classification of sequences of frames, as opposed to single frames. We believed that this approach would help the network better understand the presence of flames and smoke, as it could observe the movement of objects and dynamic changes within these sequences. Unlike single-frame models, our model may not be able to precisely determine the exact frame in which ignition or the appearance of fire occurs, as frames are analysed together rather than individually.

However, we anticipated that this limitation would not pose a significant issue during the real operating time of our model. During inference, we could analyse overlapping sequences of frames and detect where the fire first appears. By examining multiple overlapping sequences, we can ascertain the approximate moment of ignition or the fire's initial appearance. This approach provides a reliable way to determine the occurrence of fire, even though the exact frame may not be precisely identified.

### Discussion of evaluation metrics used to measure model performance

As already done for single-frame models, during each epoch of the training phase, we computed the loss function as well as the sets of true positives, false positives, and false negatives, as well as accuracy, precision, recall, and F-score.

$$A = \frac{|TP| + |TN|}{|TP| + |FP| + |TN| + |FN|}$$

$$P = \frac{|TP|}{|TP| + |FP|}$$

$$R = \frac{|TP|}{|TP| + |FN|}$$

$$F - Score = \frac{2PR}{P + R}$$

During this phase of our approach, our primary goal is not to precisely recognize the first frame of fire ignition or appearance. Instead, we classify a sequence of frames as a true positive if the predicted class aligns with the positive class indicated in the ground truth. In this evaluation process, we do not specifically consider the frame number reported in the label. The reason for this approach is that we uniformly select a sequence of four frames to analyse, and the fire

may have already started within these sequences. In such cases, looking for the exact frame of ignition becomes less relevant, as it typically corresponds to the first frame in most instances. Instead, our focus is on the classification of the entire sequence accurately, determining if the sequence indeed contains fire (positive class) as indicated by the ground truth. By considering the entire sequence of frames, we gain a holistic understanding of the presence of fire, rather than emphasizing the exact frame of ignition.

At the end of each epoch, we extended the evaluation of these parameters to the validation dataset specifically chosen for that particular iteration.

## Test Results

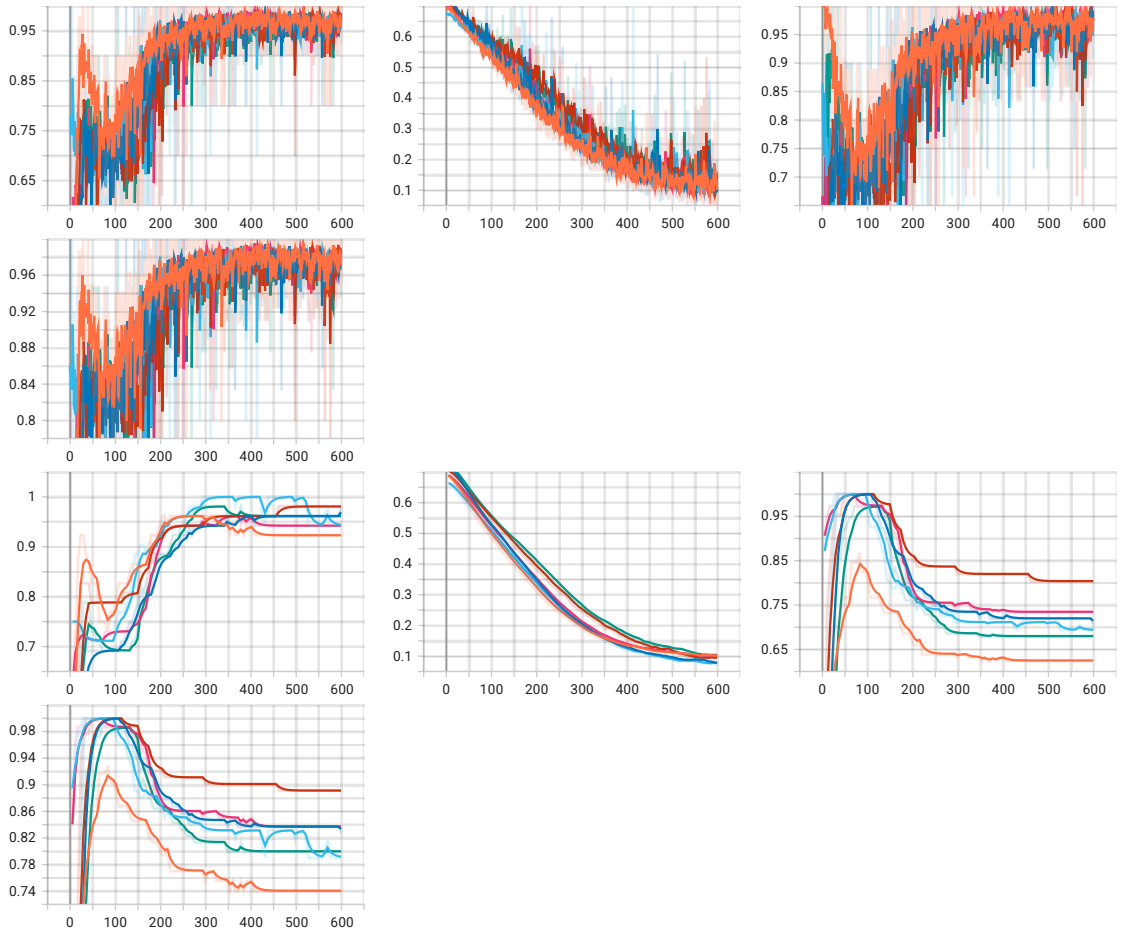
During the testing phase of a CNN+LSTM model, it is checked whether the frames comprising the temporal sequence have a prediction greater than the threshold (set at 0.5). In this case, the sequence is classified as positive.

For each experiment, the obtained graphs on TensorBoard are shown in the following order: train/acc, train/loss, train/precision, train/F-Score, val/acc, val/loss, val/precision and val/F-Score.

Considering the architecture as CNN+LSTM RESNET18, for the experiment:

- 07172023-2:

  The recall is high for train e val.




  The model classifies a sequence as positive even when there is no fire because it confuses it with red or orange-colored houses.

  There are also special cases where the network completely fails in the classification, such as:

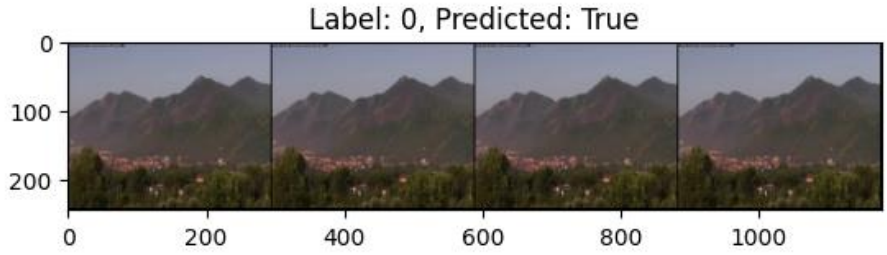
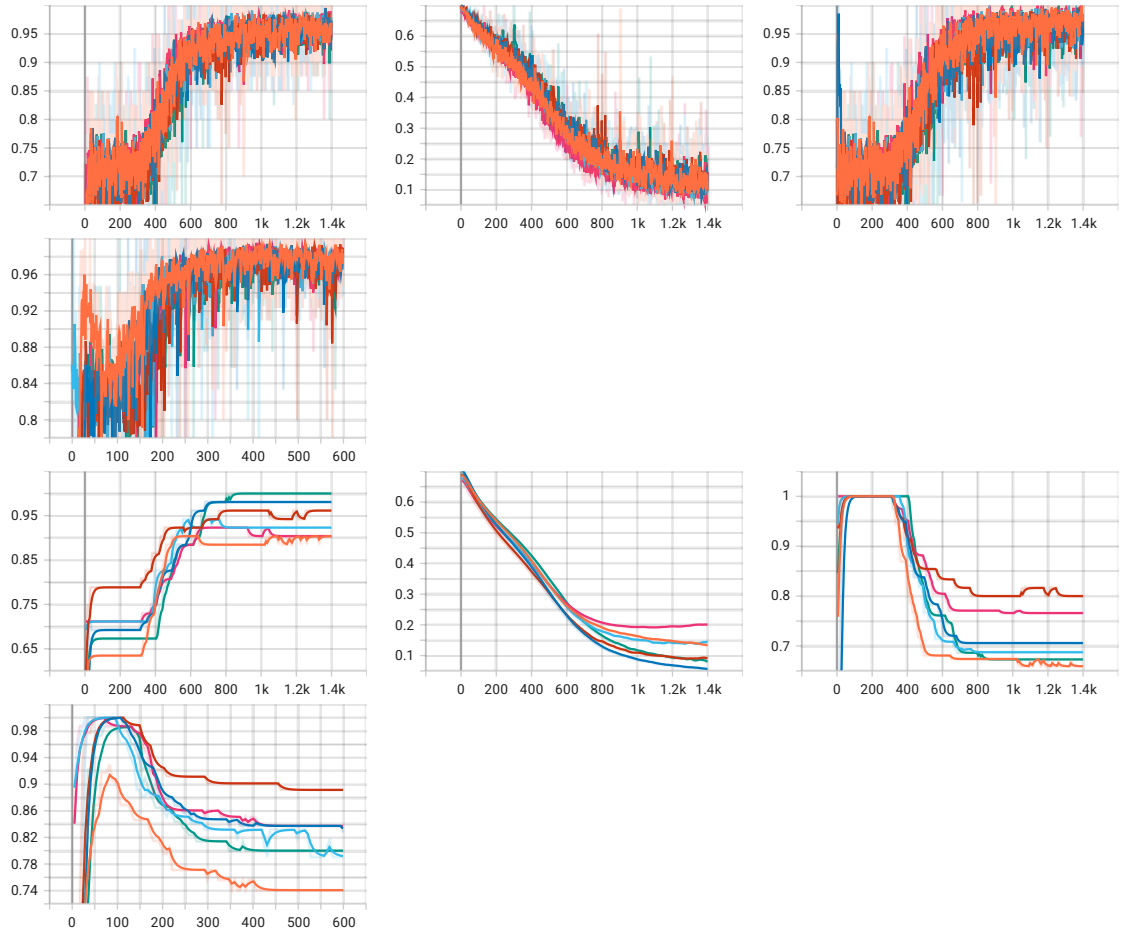



