FIRE DETECTION IN VIDEO SURVEILLANCE APPLICATIONS

A PROJECT REPORT

Submitted by

RESHMA KRISHNAN (TKM20MCA2030)

to

The APJ Abdul Kalam Technological University

in partial fulfilment of requirements for the award of degree of

MASTER OF COMPUTER APPLICATIONS

Thangal Kunju Musaliar College of Engineering
Kerala

DEPARTMENT OF COMPUTER APPLICATION

JULY 2022

DECLARATION

I undersigned hereby declare that the project report titled **“FIRE DETECTION IN VIDEO SURVEILLANCE APPLICATIONS**”, submitted for partial fulfilment of the requirements for the award of degree of Master of Computer Applications of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under the supervision of **Prof. Vaheetha Salam.** This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed as the basis for the award of any degree, diploma or similar title of any other University.

Place: Kollam

Date: 18/07/2022 RESHMA KRISHNAN

DEPARTMENT OF COMPUTER APPLICATION

THANGAL KUNJU MUSALIAR COLLEGE OF ENGINEERING

Certificate

This is to certify that, the project report entitled "**FIRE DETECTION IN VIDEO SURVEILLANCE APPLICATIONS**", submitted by **RESHMA KRISHNAN (TKM20MCA2030),** to the **APJ Abdul Kalam Technological University** in partial fulfilment of the requirements for the award of the Degree of **Master of Computer Applications,** is a bonafide record of the project work carried out by her under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Internal Supervisor Head of the Department External Examiner

ACKNOWLEDGEMENT

First and foremost, I would like to thank the almighty for giving me the wisdom and grace for making my project a memorable one. I thank him for steering me to the shore of fulfilment under his protective wings.

I am extremely grateful to ***Dr. Fousia M Shamsudeen***, Assistant Professor and Head of the Department, MCA, TKMCE, for her constant support and encouragement throughout the project work.

With a profound sense of gratitude, I would like to express my heartfelt thanks to my guide ***Prof. Vaheetha Salam***, Associate Professor, Department of Computer Application, TKMCE, for her expert guidance, co-operation, and immense encouragement. I also extend my thanks to the entire faculty and staff of the Department of Computer Application, TKMCE, who have encouraged me throughout my course of study.

I also express my thanks to my family and friends, for their support and encouragement in the successful completion of this project work.

RESHMA KRISHNAN

ABSTRACT

Fire is one among the most frequently occurring hazards and is the primary cause of disastrous individual injury and crushing property harm. Hence a framework for the early recognition of fire is important to keep fires from fanning out of control. Fire alarms are often employed in ordinary structures using sensors based upon physical signals, such as infrared flame detectors that release heat, smoke sensors and ultraviolet flame detectors, and so on. However, these necessitate significant human involvement, such as going to the scene of fire to verify there is a fire on receiving fire alarms. Hence, a two-stage fire detection technique utilizing Convolutional Neural Networks and Recurrent Neural Networks has been proposed. In the first stage, a pre-trained CNN model is used to extract the flame characteristics. In the second stage, feature vector from CNN reaches the RNN which then predicts the probability of the input being a fire or a non-fire event.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
INTRODUCTION

The size and complexity of structures have grown in tandem with the economy's fast growth, creating significant fire control issues. To minimize fire losses, it is essential to have instantaneous fire identification and alarm systems with elevated sensitivity and precision. Traditional fire detection methods, such as heat and smoke detectors, are ineffective for real-time detection in broad areas, intricate structures, or areas where there are many disturbances. Missed identifications, falsely generated alarms, delayed detection, and other issues often arise as a result of the limitations of the aforementioned detection technologies, turning it as even more challenging to accomplish early fire warnings. Security cameras may be utilized to monitor big, open spaces and are simpler to install than sensor-based systems. Considering how commonplace CCTV is now, using one to monitor fires might be a cost-effective and efficient option. As a result, video-based fire detection has gained popularity as a study area.

The video surveillance-based fire detection technique has a large number of benefits such as early identification of fire, hiked accuracy, flexibility in the installation, and the potential to instantly identify fires in real time. The suggested technique processes video sequences captured by a camera which are extracted into frames by algorithms to identify the presence of a fire or fire risk in them.

This procedure is broken up into three steps: pre-processing, feature extraction, and detection. Extraction of flame characteristics from video pictures is essential for fire identification. Some algorithms have the problem of needing professional feature selection expertise. Even though researchers conduct several investigations on the picture characteristics of smoke and flame, they only manage to identify basic image characteristics like color, edges, and simple textures. The algorithms that extract image features with low and medium complexities, however, find it challenging to identify a way to tell fire from fire-like events due to complex scenes and fire types as well as numerous interference events present in practical application, resulting in decreased precision and poor generalization skills. CNN-based image detection algorithms have the ability to automatically learn and successfully extract complicated visual information. The performance of these algorithms in areas like optical search, medical diagnosis, autonomous driving, etc. has raised significant concerns. To fulfill long-range dependencies while categorizing video sequences, it is crucial to have a neural network remember the state of prior frames while assessing the current frame. RNNs may thankfully help to address this issue. Recurrence relations are present in these neural networks, where each new output is dependent on a mixture of earlier ones.

1.1. Problem Statement

Human mistake or system failure are the major causes of fire catastrophes. Different approaches to fire detection have been developed, which includes:

- Conventional fire alerting systems
  - based on ion or optical sensors.
  - necessitate significant human involvement, such as going to the scene of the fire to verify there is a fire during a fire alarm.
- Visual sensors-based systems that may face several problems such as
  - the intricacy of the situations being monitored since there are people and things that seem like fire.
  - the poor visual quality, the lack of contrast, and the weak signal transmission.

The proposed project aims to develop an early fire detection framework to detect fire from CCTV surveillance videos using the combination of convolutional neural networks and recurrent neural networks.

1.2. Objectives

The main objectives behind the project are as follows:

- To detect fire using less complex, yet efficient network architectures.
- To accurately and efficiently detect fire while minimizing false alarms.
- To use a two-stage CNN-RNN model for processing fire videos.
- To perform a comparative study on two similar models.

Chapter 2
LITERATURE SURVEY

Literature survey is a thorough examination and analysis of the literature that relates to a certain subject when using a literature review, it is possible to identify connected research questions. Thereafter, it is possible to look for and analyze pertinent material in an effort to find answers to these research questions. The ability to re-analyze study results can lead to the development of fresh insights which is one benefit of literature reviews. A literature survey summarizes and explains the entire and up-to-date body of information on a certain subject that may be found in scholarly publications and journal articles. One may be required to write a literature survey as a stand-alone assignment for a course, or they may be required to write one as part of the introduction to or planning phase for a larger work, typically a thesis or research report. The type of survey that was carried out will determine the scholar's focus, the viewpoint, and the type of hypothesis or thesis argument. Reading the literature surveys published or the introductory sections of theses and dissertations in our own field might help us grasp the differences between these two types by studying the organization of their arguments and observing how they approach the topics.

2.1. Purpose of the Literature Review

1. It provides readers with quick access to information on a specific subject by choosing excellent articles or studies that are pertinent, significant, important, and valid and compiling them into one comprehensive report alone.
2. By requiring the researchers to describe, assess, and compare original research in that particular field, it gives researchers starting out in a new field a great place to start.
3. It makes sure that previous work is not duplicated by researchers.
4. It might provide hints regarding the way future researches should go or suggest topics to concentrate on.
5. It highlights the key findings.