*Figure 5 - Misclassified frame CNN+LSTM ResNet18 07172023-2*

Considering the architecture as CNN+LSTM MobileNetV3, for the different experiments, we obtain:

- 07172023-1:

  As the number of steps increases, the values of acc, precision, and F-Score increase, while the Loss value decreases. The recall remains stable and is approximately equal to 1.



The model incorrectly classifies the sequence if it contains red or orange houses.

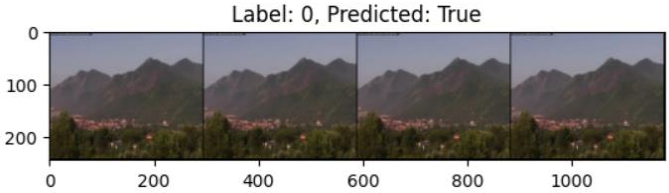
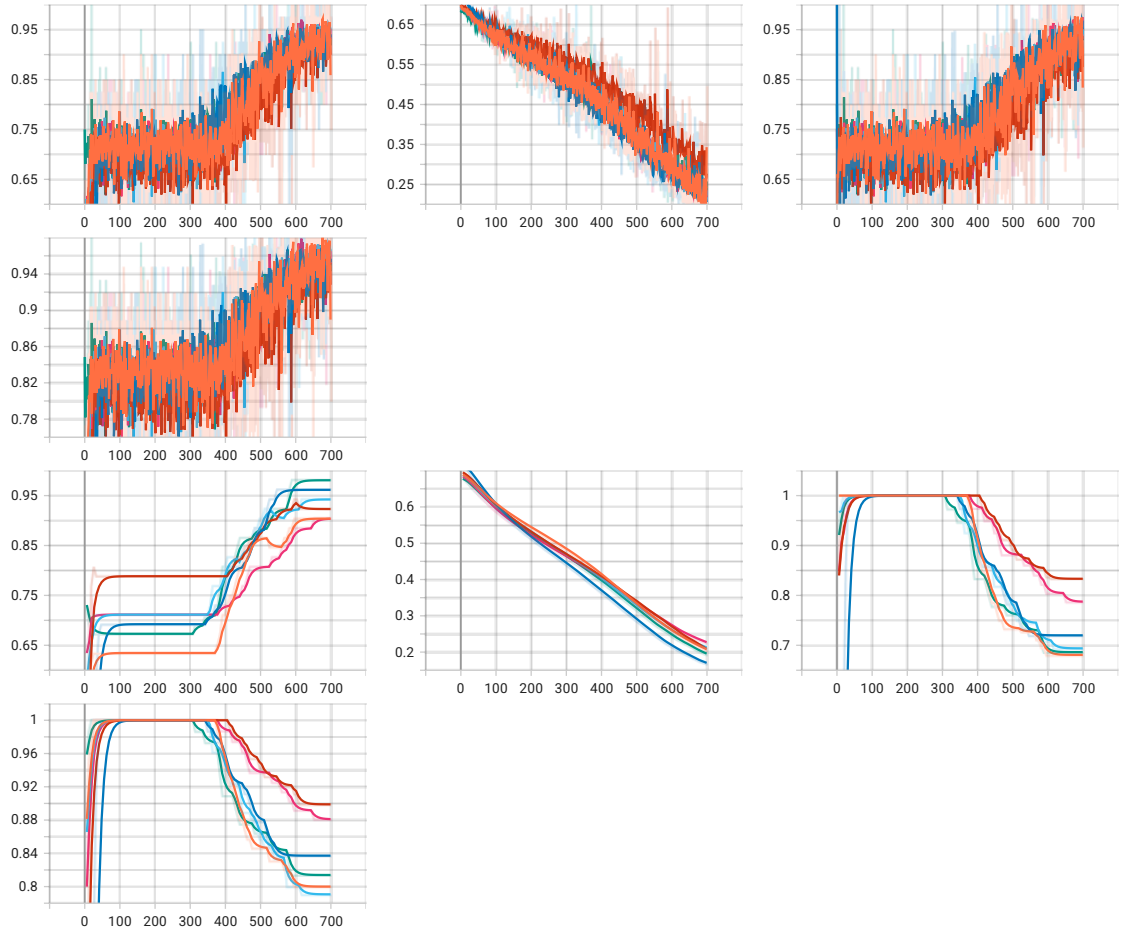



*Figure 6 - Misclassified frame CNN+LSTM MobileNetV3 07172023-1*

- 07172023-2:

The recall is near to 1 and as the number of steps increases, the model's performance improves.




The model incorrectly classifies the sequence if it contains clouds, fog, and if there are houses in red or orange colors.