6. It points up gaps, discrepancies, and inconsistencies in the literature.
7. It offers a helpful critique of the methods and strategies used by other researchers.

2.2. Related Works

The section mainly describes related study modes in the area of efficient fire detection. And some of them are listed below.

2.2.1. Multi-feature Fusion Method

In their study [1], Pasquale Foggia et al., suggested a technique that relies on shape variation, color, and motion analysis to identify fire components using complimentary information. Using the Background Subtraction Algorithm, the first step detects things moving inside the blobs. The fact that flames change form extremely fast led the model to assess the evolution of the blob shape cross-wise two subsequent frames in the second step, following the method provided in [10]. Finally, an innovative descriptor that used a bag-of-words strategy was applied to analyze the mobility of fire components. When using the bag-of-words approach to solve their issue, they had taken into consideration the following steps, including the derivation of lower level representation, the dictionary definition that regulates the construction of higher-level representation, and the paradigm that was used for classification.

Only once motion detection is complete is color detection relevant for areas of motion. Precision has increased and superfluous calculations have been minimized by the. The suggested technique is shown to have greater improvements based on the spatial correlation between subsequent frames of pictures. A space-time stability-based flame centroid detection approach combines the flame properties with temporal information. The final choice is made by maximizing the Multi-Expert Evaluation's overall reliability in classifying the frame. The primary premise upon which they built the classifier was that the produced feature vector differed for the two classes, namely, ‘Fire’ and ‘NonFire’ classes, based on the evidence. When there is fire present, the movement is warped, which enabled them to identify the rather evenly spaced instances of the words.

2.2.2. MobileNet-based Fire detection

The examination of the literature reveals that CNNs have attained cutting-edge performance for a variety of difficult real-world issues, including image classification, object recognition, segmentation, localization, etc. Khan Muhammad et al., developed a model with a MobileNet-like architecture and applied it in their suggested approach [2] to detect fire under ambiguous surveillance environment settings.

The primary block of the proposed model's MobileNet (V2) architecture was given an extension layer. This layer enlarged the extend of channels along the input data prior to passing it to the depth-wise convolution layer, which filtered the input while a projection layer was employed to reduce the number of channels. After each layer, batch normalization using a ReLu activation function was used. Because the output data from the projection layer is modest in dimension and because an activation function might change important information, no activation function is applied after it. The proposed design consists of 17 blocks in total, which are succeeded by a 1 x 1 convolution and the classification conduit. The suggested architecture receives an input picture, and processes it to provide two probabilities for inference.

A limited dataset was utilized to evaluate such an innovative approach, which is not a benchmark, and apart from that, its accuracy is still poor despite all the improvements and developments that their work has made. Despite several adjustments, the model's false alert rate did not decrease.

2.2.3. GoogleNet-based Fire detection

Jamid Ahamad et al., [3] learned deep characteristics from several raw fire data using a model resembling the design of GoogleNet. The 100 layers in the suggested design include 2 primary convolutions, 4 maximum pooling, 1 average pooling, and 7 inception modules. The input picture was 224 x 224 x 3 pixels in size. S=2 was used to apply 64 kernels of size 7 x 7 to them, creating 64 feature maps (112 x 112). More activations were removed from those 64 feature maps using a max pooling layer with a kernel size of 3 3 (S=2). Again, using a convolution with kernel size 3 3 and S=1, 192 feature maps (56 56) were produced. Then, to separate rich variant features from less significant ones, a further max pooling layer with a kernel size of 3 3(S=2) was used. After then, a dimensionality reduction process was used, and just two inception layers were ultimately used.

On the basis of two datasets, the model's performance was assessed. The initial dataset that was compiled included 31 films shot in various situations. There were 14 movies of fire and 17 normal videos in the sample. The dataset is difficult since it only contains a little amount of training and validation data and is bigger than average because it contains high-quality video recordings. This makes it a premium alternative for experimentation. By recording films of items that resemble fires and mountains covered in smoke and clouds, we have created a dataset that is difficult for fire recognition techniques based on both color and motion. This was very important when comparing different models.

226 photos made up their second dataset, out of which 119 belonged to the 'Fire' class and 107 to the 'non-fire' class. Despite being modest, the dataset was effective enough since it included reddish and fire-seeming items, fire-like sunshine settings, and illumination in various structures. The outcomes were contrasted with five approaches, including both deep learning-based methods and methods based on manually created features. Even though their studies showed better flame detection accuracy, there were still a lot of false alarms, which has to be fixed.

2.2.4. Motion-Flicker-based Fire Detection

Because video fire scenarios are so complicated, artificially created static characteristics are very redundant. It is far more difficult to intelligently extract as many deep static characteristics as feasible. However, Yakun Xie et al., in their study [4] suggested a system that takes the flicker-motion based dynamic features as well deep static characteristics to increase overall efficiency and accuracy of video-based fire detection.

The [4] method has two stages: flicker detection and background removal. In the first phase, background subtraction is used to obtain motion characteristics. To perform video stream processing and background removal, they had employed the OpenCV library. Flicker detection is the next phase. Due to the fact that fire's process of combustion results in a continuous, disordered elevated-frequency time series of changes in comparison to objects that are ordinary, it is possible to conclude whether a potential RoI displays flicker features; if it does, the area in motion is thought to have fire's dynamic characteristics.

An ‘Adaptable Lightweight Convolutional Neural Network (AL-CNN)’ with three stages—network initialization, upturned residual-block stage, and spatial-pyramid pooling stage—was used to derive the deep static properties of fire.

The model's overall effectiveness is still debatable since it neglected to take into account the fact the static and dynamic characteristics of fire might help to improvise the accuracy of fire detection and decrease wrongly generated alarms.

2.2.5. Multi-Stage CNN-LSTM Fire Detection

In [5], Mahn Dung Nguyen introduced a multiple-staged fire detection approach that used CNNs and LSTMs in conjunction with video-based applications to detect fire. Initially, the colour, luminance shift, and brightness of the fire candidates were used to identify them. After that, the two-dimensional characteristics of flames were extracted using a pretrained CNN model. The pre-trained CNN's feature vectors served as the input for the LSTM network. A softmax classifier was used in the final step to recognize if the flames indicate a real fire or a non-fire event.

A fire colour model that incorporates HSV data was used to separate fire pixels, as presented in [6]. Mahn et al., made a modest modification to this model, in which an image pixel may be labelled as a fire pixel provided it satisfies certain HSV algorithm requirements. They had discovered that the colour of the image's pixels alone is insufficient to identify fire, and that other factors must be taken into account in a view to get rid of stationary backdrop images' fire-colored pixels. As a result, the flicker properties of flames were exploited to enhance fire pixel categorization in addition to colour. They had included [7]'s suggestion to employ the frequency of brightness flashing into their model. It determines the luminance's cumulative time derivative. The greatest values for the cumulative time derivative of brightness are seen in areas where there is a propensity for fire to flicker regularly.

The contenders for the fire pixel were then forwarded for feature extraction. ResNet-18 architecture, which was employed in [8], was the network architecture used for feature extraction. There are five convolutional layers in the design. The final global pooling layer uses activation functions to extract feature representational vectors from the training and test pictures. A total of 256 features were created by the global pooling layer. As a result, following feature extraction, the picture sequences from each fire candidate are sent into a multilayer BiLSTM as feature vectors [9]. Two BiLSTM layers were included in the suggested classifier in the model, one of which moved in a forward motion, the other in a backward motion. In order to calculate the output from both units, the hidden states of both layers were merged. The final classification was made using a softmax classifier in the LSTM's final state.

The network design proved very difficult to construct, despite the fact that the suggested method produced superior results in terms of identifying tiny fires and reducing false detection of moving objects that weren't really flames. The total accuracy of the model was still only around 93% despite being quite complicated, which is another flaw in the suggested approach.

The merits and demerits of these related works are summarized in Table 2.1.

SL. No.:	***Title of the paper***	***Pros***	***Cons***
1.	“Real-Time Fire Detection for Video-Surveillance Applications Using a Combination of Experts Based on Color, Shape, and Motion”	Significant increase in computational speed	Sensitivity to fire-colored and blinking objects, and also to some light sources, which frequently generate false positives
2.	“Efficient Fire Detection for Uncertain Surveillance Environment”	Light weight, computationally inexpensive	False alarms were still high
3.	“Convolutional Neural Networks Based Fire Detection in Surveillance Videos”	Balances out the computational complexity and accuracy of fire detection	False alarm rates continue to remain high.
4.	“Efficient Video Fire Detection Exploiting Motion-Flicker-Based Dynamic Features and Deep Static Features”	Enough portability to be deployed on embedded systems	Sensitivity to fire-colored and blinking objects
5.	“Multistage Real-Time Fire Detection Using Convolutional Neural Networks and Long Short-Term Memory Networks”	Threshold-free, appropriate for use in a variety of weather situations	Fire detection with various hues is unsuccessful (blue fire, white fire)

Table 2.1: Related works

Chapter 3

METHODOLOGY

An organized series of frames makes up a video. The order of the frames includes the temporal information, and then each frame consist of spatial information. To model both of these facets (for temporal processing), a hybrid method that combines recurrent layers and convolutionals (for spatial processing). Traditional computer vision-based fire detection techniques frequently use static characteristics or the short-term temporal variability, such as motions of flames and colors. However, because fires exhibit variable temporal appearance, the detection effectiveness of such methods that rely on static and short-term temporal behaviors is only partially accurate.

The proposed model is broken up into three parts

1. Data Collection and Pre-processing;
2. Building fire feature extraction model by Transfer Learning;
3. Building fire classification model.

In this study, the feature extractor used was the pre-trained InceptionV3 model and the classifiers that were used for performing the detection were GRU and LSTM. The results of both the classifiers were employed to identify the top performing model and for the detection of fire from the dataset.

3.1. Algorithm

Fire detection was carried out through the following steps:

1. Load datasets and fetch class labels ‘Fire’ and ‘Non-Fire’.
2. Capture the frames of videos.

3. Until a maximum frame count is achieved, extract frames from the videos.
4. Define size for the frames to be resized.
5. Normalize the count of frame sequences.
6. Employ a pre-trained network to extract relevant frame characteristics thus extracted.
7. Building and training the classifier models.
8. Predict the probabilities of each class.
9. Save the models for future use.

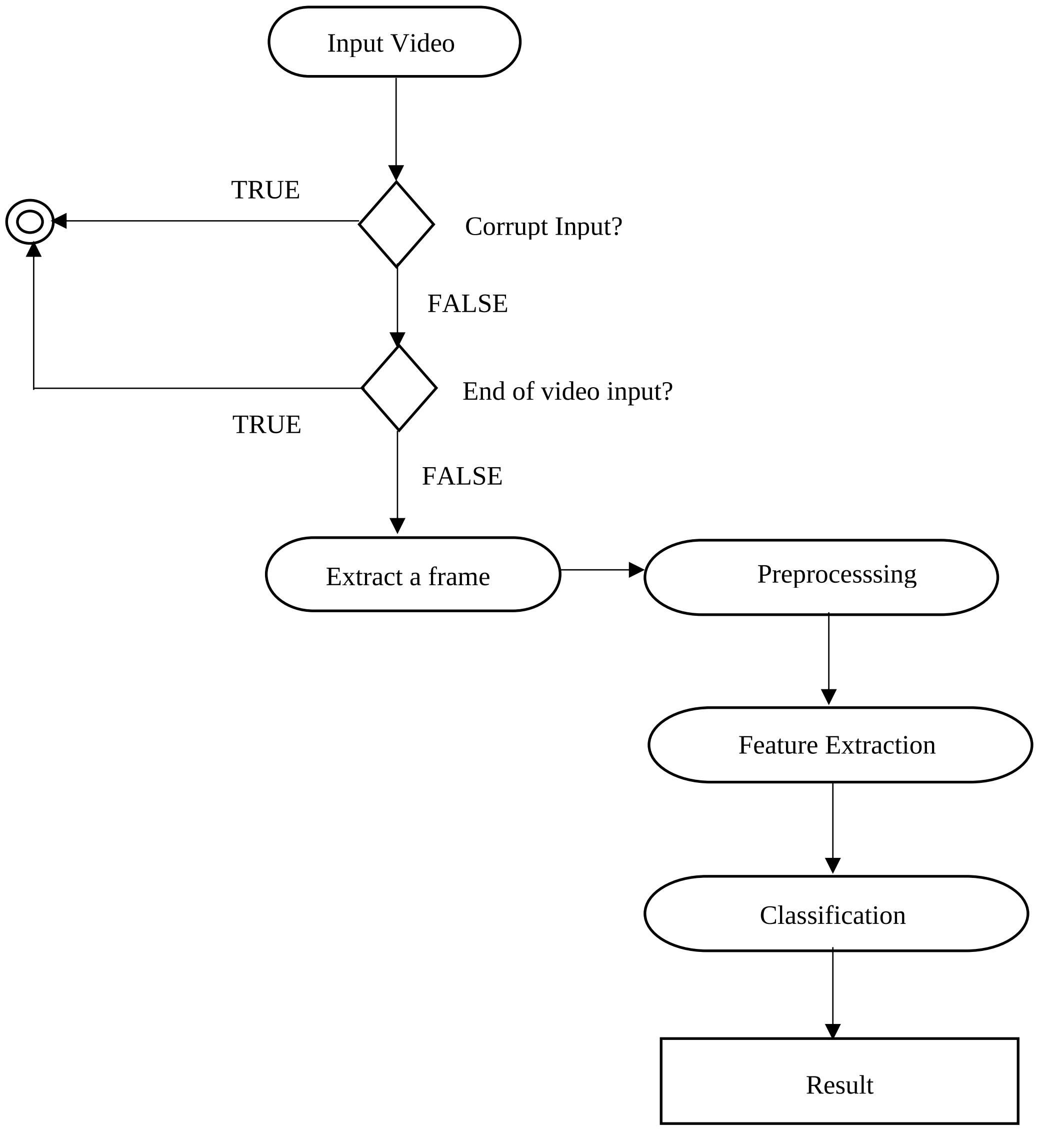


Figure 3.1: Dataflow diagram of the proposed model

3.2. System Architecture

In order to abstract the interactions, limitations and constraints between components as well as the overall build design of the software system, system architectural design is employed. It is a crucial tool because it gives a comprehensive picture of how the entire software system has been deployed physically. This study uses a pre-trained InceptionV3 model for feature extraction and two well-known recurrent neural network models GRU and LSTM for detection of Fire as shown in the architectural block diagram in Figure 3.2.

The data flow model of the suggested model is shown in Figure 3.1.