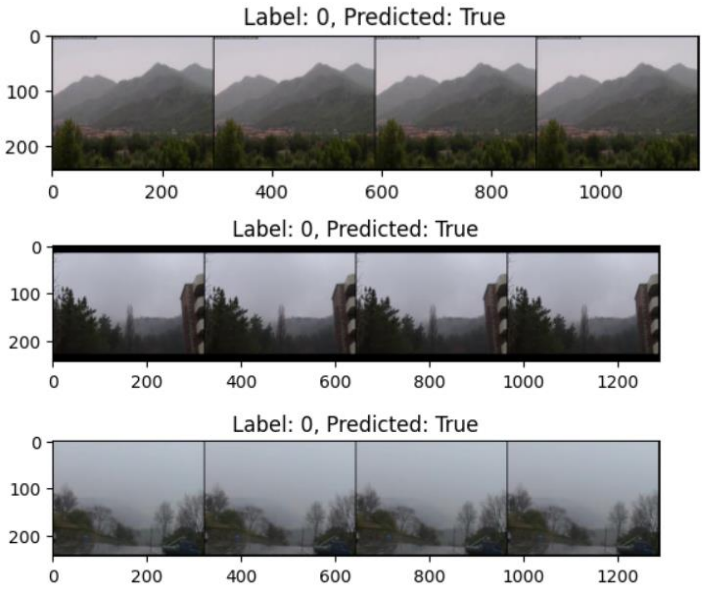
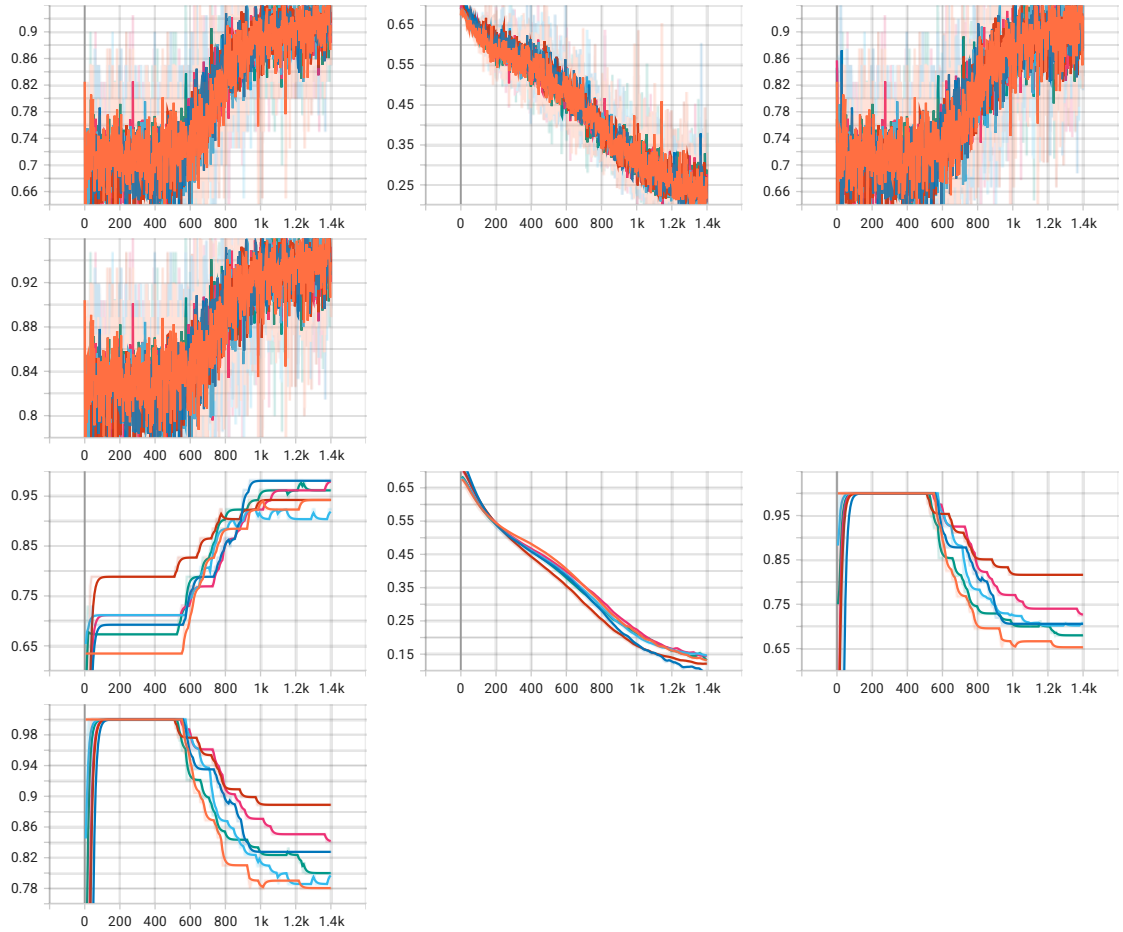



*Figure 7 - Misclassified frame CNN+LSTM MobileNetV3 07172023-2*

- 07172023-4:

The recall is high.




The model incorrectly classifies the sequence as positive because it confuses red objects with flames, for example, a truck. It also makes mistakes in classification in case of fog and haze.

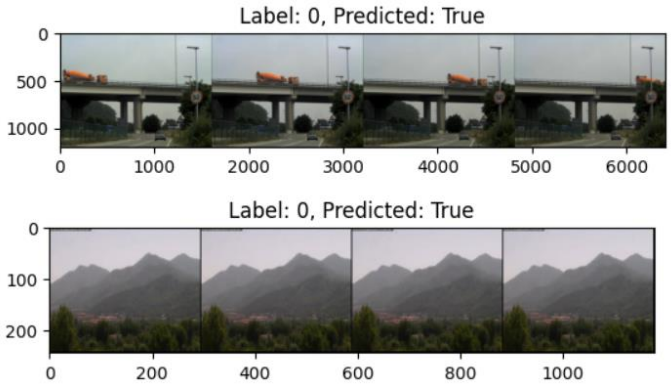



*Figure 8 - Figure 13 - Misclassified frame CNN+LSTM MobileNetV3 07172023-4*

# Experimental Results

## Evaluation Metrics: Measuring Performance of Artificial Intelligence Models

The fire detection accuracy of the competing methods will be evaluated in terms of Precision and Recall. To formalize these metrics, the challenge defines the sets of true positives ($TP$), false positives ($FP$) and false negatives ($FN$). Each positive video shows one fire event only. Each prediction is evaluated by comparing the fire start instant $g$ with the detection one $p$. The ground truth indicates as the fire start instant the first frame in which it is visible. It is manually labelled by the human operator after visioning the whole video, thus it is a ground truth.

The project defines the following classifications:

- $TP$: all the detections occurring in positive videos at $p \geq max(0, g - \Delta t)$;
- $FP$: all the detections occurring at any time in negative videos or in positive videos at $p < max(0, g - \Delta t)$;
- $FN$: the set of positive videos for which no fire detection occurs.

with $\Delta t$ equal to 5 seconds.

Defined these sets for $TP$, $FP$ and $FN$, we can compute the Precision ($P$) and Recall ($R$) with respect to the number of true positives ($|TP|$), false positives ($|FP|$) and false negatives ($|FN|$):

$$P = \frac{|TP|}{|TP| + |FP|}$$

$$R = \frac{|TP|}{|TP| + |FN|}$$

Precision assumes values in the range [0,1] and measures the capability of the methods to reject false positives; the higher is $P$, the higher the specificity of the method. Recall assumes values in the range [0,1] and evaluates the sensitivity of the method to detect fire; the higher is $R$, the higher the sensitivity of the method.

The fire notification time of a method for the i-th true positive is $p_i$ , while the fire start time is $g_i$. Therefore, the notification delay $d_i$ on the i-th sample is:

$$d_i = |p_i - g_i|$$

The average notification delay $D$ is defined as the ratio between the sum of all the notification delays $d_i$ and the total number of true positives $|TP|$:

$$D = \frac{\sum_{i=1}^{|TP|} d_i}{|TP|}$$

The smaller is $D$, the faster the notification of a fire. To obtain a value in the range [0,1], the normalized average notification delay $D_n$ is computed as follows:

$$D_n = \frac{max(0; 60 - D)}{60}$$

To carry out the evaluation of these metrics, we wrote a dedicated code. Specifically, we are concerned with retrieving the results obtained from the execution of the test code on the test set and then calculating these metrics using the formulas above.

## Definition of criteria to classify videos

### CNN models

The models we trained are capable of detecting flames and smoke in individual frames. However, to classify the entire video as positive (indicating the presence of fire), we utilize a slightly different approach. We extract one frame per second from the video and obtain predictions for each frame. To determine if the entire video contains fire, we examine consecutive frames within the video. Specifically, if there are *min_duration* consecutive frames where the predictions are greater than the specified *threshold*, we classify the entire video as positive (fire appears). We also track the detected fire start frame, which is the first frame within the first sequence of *min_duration* frames where the prediction exceeds the *threshold*.

During our testing, we experimented with different values for *min_duration* and *threshold* to assess their impact on model performance. By increasing the *threshold*, we can potentially achieve higher precision, as the model becomes more conservative in classifying positive cases. However, this may introduce a delay in detecting the fire, as the model requires more consecutive frames with high predictions to make a positive classification. Finding the right balance between *threshold* and *min_duration* is crucial to ensure both accuracy and efficiency in fire detection. Setting these parameters appropriately can help optimize the performance of the system in real-world scenarios, where timely and accurate fire detection is essential.