The model is broken into two parts:

1. Data Collection and Pre-processing;
2. Developing fire detection model by Transfer Learning.

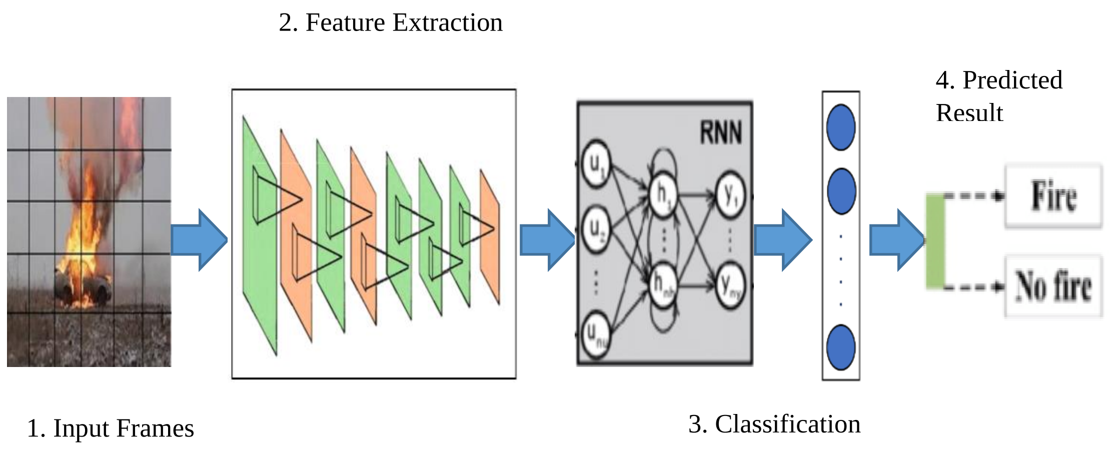


Figure 3.2: Architectural block diagram of the proposed model

3.2.1. Dataset

Three sets of data have been used, all of which were from Kaggle.

1. Fire-Detection dataset

   Two labels:

   -Label 1 → Fire (541 files)

   -Label 0 → Normal (110 files)

2. Fire_dataset

   - contains two folders/classes

   - 755 fire images, 244 non-fire images

3. FIRESENSE videos for flame detection

   - contains 11 positive and 16 negative videos

The files from all the three datasets were clubbed to one having two classes - fire and non-fire. Positive samples contain images or videos with real fire. Positive samples have real-fire pictures or videos. False Positives are pictures or movies with what seem to be flames, but aren't genuine fire incidents. False positive data are simpler to get. Therefore, gathering a variety of video frames is necessary to improve the fire detection procedure. Training and testing image/video frames are separated from the obtained dataset. The combined dataset currently has 1870 training samples and 233 test samples which are out-sourced from google since there is no dedicated dataset available online.

3.2.2. Data Preprocessing

Data preprocessing is a crucial phase in the machine learning process since the quality of the data and the information that can be extracted from it directly influence how well the model can learn. As a result, it is crucial to finish preprocessing the data before feeding it to the model.

Preparing the primary data for a deep learning model is a technique known as data preparation. It is the first and most important stage in developing a DL model. The format of the data in Deep Learning projects must be correct in order to get better results from the applied model. The dataset has been pre-processed to deal with a variety of conditions, including attributes with missing data, attributes with no values, attributes with the value of NA, and attributes with other conditions. Misleading information, such as having a different datatype or having a feature's format that is different from what is necessary, has also been taken into account.

Here, the folder has various durations of films and photographs. A video may simply be separated into individual frames and placed in a 3D tensor since it is an ordered series of frames. However, the frame count may vary from video to video, making it impossible to bulk stack them. The video frames may also be stored until a maximum frame count is achieved at regular intervals.

The following steps are adopted:

- ✓ Capture the video sequences.
- ✓ Keep extracting frames from them until a preset frame count is attained.
- ✓ In the situation, when fewer frame counts are generated than the preset frame count, it can be padded with zeros.
- ✓ Also define some variables such as the height and width of each frame that is extracted from the video. As neural network needs images to be of same size. Resized the images to a size of 224 x 224 px. Therefore, Added the input tensor of shape (224, 224, 3) to the pretrained model, namely, InceptionV3, where ‘3’ is the number of channels.

3.2.3. Building the models

Neural networks are employed to develop models for DL. A neural net receives inputs, which are subsequently processed using the weights that have been modified while being trained in hidden layers. The model then formulates a forecast. The weights are then changed to identify trends in order to improve predictions. The neural net then learns on its own, thus the user is not required to indicate the patterns to search for.

3.2.3.1. Building and training the Feature Extractor model

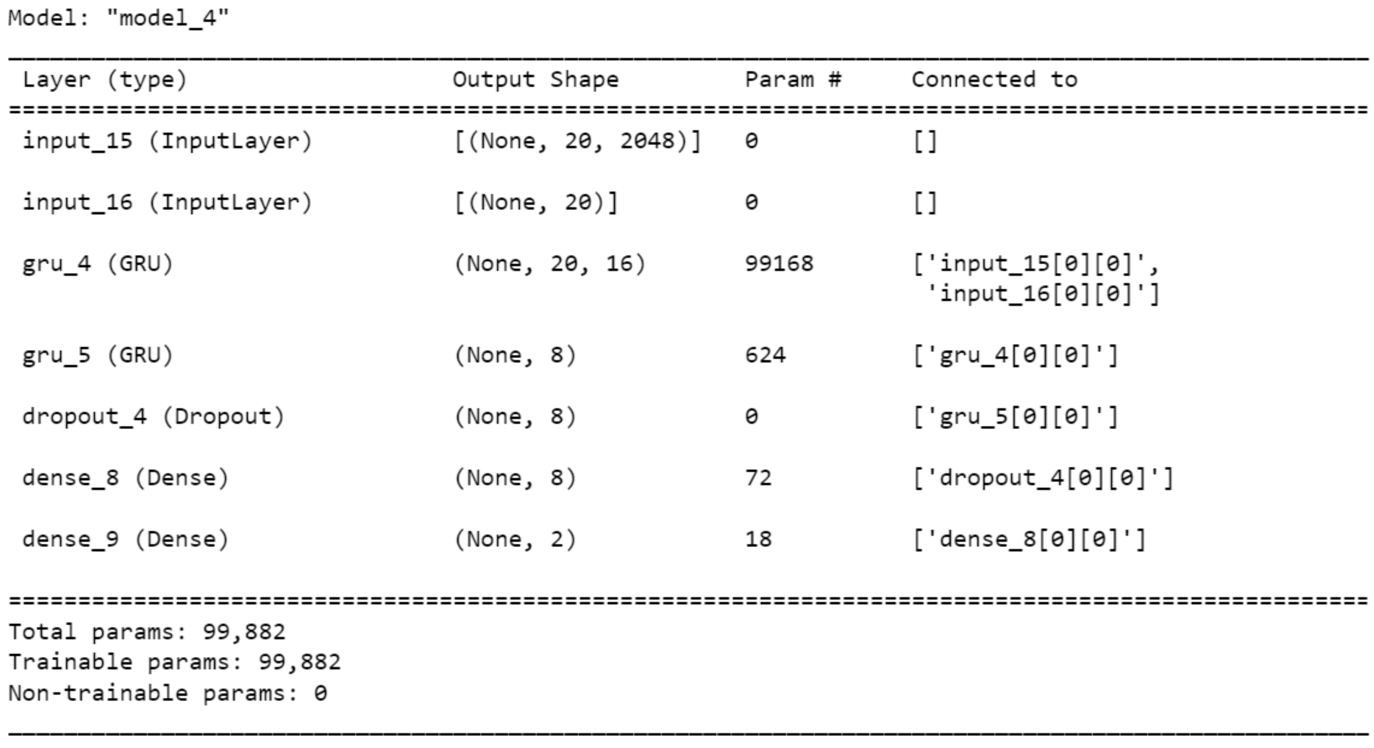
Here, the features from the video frames are extracted using only a Keras pre-trained model. The pre-trained models were generated considering the classification challenges for extremely large numbers of video frames. Fully connected layers (FCCs) serve as classifiers, whereas convolutional layers serve as feature extractors. These models often learn extremely excellent, discriminant information since they are quite massive and have seen a ton of pictures. To extract features from the video frames, the final layer, or FCC, is eliminated. Feature vectors are created in this stage. The feature vector sizes range from model to model. The main idea behind transfer learning is to use the knowledge gained from a more complicated but effective pre-trained DNN model to our simpler challenge. These deep neural net designs are built on ImageNet, and their pre-learned weights are utilized to learn features from the current dataset., instead of generating and training deep neural network models from scratch.

InceptionV3 is the pre-trained model used in the proposed method to extract significant characteristics from the extracted frames. The model is initialized with the weights of ImageNet. The output layer that performs the classification of data is removed as a RNN model is used as classifier in the proposed model.

The model was then trained using all the necessary parameters, for 100 epochs with a batch size of 17 frames and the model was saved as H5 file.

3.2.3.2. Building and training the Classifier model

In this study, the classifiers that were used for performing the detection of fire were GRU and LSTM. The results of both the classifiers were used to identify the top performing model and for the detection of fire from the dataset as well as for real time predictions.

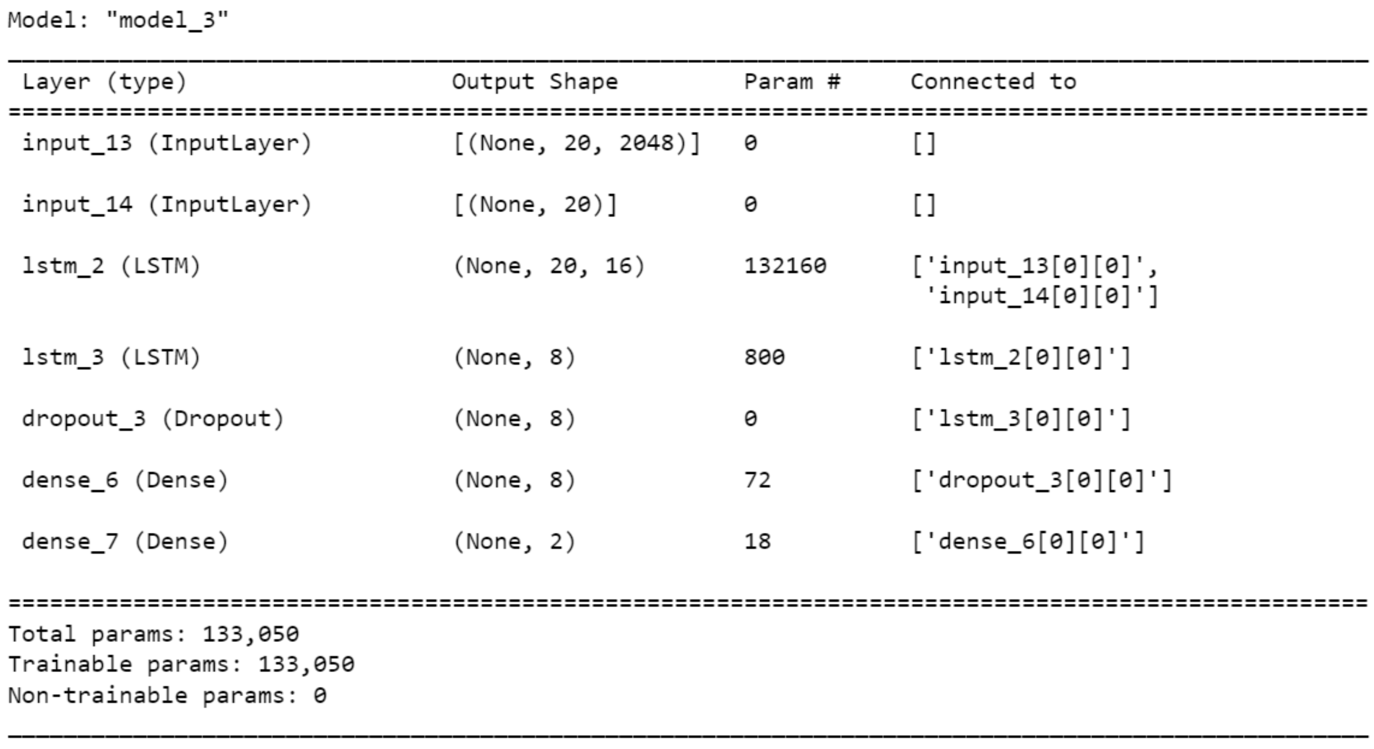


```
Model: "model_4"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_15 (InputLayer)          [(None, 20, 2048)]   0           []

 input_16 (InputLayer)          [(None, 20)]         0           []

 gru_4 (GRU)                    (None, 20, 16)       99168       ['input_15[0][0]',
                                                                  'input_16[0][0]']

 gru_5 (GRU)                    (None, 8)            624         ['gru_4[0][0]']

 dropout_4 (Dropout)            (None, 8)            0           ['gru_5[0][0]']

 dense_8 (Dense)                (None, 8)            72          ['dropout_4[0][0]']

 dense_9 (Dense)                (None, 2)            18          ['dense_8[0][0]']

==================================================================================================
Total params: 99,882
Trainable params: 99,882
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 3.3: GRU Model Summary

The necessary modules are first imported from Keras. For building the classifier model, first added layer with 16 internal nodes with return_sequences=True so that the second layer has a three-dimensional sequence input. Into the second layer, 8 internal units are added. A Dropout layer is added to the model to randomly set input units to 0 with a 40 % frequency of rate at each step during training time, thereby preventing overfitting. Then a dense layer is added with the activation ‘relu’. The output layer as well is added with dense units with an activation of ‘softmax’. Finally, compiled the model using ‘adam’ and ‘sparse_categorical_crossentropy’ as its optimizer and loss function respectively.

Next, training of the models was performed, with all the required parameters. Both GRU and LSTM sequential models were trained for 30 epochs and these models were saved as H5 files.



```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_13 (InputLayer)          [(None, 20, 2048)]   0           []

 input_14 (InputLayer)          [(None, 20)]         0           []

 lstm_2 (LSTM)                  (None, 20, 16)       132160      ['input_13[0][0]',
                                                                  'input_14[0][0]']

 lstm_3 (LSTM)                  (None, 8)            800         ['lstm_2[0][0]']

 dropout_3 (Dropout)            (None, 8)            0           ['lstm_3[0][0]']

 dense_6 (Dense)                (None, 8)            72          ['dropout_3[0][0]']

 dense_7 (Dense)                (None, 2)            18          ['dense_6[0][0]']

==================================================================================================
Total params: 133,050
Trainable params: 133,050
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 3.4: LSTM Model Summary

3.2.4. Testing the models

The proposed GRU and LSTM models were tested with testing data images and videos for performing the detection of fire as well as predicting fire on real-time images/video data. Testing accuracy is among the most relevant performance metrics of classification algorithm. The testing accuracy of the proposed LSTM model has outperformed the GRU model.