### CNN+LSTM models

Our models are capable of determining the presence of fire (flames and/or smoke) in a sequence of four consecutive frames. To classify the entire video as positive for fire detection, we begin by extracting two frames per second. Using a sliding window with a size of four frames and a stride of two frames, we collect predictions for sequences of frames. This approach allows us to compute predictions on overlapping sequences, enabling the model to consider cases where all frames in a sequence have fire or only some of them. By using a stride of two frames, we obtain predictions at the beginning of each second of the video, reducing the computational workload for the model. To classify the entire video as positive, we check if there are *min_duration* consecutive predictions with values greater than the specified *threshold*. If this condition is met, the entire video is classified as positive for fire detection. The first frame of fire appearance corresponds to the first frame of the sequence of sliding windows we selected, which has a duration of *min_duration*. Thanks to the stride of two frames, we are able to determine the second in which the flame appears.

As before, we conducted experiments with different values of *min_duration* and *threshold* to evaluate their impact on system performance. By increasing the *threshold*, the system focuses on sequences where the fire appears in the first frame, potentially providing a more accurate estimation of the time of appearance.

## Usage of a test set

During the testing phase, a dedicated test dataset was created to evaluate the performance of neural networks trained on new data. Videos of this dataset have been manually labelled, indicating whatever a fire appears or not and indicating, in case of a positive sample, the fire start instance. This test set consists of completely new samples that the network has never encountered before. It was used to compare different algorithms and to compare different folds of the same algorithm.

While it is a common practice to create a standard test set once and use it consistently to evaluate all proposed algorithms for a particular task (as it saves time and facilitates comparisons), this approach may lead to incorrect results over time. Regardless of the test set's size, there is always a probability, even if very small but greater than zero, that the estimated performance on the test set differs significantly from the expected future performance for certain functions.

Since the test set's samples are entirely unseen by the network (never used during the training phase), it serves as a useful, albeit suboptimal, tool to distinguish between different alternatives that have been realized.

To evaluate the performance of the models on the test set, a specialized code was created to automate the calculation of the previously mentioned metrics. Utilizing the predictions obtained from the earlier test code, we carefully initialized and populated specific data structures for calculating comparison parameters, such as precision, recall, and more.

This implementation of the code proved highly valuable, not only for precisely assessing the performance of various models but also for evaluating the errors made in individual videos. For videos without fire, the code indicates whether the network predicted correctly or not. For videos with fire or flames, the code also provides the instant of fire onset along with the correctness of the prediction. This facilitated the selection of the best model.

## Presentation of results obtained

For model selection, we analyzed the results collected during the testing phase and have reported them for better visualization in the "Attachments" chapter on the following pages. Specifically, we created tables categorized according to the representation model, with several alternatives presented for each model. The models used for the results are as follows:

- ResNet50
- ResNet18
- MobileNetV3
- EfficientNetB1
- CNNLSTM ResNet18
- CNNLSTM MobileNetV3

From these results, we selected the "best" alternatives for each model and compiled them in additional summary tables presented later in the "Results" chapter. The chapter includes three tables:

1. Description of models: containing the main information about the models.
2. Training information: providing specific details about the training phase of the models.
3. Results: showing the outcomes obtained during the testing phase, along with the differences between each test.

Based on these three tables, we performed various analyses, ultimately leading us to choose one model considered most suitable for our purpose.

The accuracy values indicate that the performance of the tested ResNets is not promising for real-world applications. However, using MobileNetV3 tends to improve performance, and incorporating LSTM demonstrates a noticeable advantage due to the additional information used for predicting outcomes. Among the different LSTM models, the most convincing one is the configuration where the initial convolutional network is a previously trained MobileNetV3, leading to a proposed solution involving double training:

1. Only MobileNetV3: In this model, we implemented MobileNet through fine-tuning, adjusting the size of the second layer in the dense part, and adding an additional third layer. The training of this model utilized K-Fold cross-validation, and we introduced a single transformation, namely horizontal flip, for augmenting the available data. From the conducted experiments, it was observed that adding other transformations like noise, blur, and reducing image intensity did not result in performance improvement; instead, it had a detrimental effect. This negatively impacted the network's ability to make accurate predictions.
2. MobileNetV3 + LSTM: For the construction of the final model, we began with the convolutional network and integrated Long Short-Term Memory (LSTM) into it. Specifically, the model was designed so that the feature vector input to the LSTM was also passed as output to the MobileNet (thereby removing the fully connected part). The LSTM then concluded with a fully connected layer having a single output, which represented the network's overall output.

Overall, the addition of LSTM brought about modest improvements: a basic convolutional model was unable to achieve high accuracy values because a single frame does not always provide sufficient information to distinguish a video as positive or negative. Having more information to base decisions on, due to the ability to work with sequences of frames rather than individual frames, led to enhanced performance and increased accuracy in the predictions. Consequently, the network now exhibits reduced errors in labelling samples as positive, thereby mitigating the problem of detecting false positives.

Another important factor in selecting the best model was the decision regarding the min_duration and threshold parameters used to classify the entire video as positive. The classification depended on the presence of min_duration consecutive predictions with values higher than the specified threshold.

Various threshold values, ranging from 0.5 to 0.7, and min_duration variable values, ranging from 2 to 4, were tested. However, no definitive criteria were identified to distinguish between models based on these values.

| Videos in Test Set | | Fire detection | |
|---|---|---|---|
| Video N° | Description | MobileNetV3 07082023-2 Fold 4 | CNNLSTM MobileNetV3 07172023-2 Fold 2 |
| Video111 | Moving car with lights on in the night | True Negative | True Negative |
| Video104 | Red baskteball players moving in playground | True Negative | True Negative |
| Video107 | Foggy landscape | False Positive | False Positive |
| Video106 | High Speed Motorway | False Positive | True Negative |
| Video105 | Wind moving tree leaves | False Positive | True Negative |
| Video109 | People waiting on train platform | False Positive | False Positive |
| Video108 | Urban road | False Positive | False Positive |
| Video110 | Las Vegas Red Neon Sign | False Positive | True Negative |
| Video278 | Outdoor controlled fire | True Positive | True Positive |
| Video279 | Indoor controlled fire | True Positive | True Positive |
| Video269 | Outdoor controlled fire | True Positive | True Positive |
| Video272 | Outdoor fire and smoke | True Positive | True Positive |
| Video262 | Incident inside a motorway tunnel | True Positive (real start frame: 3, predicted 0) | True Positive (real start frame: 3, predicted 0) |
| Video263 | Indoor controlled small fire | True Positive (real start frame: 4, predicted: 5) | True Positive (real start frame: 4, predicted: 4) |
| Video264 | Indoor controlled fire | True Positive | True Positive |
| Video265 | Outdoor smoke | True Positive | True Positive |
| Video266 | Indoor candle | True Positive | True Positive |
| Video267 | Outdoor smoke | True Positive (real start frame:5, predicted: 0) | True Positive (real start frame:5, predicted: 0) |
| Video268 | Outdoor smoke on mountain | True Positive | True Positive |
| Video270 | Indoor controlled fire | True Positive | True Positive |
| Video271 | Outdoor fire and smoke | True Positive | True Positive |
| Video273 | Outdoor fire and smoke | True Positive (real start frame: 2, predicted: 0) | True Positive (real start frame: 2, predicted: 0) |
| Video274 | Outdoor smoke on mountain | True Positive | True Positive |
| Video275 | Indoor smoke | True Positive (real start frame: 5, predicted: 0) | True Positive (real start frame: 5, predicted: 0) |
| Video276 | Incident inside a motorway tunnel | True Positive (real start frame: 2, predicted: 0) | True Positive (real start frame: 2, predicted: 0) |
| Video277 | Indoor smoke | True Positive (real start frame: 4, predicted: 0) | True Positive (real start frame: 4, predicted: 1) |

*Table 1: Comparisons between MobileNetV3 only and CNNLSTM MobileNetV3 on test set*

## Discussion of strengths and limitations of the chosen model

The proposed solution has several strenghts and several disadvantages, including:

- Long-range dependencies: LSTM networks can capture long-range dependencies in sequential data.
- Handling vanishing gradients: LSTMs utilize a gating mechanism to control the flow of information, making it easier to train models on long sequences. The gating mechanism allows them to selectively remember or forget information, mitigating the vanishing gradient problem and improving the learning process.
- The Precision value: The precision value of the presented model is 0.857, which is certainly a good result. However, it assumes that the model classifies videos as positive even when there is no fire or smoke present. Therefore, if this system were used in the real world, it could alert authorities even when not necessary.
- Inconsistent detection of the fire's starting frame: The system does not always correctly identify the starting frame of the fire, which is a critical element in determining the system's efficiency. Delayed reporting of a fire could be unhelpful in saving human lives or limiting environmental damage.
- Memory and computational overhead: The proposed solution is not advantageous in terms of memory and operations compared to a simple MobileNetV3. The addition of LSTM introduces increased computational complexity, difficulties in maintaining long-term information, and, due to its recursive nature, is more complex to interpret.
- Insufficient amount of data available for the training phase. CNN+LSTM models require a significant amount of data to learn to recognize fires. With limited data, there is a risk that the network memorizes the few provided

examples and fails to generalize. A restricted dataset may not be representative of real-world scenarios, thus not accounting for the variability of situations where a fire can occur. With limited data available, the resulting system is likely to be unreliable.

# Conclusions

## Summary of key points from the project

The proposed solution was obtained by firstly training a modified MobileNetV3 where the second FC layer was changed and a third one was added. The transformation used is HorizontalFlip, and the threshold is fixed at 0.5. Then, a LSTM (Long Short-Term Memory) was added to the MobileNetV3 (without the classifier part), offering advantages when working with temporal sequences. In the proposed solution, their ability to retain relevant information over time is leveraged, which is then used to classify a video as positive or negative.

After extracting frames from the videos, those used during the training phase are transformed using the HorizontalFlip transformation with a probability of 0.5. During the training phase, hyperparameters such as the number of neurons, number of layers, dropout probability, LSTM size, and others were adjusted to obtain the best possible model.

During the testing phase, however, other values such as those of min_duration and treshold were changed to refine the choice made, although their modification has not always led to better results.

Subsequently, performance was evaluated using metrics such as precision, recall, F-Score, and accuracy. These values were helpful in assessing the model's ability to detect fires when present.

## Assessment of future prospects and potential improvements to the system

In the future, more data could be collected and used to train the network. Indeed, LSTM can achieve better results when they have access to a large amount of training data. In the case of limited datasets, the network's ability to generalize may be compromised.

More advanced data augmentation techniques could be used to create artificial data from existing data. By doing so, it increases the diversity of the data used during the training phase. In particular, the system could be made more robust to different environmental conditions, such as adverse weather conditions or variations in lighting.

To improve the performance of the proposed system, one can consider implementing continuous learning. With continuous learning mechanisms, it is possible to update and enhance the model by retraining it with new data. This way, the system will be updated and reliable.

# Results

*Tabella 3: Description of models*

| Tipologia di rete | Esperimento | Numero di segmenti per video | Numero di frame per segmento | Dimensione degli strati della FC | input_size | hidden_size | num_layers | bidirectional | requires_grad | Trasformazioni | Ordine applicazione delle trasformazioni | Rete di partenza |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ResNet50 | 07082023-1 | 1 | 1 | [2048, 1024, 512, 1] | - | - | - | - | TRUE | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augmentation, Normalize | - |
| ResNet50 | 07092023-1 | 1 | 1 | [2048, 1] | - | - | - | - | TRUE | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augmentation, Normalize | - |
| ResNet18 | 06262023-2 | 1 | 1 | [512, 256, 1] | - | - | - | - | TRUE | HorizontalFlip | preprocessing, augumentation | - |
| MobileNetV3 | 07032023-1 | 1 | 1 | [960, 1280, 1] | - | - | - | - | TRUE | HorizontalFlip | preprocessing, augumentation | - |
| MobileNetV3 | 07052023-1 | 1 | 1 | [960, 1280, 1] | - | - | - | - | TRUE | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | - |
| MobileNetV3 | 07082023-2 | 1 | 1 | [960, 1280, 1280, 1] | - | - | - | - | TRUE | HorizontalFlip | preprocessing, augumentation | - |
| CNNLSTM ResNet18 | 07132023-1 | 1 | 4 | [200, 1] | 512 | 100 | 2 | TRUE | TRUE | HorizontalFlip | preprocessing | ResNet18 |
| CNNLSTM Resnet18 | 07172023-2 | 1 | 4 | [200, 1] | 512 | 100 | 2 | TRUE | TRUE | HorizontalFlip | preprocessing, augumentation | ResNet18 |
| CNNLSTM MobileNetV3 | 07172023-1 | 1 | 4 | [200, 1] | 960 | 100 | 2 | TRUE | FALSE | HorizontalFlip | preprocessing | 07082023-2 |
| CNNLSTM MobileNetV3 | 07172023-2 | 1 | 4 | [200, 1] | 960 | 100 | 2 | TRUE | FALSE | HorizontalFlip | preprocessing | 07082023-2 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 4 | [200, 1] | 960 | 100 | 2 | TRUE | FALSE | HorizontalFlip, GaussNoise, Blur | preprocessing | 07052023-1 |

*Tabella 4: Training informations*

| Tipologia di rete | Esperimento | Batch_size | lr | Momentum | lambda_reg | epochs | early_stopping_patience |
|---|---|---|---|---|---|---|---|
| ResNet50 | 07082023-1 | 50 | 0,001 | 0,9 | 0 | 100 | 20 |
| ResNet50 | 07092023-1 | 50 | 0,001 | 0,9 | 0 | 100 | 20 |
| ResNet18 | 06262023-2 | 50 | 0,001 | 0,9 | 0 | 300 | 40 |
| MobileNetV3 | 07032023-1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 |
| MobileNetV3 | 07052023-1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 |
| MobileNetV3 | 07082023-2 | 50 | 0,001 | 0,9 | 0 | 100 | 40 |
| CNNLSTM ResNet18 | 07132023-1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 |
| CNNLSTM Resnet18 | 07172023-2 | 50 | 0,001 | 0,9 | 0 | 100 | 40 |
| CNNLSTM MobileNetV3 | 07172023-1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 |
| CNNLSTM MobileNetV3 | 07172023-2 | 40 | 0,001 | 0,9 | 0 | 200 | 40 |
| CNNLSTM MobileNetV3 | 07172023-4 | 40 | 0,001 | 0,9 | 0 | 200 | 40 |

*Tabella 5: Results*

| Tipologia di rete | Esperimento | Fold | Threshold | Min Durata | TP | TN | FP | FN | Accuracy | Precision | Recall | F-Score | Average Delay | Normalized Average Delay |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ResNet50 | 07082023-1 | 3 | 0.5 | 5 | 18 | 0 | 8 | 0 | 0,6923076923 | 0,6923076923 | 1 | 0,8181818182 | 1,389 | 0,97685 |
| ResNet50 | 07092023-1 | 2 | 0,5 | 5 | 17 | 1 | 7 | 1 | 0,6923076923 | 0,7083333333 | 0,9444444444 | 0,8095238095 | 1,471 | 0,9754833333 |
| ResNet18 | 06262023-2 | 4 | 0.5 | 5 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,333 | 0,9777833333 |
| ResNet18 | 06262023-2 | 4 | 0.7 | 5 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,5 | 0,975 |
| MobileNetV3 | 07032023-1 | 2 | 0,8 | 5 | 17 | 3 | 5 | 1 | 0,7692307692 | 0,7727272727 | 0,9444444444 | 0,85 | 1,941 | 0,96765 |
| MobileNetV3 | 07052023-1 | 5 | 0,7 | 5 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 2,611 | 0,9564833333 |
| MobileNetV3 | 07082023-2 | 4 | 0,5 | 5 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,222 | 0,98 |
| CNNLSTM ResNet18 | 07132023-1 | 1 | 0,7 | 2 | 18 | 5 | 3 | 0 | 0,8846153846 | 0,8571428571 | 1 | 0,9230769231 | 1,5 | 0,975 |
| CNNLSTM Resnet18 | 07172023-2 | 2 | 0,7 | 2 | 17 | 5 | 3 | 1 | 0,8461538462 | 0,85 | 0,9444444444 | 0,8947368421 | 0,941 | 0,984 |
| CNNLSTM MobileNetV3 | 07172023-1 | 5 | 0,7 | 2 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,167 | 0,98055 |
| CNNLSTM MobileNetV3 | 07172023-2 | 2 | 0,7 | 2 | 18 | 5 | 3 | 0 | 0,8846153846 | 0,8571428571 | 1 | 0,9230769231 | 1,056 | 0,9824 |
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 0,7 | 2 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,556 | 0,9740666667 |