3.3. Software Requirements and Specifications

The software requirements for the project includes:

- ✓ Python
- ✓ Anaconda
- ✓ Jupiter Notebook

3.3.1. PYTHON

Python being an object-oriented programming language, is ideally modelled for fast proto- typing of complicated applications. It has interfaces to several OS system calls and libraries and is protractile to C or C++. The Python programming language is utilized by many large companies, including NASA, Google, YouTube, BitTorrent, etc. Python programming is extensively used in artificial intelligence, natural language processing, neural networks, and other cutting-edge computer science disciplines. Python is a potent language that can be used to create GUIs, create online applications, and create games. Python reading and writing are quite different from reading and writing standard English statements. Python programs must first be processed by machines since they are not written in a language that is machine readable. This indicates that each time a program is executed, its interpreter reads the program's code and translates it into byte code that can be read by a computer. The quality of Python is excellent throughout. In Python, all classes, data types, functions, and methods are treated equally. Programming languages are developed to meet the needs of users and programmers for an effective tool to construct program that have an influence on people's lives, way of life, economy, and society. By boosting productivity, improving communication, and boosting power, they help improve life. Here, python version 3.8.5 is used.

3.3.2. ANACONDA

For scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), Anaconda is a free and open-source version of the Python and R programming languages that attempts to streamline package management and deployment. Conda, a package management system, controls package versioning. For Windows, Linux, and MacOS, the Anaconda distribution provides data-science packages. The Anaconda distribution includes the Virtual Environment Manager and Conda package management in addition to more than 1,500 packages. Additionally, Anaconda Navigator, a graphical user interface, is provided as a substitute to CLI, the command line interface. The management of package dependencies is a big difficulty for Python data science, and the main distinction between conda and the pip package manager. For this reason, conda was created.

3.3.3. JUPYTER NOTEBOOK APP

The Jupyter Notebook App is a server-client program that enables the web browser-based editing and execution of notebook papers. The Jupyter Notebook App may be used locally, without an internet connection, or it can be deployed on a remote server and viewed online.

When a Jupyter Notebook App is opened, the component that is shown immediately is the Notebook Dashboard. The Notebook Dashboard is primarily used to manage the running kernels and open notebook papers (visualize and shutdown)

The "computational engine" that runs the code in a Notebook document is called a notebook kernel. Python code is run via the ipython kernel. The corresponding kernel is started instantly when a Notebook document is opened. The kernel does the calculation and generates the results when the notebook is run (either cell-by-cell or through the menu Cell -> Run All). The kernel may use a lot of CPU and RAM, depending on the kind of calculations.

3.3.4. Transfer Learning Models

Reusing old models to address fresh-new problems or challenges is referred to as transfer learning in machine learning. Transfer learning is generally used:

- To avoid having to train several machine learning models from start to fulfil identical tasks, saving time and resources.

- ➢ As a means of increasing efficiency in machine learning tasks that demand a lot of resources, including image classification or natural language processing.
- ➢ To use pre-trained models to compensate for an organization's lack of labelled training data.

Transfer learning is the process of utilizing the pertinent components of a machine learning model that has already been trained to solve a different but related issue. The model will often need this information as its foundation, with other components added as needed to address certain problems.

3.3.4.1. INCEPTIONV3

InceptionV3 is a Convolutional Neural Network which is 48 layers deep. It is possible to load a network that has already been trained using more than a million photos from the ImageNet database. The pretrained network can categorize photos into 1000 different item categories, including several animals, a keyboard, a mouse, and a pencil. The network has thus acquired rich feature representations for a variety of pictures. The Figure 3.5 shows the network architecture of InceptionV3 used in the proposed model.

3.3.5. Recurrent Neural Network Models

A family of neural networks known as recurrent neural networks (RNN) is effective in modelling sequential information, comprising time series (data that is collected at regular periods of time) and natural language. A RNN layer iterates through a sequence's timesteps using a for loop as shown in the diagram, while retaining an internal state that contains data on the timesteps it has already observed. The Keras RNN API is designed with a focus on:

- ✓ Ease of use: the built-in keras.layers.RNN, keras.layers.LSTM, keras.layers.GRU layers make it possible to swiftly construct recurrent models without the need to make challenging configuration decisions.
- ✓ Ease of customization: The innermost portion of the for loop, the RNN cell layer, may have specific behaviour specified and utilized with the standard keras.layers.RNN layer (the for

loop itself). This enables rapid, flexible prototyping of many research concepts with little coding.

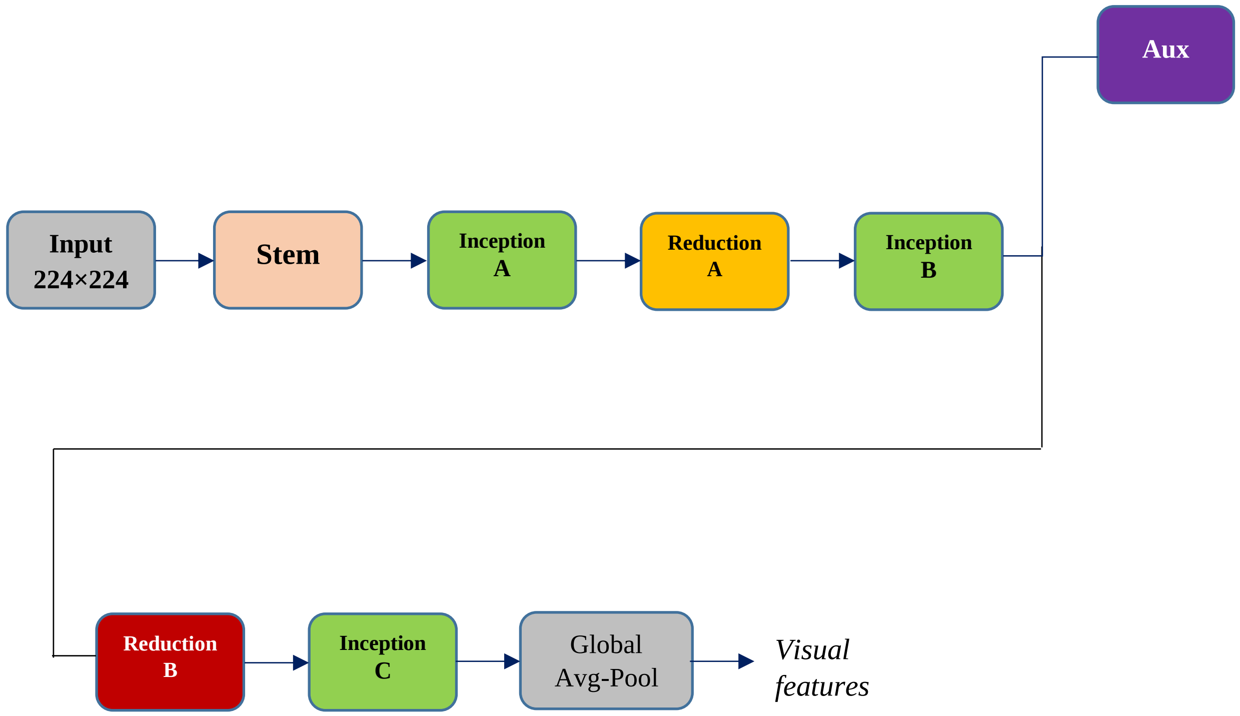


Figure 3.5: InceptionV3 Architecture

An organized series of frames makes up a video. The order of the frames carries the time information, while each frame contains space information. Recurrent Neural Networks (RNNs), one among the many Deep Learning models, are taken into consideration in this article because they may be used to simulate the long-range relationships between succeeding sequences of frames.