# Attachments

*Tabella 6: ResNet50 experiments*

| Tipologia di rete | Esperimento | Fold | Threshold | Min Durata | TP | TN | FP | FN | Accuracy | Precision | Recall | F-Score | Average Delay | Normalized Average Delay | Numero di segmenti per video | Numero di frame per segmento | Dimensione degli strati della FC | Batch_size | lr | Momentum | lambda_reg | epochs | early_stopping_patience | Trasformazioni | Ordine applicazzione delle trasformazioni |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ResNet50 | 07052023-1 | 2 | 0.8 | 5 | 15 | 6 | 2 | 3 | 0,8076923077 | 0,8823529412 | 0,8333333333 | 0,8571428571 | 4,2 | 0,93 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 2 | 0.8 | 10 | 14 | 7 | 1 | 4 | 0,8076923077 | 0,9333333333 | 0,7777777778 | 0,8484848485 | 4,286 | 0,929 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 2 | 0.5 | 10 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 2,706 | 0,955 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 3 | 0,7 | 5 | 15 | 5 | 2 | 3 | 0,8 | 0,8823529412 | 0,8333333333 | 0,8571428571 | 2,867 | 0,952 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 3 | 0.5 | 5 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 1,824 | 0,97 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 3 | 0.8 | 5 | 13 | 6 | 2 | 5 | 0,7307692308 | 0,8666666667 | 0,7222222222 | 0,7878787879 | 3,692 | 0,938 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 3 | 0.8 | 10 | 12 | 7 | 1 | 6 | 0,7307692308 | 0,9230769231 | 0,6666666667 | 0,7741935484 | 4,083 | 0,932 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 2 | 0.7 | 5 | 17 | 5 | 3 | 1 | 0,8461538462 | 0,85 | 0,9444444444 | 0,8947368421 | 4,588 | 0,924 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 2 | 0.5 | 5 | 17 | 1 | 7 | 1 | 0,6923076923 | 0,7083333333 | 0,9444444444 | 0,8095238095 | 1,412 | 0,976 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07052023-1 | 3 | 0.5 | 10 | 16 | 2 | 6 | 2 | 0,6923076923 | 0,7272727273 | 0,8888888889 | 0,8 | 1,562 | 0,974 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet50 | 07082023-1 | 3 | 0.5 | 5 | 18 | 0 | 8 | 0 | 0,6923076923 | 0,6923076923 | 1 | 0,8181818182 | 1,389 | 0,97685 | 1 | 1 | [2048,1024.512,1] | 50 | 0,001 | 0,9 | 0 | 100 | 20 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize |
| ResNet50 | 07092023-1 | 2 | 0,7 | 5 | 16 | 3 | 5 | 2 | 0,7307692308 | 0,7619047619 | 0,8888888889 | 0,8205128205 | 2,062 | 0,966 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 20 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize |
| ResNet50 | 07092023-1 | 2 | 0,8 | 5 | 16 | 5 | 3 | 2 | 0,8076923077 | 0,8421052632 | 0,8888888889 | 0,8648648649 | 5,625 | 0,906 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 20 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize |
| ResNet50 | 07092023-1 | 2 | 0,5 | 5 | 17 | 1 | 7 | 1 | 0,6923076923 | 0,7083333333 | 0,9444444444 | 0,8095238095 | 1,471 | 0,9754833333 | 1 | 1 | [2048,1] | 50 | 0,001 | 0,9 | 0 | 100 | 20 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize |
| ResNet50 | 07092023-2 | 3 | 0.5 | 5 | 18 | 0 | 8 | 0 | 0,6923076923 | 0,6923076923 | 1 | 0,8181818182 | 1,389 | 0,97685 | 1 | 1 | [2048,1024.512,1] | 50 | 0,001 | 0,9 | 0 | 100 | 20 | Nessuna trasformazione | preprocessing, Normalize |

*Tabella 7: ResNet18 experiments*

| Tipologia di rete | Esperimento | Fold | Threashold | Min Durata | TP | TN | FP | FN | Accuracy | Precision | Recall | F-Score | Average Delay | Normalized Average Delay | Numero di segmenti per video | Numero di frame per segmento | Dimensione degli strati della FC | Batch_size | lr | Momentum | lambda_reg | epochs | early_stopping_patience | Trasformazioni | Ordine applicazzione delle trasformazioni |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ResNet18 | 06262023-2 | 4 | 0.8 | 5 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,5 | 0,975 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 4 | 0,5 | 10 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,333 | 0,9777833333 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 4 | 0.8 | 10 | 17 | 4 | 4 | 1 | 0,8076923077 | 0,8095238095 | 0,9444444444 | 0,8717948718 | 1,824 | 0,9696 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 4 | 0.5 | 5 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,333 | 0,9777833333 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 4 | 0.7 | 5 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,5 | 0,975 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 3 | 0,5 | 10 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,667 | 0,9722166667 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 3 | 0,5 | 5 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,333 | 0,9777833333 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 3 | 0,8 | 10 | 14 | 4 | 4 | 4 | 0,6923076923 | 0,7777777778 | 0,7777777778 | 0,7777777778 | 2,071 | 0,9654833333 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 06262023-2 | 3 | 0,8 | 5 | 15 | 3 | 5 | 3 | 0,6923076923 | 0,75 | 0,8333333333 | 0,7894736842 | 3,533 | 0,9411166667 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 300 | 40 | HorizontalFlip | preprocessing, augumentation |
| ResNet18 | 7082023 | 2 | 0,5 | 5 | 18 | 0 | 8 | 0 | 0,6923076923 | 0,6923076923 | 1 | 0,8181818182 | 1,389 | 0,97685 | 1 | 1 | [512,256,128,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize |
| ResNet18 | 07162023-1 | 4 | 0,5 | 15 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,667 | 0,9722166667 | 1 | 1 | [512,256,1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize |

*Tabella 8: MobileNetV3 experiments*

| Tipologia di rete | Esperimento | Fold | Threshold | Min Durata | TP | TN | FP | FN | Accuracy | Precision | Recall | F-Score | Average Delay | Normalized Average Delay | Numero di segmenti per video | Numero di frame per segmento | Dimensione degli strati della FC | Batch_size | lr | Momentum | lambda_reg | epochs | early_stopping_patience | Trasformazioni | Ordine applicazzione delle trasformazioni | Note aggiuntive |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MobileNetV3 | 06282023-1 | 1 | 0,8 | 5 | 16 | 5 | 3 | 2 | 0,8076923077 | 0,8421052632 | 0,8888888889 | 0,8648648649 | 3,688 | 0,9385333333 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 06282023-1 | 1 | 0,8 | 10 | 15 | 5 | 3 | 3 | 0,7692307692 | 0,8333333333 | 0,8333333333 | 0,8333333333 | 4,8 | 0,92 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 06282023-1 | 1 | 0,5 | 5 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,444 | 0,9759333333 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 06282023-1 | 1 | 0,5 | 10 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,444 | 0,9759333333 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07032023-1 | 2 | 0,5 | 5 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,944 | 0,9676 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07032023-1 | 2 | 0,5 | 10 | 17 | 1 | 7 | 1 | 0,6923076923 | 0,7083333333 | 0,9444444444 | 0,8095238095 | 1,294 | 0,9784333333 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07032023-1 | 2 | 0,8 | 5 | 17 | 3 | 5 | 1 | 0,7692307692 | 0,7727272727 | 0,9444444444 | 0,85 | 1,941 | 0,96765 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07032023-1 | 2 | 0,8 | 10 | 16 | 5 | 3 | 2 | 0,8076923077 | 0,8421052632 | 0,8888888889 | 0,8648648649 | 6,312 | 0,8948 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07032023-1 | 2 | 0,7 | 5 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 1,176 | 0,9804 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07042023-1 | 2 | 0,5 | 5 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,5 | 0,975 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07042023-1 | 2 | 0,5 | 10 | 17 | 1 | 7 | 1 | 0,6923076923 | 0,7083333333 | 0,9444444444 | 0,8095238095 | 2 | 0,9666666667 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07042023-1 | 2 | 0,8 | 5 | 8 | 8 | 0 | 10 | 0,6153846154 | 1 | 0,4444444444 | 0,6153846154 | 2,375 | 0,9604166667 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07042023-1 | 2 | 0,8 | 10 | 6 | 8 | 0 | 12 | 0,5384615385 | 1 | 0,3333333333 | 0,5 | 6,833 | 0,8861166667 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 5 | 0,5 | 5 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,5 | 0,975 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 5 | 0,8 | 5 | 16 | 5 | 3 | 2 | 0,8076923077 | 0,8421052632 | 0,8888888889 | 0,8648648649 | 6,438 | 0,8927 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | é stato cambiato |