Video sequences have temporal dependence since it is unlikely that things would change quickly between timesteps. The Recurrent Neural Networks are thus the most appropriate for fire recognition from videos. RNNs often experience the well-known "Vanishing" and "Exploding" Gradient issues, which prevents them from learning long-range relationships.

Two variations of recurrent neural networks have been presented that employ a "gating" strategy to get around these issues in order to prevent them:

1. Long Short-Term Memory
2. Gated Recurrent Unit.

To model temporal aspects, recurrent layers have been used in the proposed architecture. Specifically, separate Recurrent Neural Networks (RNN) consisting of GRU layers as well as LSTM layers have been used.

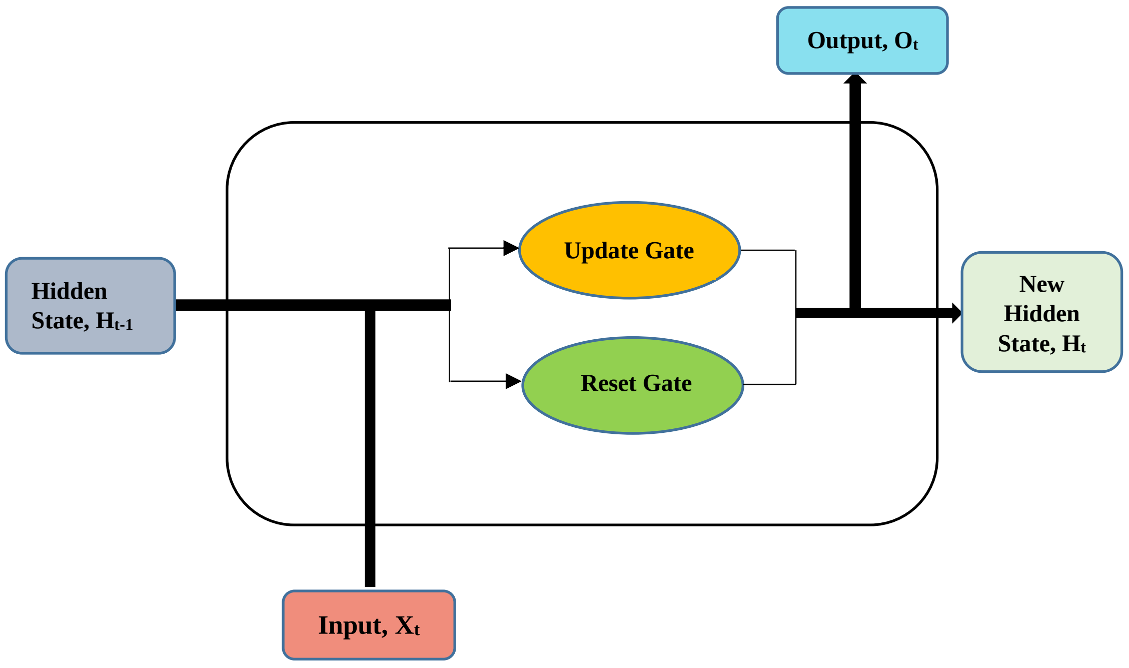


Figure 3.6: GRU Cell

3.3.5.1. LONG SHORT-TERM MEMORY NETWORK

A unique kind of recurrent neural networks called long short-term memory addresses the issue of disappearing or exploding gradients in RNNs. Because of this, LSTM can identify, analyze, and forecast time series when there are extremely large and ambiguous time gaps between significant events. An LSTM network is built of LSTM blocks, each of which consist of three gates; an input gate, an output gate, and a forget gate. These gates allow the LSTM cell to remember a value for any duration of time, forget the value when it is no longer relevant, or output it. The mistake that may backpropagate across time and layers is preserved with the aid of LSTMs. Figure 3.7 shows an LSTM network cell.

When transforming input, LSTMs memory cells assign addition and multiplication distinct functions. When backpropagated at depth, it must maintain a constant error and hence, they sum the two instead of multiplying the current state of the cell by the incoming input to determine the succeeding cell state.

3.3.5.2. GATED RECURRENT UNIT NETWORK

A significantly more condensed variant of the LSTM is the GRU. It combines the forget and input gates as a single “update gate” and in contrast with LSTMs, GRU has an additional “reset gate”. The end model is simpler than standard LSTM models and is becoming increasingly popular.

However, a Gated Recurrent Unit like the LSTM modulates data inside the unit without using a separate memory cell. Figure 3.7 shows a GRU cell unit. How often the unit changes its activation depends on the input gate. The device might disregard the past information while the reset gate is open. The level that the past state be considered now is controlled by the update gate. Reset gates will be activated on units having short-term dependencies. Engaged update gates are present in units with long-term dependencies.

While there are many analogies between GRU and LSTM, there are also significant distinctions between the two, which is summarized in Table 3.1.

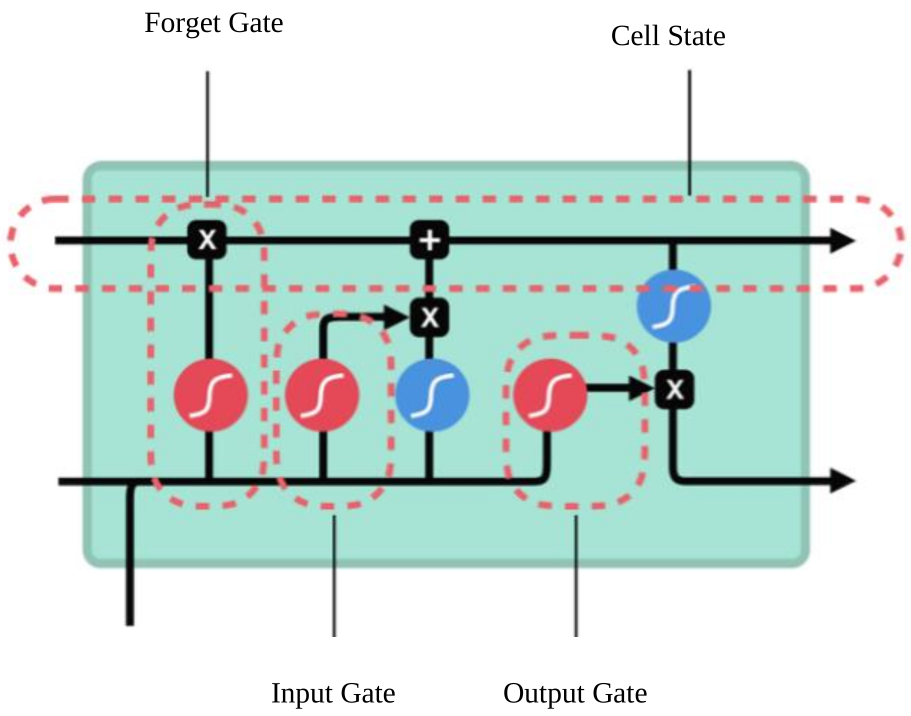


Figure 3.7: LSTM Network Architecture

3.3.6. Hardware and Experimentation Environment

The hardware used for this experiment includes Windows 11 Home 64-bit OS, x64-based processor, Intel(R) Core (TM) i7-8565U CPU @ 1.80GHz, 1992 Mhz, 4 Core(s), 8 Logical Processor(s), 16 GB RAM.