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | il secondo livello |
| MobileNetV3 | 07052023-1 | 5 | 0,7 | 5 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 2,611 | 0,9564833333 | | | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 5 | 0,5 | 10 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,5 | 0,975 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 2 | 0,5 | 10 | 17 | 3 | 5 | 1 | 0,7692307692 | 0,7727272727 | 0,9444444444 | 0,85 | 1,353 | 0,97745 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 2 | 0,5 | 5 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 1,353 | 0,97745 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 2 | 0,8 | 5 | 14 | 5 | 3 | 4 | 0,7307692308 | 0,8235294118 | 0,7777777778 | 0,8 | 2,357 | 0,9607166667 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | è stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 5 | 0,8 | 10 | 11 | 6 | 2 | 7 | 0,6538461538 | 0,8461538462 | 0,6111111111 | 0,7096774194 | 4,182 | 0,9303 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07052023-1 | 2 | 0,8 | 10 | 12 | 5 | 3 | 6 | 0,6538461538 | 0,8 | 0,6666666667 | 0,7272727273 | 6,917 | 0,8847166667 | 1 | 1 | [960, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | é stato cambiato il secondo livello |
| MobileNetV3 | 07082023-1 | 2 | 0,5 | 10 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,167 | 0,981 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize | é stato cambiato il secondo livello e aggiunto un terzo |
| MobileNetV3 | 07082023-1 | 2 | 0,5 | 5 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,167 | 0,981 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize | é stato cambiato il secondo livello e aggiunto un terzo |
| MobileNetV3 | 07082023-1 | 2 | 0,8 | 5 | 11 | 4 | 4 | 7 | 0,5769230769 | 0,7333333333 | 0,6111111111 | 0,6666666667 | 1,727 | 0,971 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize | é stato cambiato il secondo livello e aggiunto un terzo |
| MobileNetV3 | 07082023-1 | 2 | 0,8 | 10 | 9 | 4 | 4 | 9 | 0,5 | 0,6923076923 | 0,5 | 0,5806451613 | 4,556 | 0,924 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur, CLAHE | preprocessing, augumentation, Normalize | é stato cambiato il secondo livello e aggiunto un terzo |
| MobileNetV3 | 07082023-2 | 4 | 0,5 | 5 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,222 | 0,98 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello e aggiunto un terzo |
| MobileNetV3 | 07082023-2 | 4 | 0,5 | 10 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,222 | 0,98 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello e aggiunto un terzo |
| MobileNetV3 | 07082023-2 | 4 | 0,8 | 5 | 16 | 3 | 5 | 2 | 0,7307692308 | 0,7619047619 | 0,8888888889 | 0,8205128205 | 5,812 | 0,903 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello e aggiunto un terzo |

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MobileNetV3 | 07082023-2 | 4 | 0,8 | 10 | 13 | 4 | 4 | 5 | 0,6538461538 | 0,7647058824 | 0,7222222222 | 0,7428571429 | 10,231 | 0,829 | 1 | 1 | [960, 1280, 1280, 1] | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | é stato cambiato il secondo livello e aggiunto un terzo |

*Tabella 9: EfficientNetB1 experiments*

| Tipologia di rete | Esperimento | Fold | Threshold | Min Durata | TP | TN | FP | FN | Accuracy | Precision | Recall | F-Score | Average Delay | Normalized Average Delay | Numero di segmenti per video | Numero di frame per segmento | Dimensione degli strati della FC | Batch_size | lr | Momentum | lambda_reg | epochs | early_stopping_patience | Trasformazioni | Ordine applicazzione delle trasformazioni | Note aggiuntive |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EfficientNetB1 | 07092023-1 | 2 | 0,5 | 5 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 1,529 | 0,9745166667 | 1 | 1 | | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo layer |
| EfficientNetB1 | 07092023-1 | 2 | 0,7 | 5 | 10 | 7 | 1 | 8 | 0,6538461538 | 0,9090909091 | 0,5555555556 | 0,6896551724 | 6,3 | | 1 | 1 | | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo layer |
| EfficientNetB1 | 07092023-1 | 4 | 0,5 | 5 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,389 | 0,97685 | 1 | 1 | | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo layer |
| EfficientNetB1 | 07092023-2 | 2 | 0,5 | 5 | 18 | 0 | 8 | 0 | 0,6923076923 | 0,6923076923 | 1 | 0,8181818182 | 1,389 | 0,97685 | 1 | 1 | | 50 | 0,001 | 0,9 | 0 | 50 | 20 | HorizontalFlip | preprocessing, augumentation | è stato cambiato il secondo layer |
| EfficientNetB1 | 07102023-1 | 2 | 0,5 | 5 | 17 | 1 | 7 | 1 | 0,6923076923 | 0,7083333333 | 0,9444444444 | 0,8095238095 | 1,882 | 0,9686333333 | 1 | 1 | | 50 | 0,001 | 0,9 | 0 | 50 | 20 | HorizontalFlip, GaussNoise, Blur | preprocessing, augumentation | è stato cambiato il secondo layer |
| EfficientNetB1 | 07132023-1 | 3 | 0,5 | 5 | 18 | 0 | 8 | 0 | 0,6923076923 | 0,6923076923 | 1 | 0,8181818182 | 2,222 | 0,9629666667 | 1 | 1 | | 50 | 0,001 | 0,9 | 0 | 50 | 20 | HorizontalFlip, GaussNoise, Blur, Clahe | preprocessing, augumentation | è stato cambiato il secondo layer |

*Tabella 10: CNNLSTM ResNet18 experiments*

| Tipologia di rete | Esperimento | Fold | Threshold | Min Durata | TP | TN | FP | FN | Accuracy | Precision | Recall | F-Score | Average Delay | Normalized Average Delay | Numero di segmenti per video | Numero di frame per segmento | Dimensione degli strati della FC | Batch_size | lr | Momentum | lambda_reg | epochs | early_stopping_patience | Trasformazioni | Ordine applicazzione delle trasformazioni | Note aggiuntive |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNNLSTM ResNet18 | 07132023-1 | 1 | 0,5 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,389 | 0,97685 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM ResNet18 | 07132023-1 | 1 | 0,6 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,278 | 0,9787 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM ResNet18 | 07132023-1 | 1 | 0,7 | 2 | 18 | 5 | 3 | 0 | 0,8846153846 | 0,8571428571 | 1 | 0,9230769231 | 1,5 | 0,975 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM ResNet18 | 07172023-1 | 1 | 0,7 | 2 | 15 | 3 | 5 | 3 | 0,6923076923 | 0,75 | 0,8333333333 | 0,7894736842 | 1,333 | 0,978 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 1 | 0,6 | 2 | 15 | 3 | 5 | 3 | 0,6923076923 | 0,75 | 0,8333333333 | 0,7894736842 | 1 | 0,983 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 1 | 0,5 | 2 | 16 | 3 | 5 | 2 | 0,7307692308 | 0,7619047619 | 0,8888888889 | 0,8205128205 | 1,625 | 0,973 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 1 | 0,7 | 4 | 14 | 3 | 5 | 4 | 0,6538461538 | 0,7368421053 | 0,7777777778 | 0,7567567568 | 1,643 | 0,973 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 1 | 0,6 | 4 | 14 | 3 | 5 | 4 | 0,6538461538 | 0,7368421053 | 0,7777777778 | 0,7567567568 | 0,857 | 0,986 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 1 | 0,5 | 4 | 14 | 3 | 5 | 4 | 0,6538461538 | 0,7368421053 | 0,7777777778 | 0,7567567568 | 1,286 | 0,979 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 2 | 0,7 | 2 | 16 | 3 | 5 | 2 | 0,7307692308 | 0,7619047619 | 0,8888888889 | 0,8205128205 | 1,438 | 0,976 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 2 | 0,6 | 2 | 17 | 3 | 5 | 1 | 0,7692307692 | 0,7727272727 | 0,9444444444 | 0,85 | 1,529 | 0,9745166667 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 2 | 0,5 | 2 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 1,353 | 0,97745 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 2 | 0,7 | 4 | 16 | 3 | 5 | 2 | 0,7307692308 | 0,7619047619 | 0,8888888889 | 0,8205128205 | 1,438 | 0,976 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 2 | 0,6 | 4 | 16 | 3 | 5 | 2 | 0,7307692308 | 0,7619047619 | 0,8888888889 | 0,8205128205 | 1,438 | 0,976 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM ResNet18 | 07172023-1 | 2 | 0,5 | 4 | 16 | 3 | 5 | 2 | 0,7307692308 | 0,7619047619 | 0,8888888889 | 0,8205128205 | 1,25 | 0,976 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 200 | | HorizontalFlip | preprocessing | addestrata da 0 |
| CNNLSTM Resnet18 | 07172023-2 | 3 | 0,5 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,167 | 0,981 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 3 | 0,6 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,444 | 0,976 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 3 | 0,7 | 2 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,444 | 0,976 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 3 | 0,5 | 4 | 17 | 3 | 5 | 1 | 0,7692307692 | 0,7727272727 | 0,9444444444 | 0,85 | 1,294 | 0,978 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 3 | 0,6 | 4 | 17 | 4 | 4 | 1 | 0,8076923077 | 0,8095238095 | 0,9444444444 | 0,8717948718 | 1,353 | 0,977 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 3 | 0,7 | 4 | 16 | 4 | 4 | 2 | 0,7692307692 | 0,8 | 0,8888888889 | 0,8421052632 | 1,25 | 0,979 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 2 | 0,5 | 2 | 17 | 3 | 5 | 1 | 0,7692307692 | 0,7727272727 | 0,9444444444 | 0,85 | 1 | 0,983 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 2 | 0,6 | 2 | 17 | 4 | 4 | 1 | 0,8076923077 | 0,8095238095 | 0,9444444444 | 0,8717948718 | 1 | 0,983 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNNLSTM Resnet18 | 07172023-2 | 2 | 0,7 | 2 | 17 | 5 | 3 | 1 | 0,8461538462 | 0,85 | 0,9444444444 | 0,8947368421 | 0,941 | 0,984 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 2 | 0,5 | 4 | 16 | 4 | 4 | 2 | 0,7692307692 | 0,8 | 0,8888888889 | 0,8421052632 | 0,875 | 0,985 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 2 | 0,6 | 4 | 16 | 4 | 4 | 2 | 0,7692307692 | 0,8 | 0,8888888889 | 0,8421052632 | 0,875 | 0,985 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |
| CNNLSTM Resnet18 | 07172023-2 | 2 | 0,7 | 4 | 16 | 5 | 3 | 2 | 0,8076923077 | 0,8421052632 | 0,8888888889 | 0,8648648649 | 1 | 0,983 | 1 | 4 | 1 | 50 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip | preprocessing, augumentation | Da Resnet18 esperimento 06262023-2 |

*Tabella 11: CNNLSTM MobileNetV3 experiments*

| Tipologia di rete | Esperimento | Fold | Threshold | Min Durata | TP | TN | FP | FN | Accuracy | Precision | Recall | F-Score | Average Delay | Normalized Average Delay | Numero di segmenti per video | Numero di frame per segmento | Dimensione degli strati della FC | Batch_size | lr | Momentum | lambda_reg | epochs | early_stopping_patience | Trasformazioni | Ordine applicazzione delle trasformazioni | Note aggiuntive |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNNLSTM MobileNetV3 | 07162023-1 | 1 | 0,5 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,333 | 0,9777833333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07162023-1 | 1 | 0,6 | 2 | 18 | 5 | 3 | 0 | 0,8846153846 | 0,8571428571 | 1 | 0,9230769231 | 1,556 | 0,9740666667 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07162023-1 | 1 | 0,7 | 2 | 18 | 6 | 2 | 0 | 0,9230769231 | 0,9 | 1 | 0,9473684211 | 1,611 | 0,97315 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 2 | 0,5 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,167 | 0,98055 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 2 | 0,6 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,167 | 0,98055 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 2 | 0,7 | 2 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,333 | 0,9777833333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 1 | 0,5 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,167 | 0,98055 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 1 | 0,6 | 2 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,222 | 0,9796333333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 1 | 0,7 | 2 | 17 | 5 | 3 | 1 | 0,8461538462 | 0,85 | 0,9444444444 | 0,8947368421 | 1,235 | 0,9794166667 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 5 | 0,5 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,167 | 0,98055 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 5 | 0,6 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,167 | 0,98055 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-1 | 5 | 0,7 | 2 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,167 | 0,98055 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo |
| CNNLSTM MobileNetV3 | 07172023-2 | 2 | 0,5 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,222 | 0,9796333333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo, esperimento di partenza 07082023-2 |
| CNNLSTM MobileNetV3 | 07172023-2 | 2 | 0,6 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,167 | 0,98055 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo, esperimento di partenza 07082023-2 |
| CNNLSTM MobileNetV3 | 07172023-2 | 2 | 0,7 | 2 | 18 | 5 | 3 | 0 | 0,8846153846 | 0,8571428571 | 1 | 0,9230769231 | 1,056 | 0,9824 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo, esperimento di partenza 07082023-2 |
| CNNLSTM MobileNetV3 | 07172023-2 | 1 | 0,5 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,222 | 0,9796333333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo, esperimento di partenza 07082023-2 |
| CNNLSTM MobileNetV3 | 07172023-2 | 1 | 0,6 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,222 | 0,9796333333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo, esperimento di |

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | partenza 07082023-2 |
| CNNLSTM MobileNetV3 | 07172023-2 | 1 | 0,7 | 2 | 17 | 4 | 4 | 1 | 0,8076923077 | 0,8095238095 | 0,9444444444 | 0,8717948718 | 1,176 | 0,9804 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip | preprocessing | Estratti due frame per secondo, esperimento di partenza 07082023-2 |
| CNNLSTM MobileNetV3 | 07172023-3 | 2 | 0,5 | 2 | 18 | 1 | 7 | 0 | 0,7307692308 | 0,72 | 1 | 0,8372093023 | 1,389 | 0,97685 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-3 | 2 | 0,6 | 2 | 18 | 3 | 5 | 0 | 0,8076923077 | 0,7826086957 | 1 | 0,8780487805 | 1,389 | 0,97685 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-3 | 2 | 0,7 | 2 | 17 | 8 | 0 | 1 | 0,9615384615 | 1 | 0,9444444444 | 0,9714285714 | 1,471 | 0,9754833333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-3 | 2 | 0,5 | 4 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 1,294 | 0,9784333333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-3 | 2 | 0,6 | 4 | 17 | 3 | 5 | 1 | 0,7692307692 | 0,7727272727 | 0,9444444444 | 0,85 | 1,294 | 0,9784333333 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-3 | 2 | 0,7 | 4 | 16 | 8 | 0 | 2 | 0,9230769231 | 1 | 0,8888888889 | 0,9411764706 | 6,125 | 0,8979166667 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 100 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 0,5 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,278 | 0,9787 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 0,6 | 2 | 18 | 2 | 6 | 0 | 0,7692307692 | 0,75 | 1 | 0,8571428571 | 1,278 | 0,9787 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 0,7 | 2 | 18 | 4 | 4 | 0 | 0,8461538462 | 0,8181818182 | 1 | 0,9 | 1,556 | 0,9740666667 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 0,5 | 4 | 17 | 2 | 6 | 1 | 0,7307692308 | 0,7391304348 | 0,9444444444 | 0,8292682927 | 1,176 | 0,9804 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 0,6 | 4 | 17 | 5 | 3 | 1 | 0,8461538462 | 0,85 | 0,9444444444 | 0,8947368421 | 1,412 | 0,9764666667 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di |

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | partenza 07052023-1 |
| CNNLSTM MobileNetV3 | 07172023-4 | 1 | 0,7 | 4 | 16 | 6 | 2 | 2 | 0,8461538462 | 0,8888888889 | 0,8888888889 | 0,8888888889 | 2,312 | 0,9614666667 | 1 | 4 | 1 | 40 | 0,001 | 0,9 | 0 | 200 | 40 | HorizontalFlip, GaussNoise, Blur | preprocessing | Estratti due frame per secondo, esperimento di partenza 07052023-1 |