The experimental environment was prepared by using Python 3.10 programming language. Framework used is Keras with TensorFlow as background in the Anaconda environment. Machine learning and deep learning libraries include - NumPy, Pandas, Matplotlib, Scikit, Seaborn.

GRU	***LSTM***
Two gates : reset and update	Three gates : input, output, forget
Does not possess any internal memory	Does possess any internal memory
Comparitively lesser complexity	More complex
Works well with small datasets	Works well with large datasets

Table 3.1: GRU-LSTM Comparison

Chapter 4
RESULT AND DISCUSSION

The objective of this work was to develop a robust software system which can detect fire in videos and images and can work well in any environment. The combination of CNNs and specialized RNNs have been quite successful in video-based classification. In this regard, various deep learning models have been experimented with and a comparative study was performed using InceptionV3-GRU as well as InceptionV3-LSTM combinations to find out the one which offers the best performance metrics' values such as Accuracy, Precision and Recall. The values for these combinations have been shown in Figure 4.1.

During testing, the model functioned exceptionally well. In all of the test fire sequences, it was able to identify and estimate the likelihood of fires, but it also incorrectly identified several non-fire videos.

The suggested model is strong, dependable, inexpensive, and offers great performance in comparison to current hardware options. The model performs better than current software solutions that heavily rely on domain expertise and feature engineering because it uses transfer learning and deep learning approaches, which are simple to develop, update, and use less computational resources.

4.1. Training and Validation Results

The feature extractor model was trained for 100 epochs with a batch size of 17 for the training phase, allowing for the estimation of the error gradient using 17 samples from the training dataset before the model weights were updated. This results in 17 samples per epoch for training, and 17 samples per epoch for testing. As can be seen from Figure 4.1, training produced results that were optimized with high accuracy of 93.13 percent for InceptionV3-GRU and 95.71 percent for InceptionV3-LSTM. The experimental comparison of GRU and LSTM is shown in figure 4.1.

	GRU	***LSTM***
Cells	16	16
Epochs	50	50
Parameters	99,882	133,050
Training Samples	1870	1870
Test Accuracy	93.13%	95.71%

Table 4.1: Experimental Comparison between GRU and LSTM

GRU could achieve an accuracy of 93.13% with 99,882 parameters while LSTM could achieve about 95.71% with 133,050 parameters; however, the network architectural as well as computational complexity of LSTM is much higher than that of GRU.

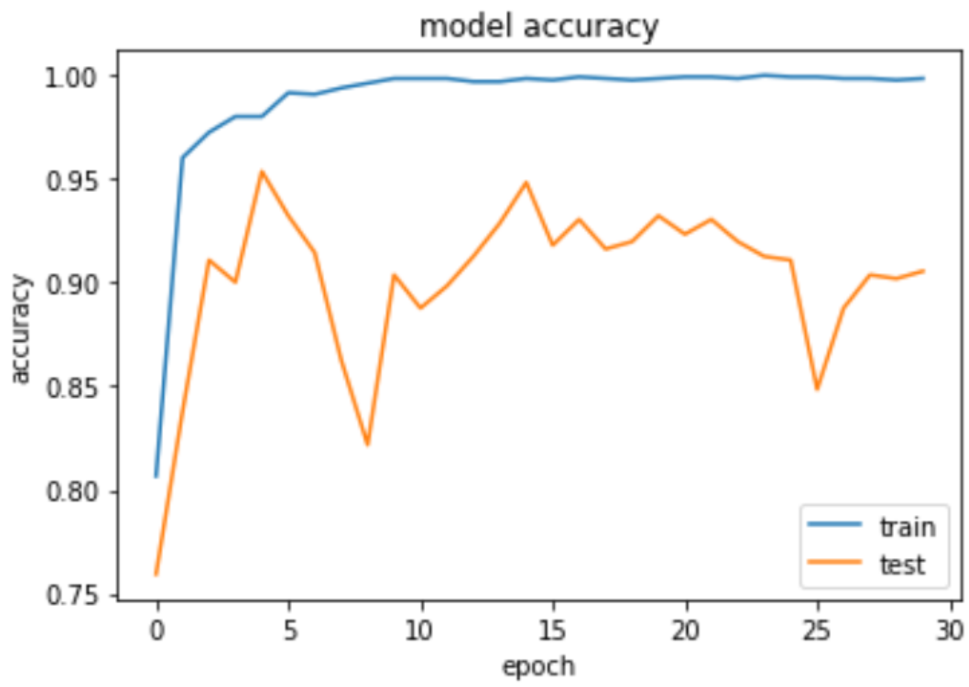


Figure 4.1 (a): Accuracy Curve for InceptionV3-GRU model

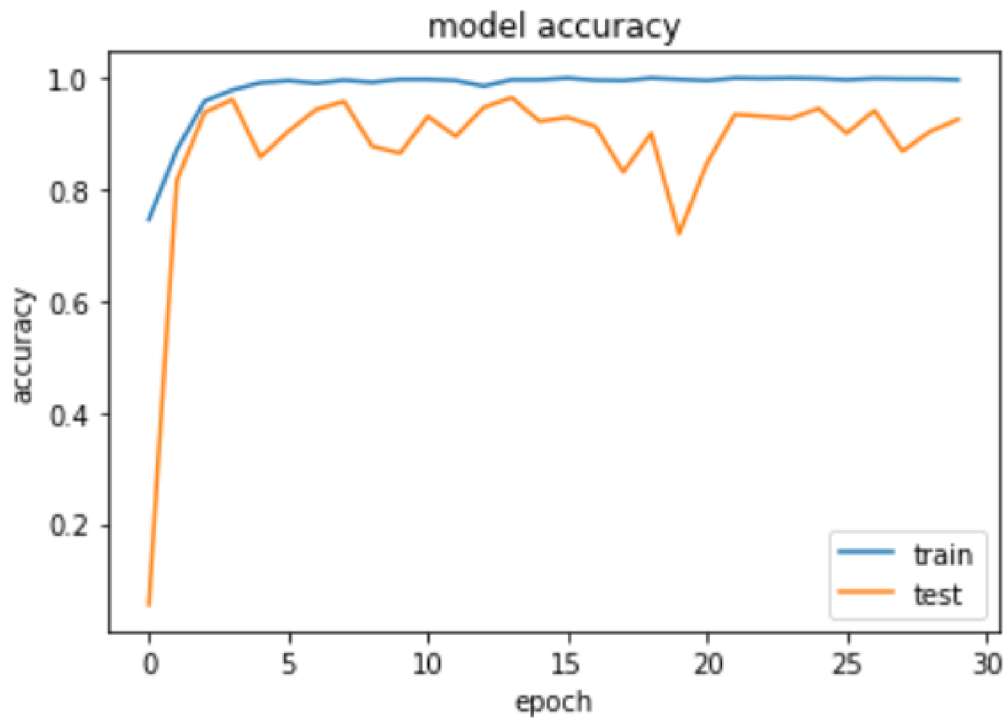


Figure 4.1 (b): Accuracy Curve for InceptionV3-LSTM model

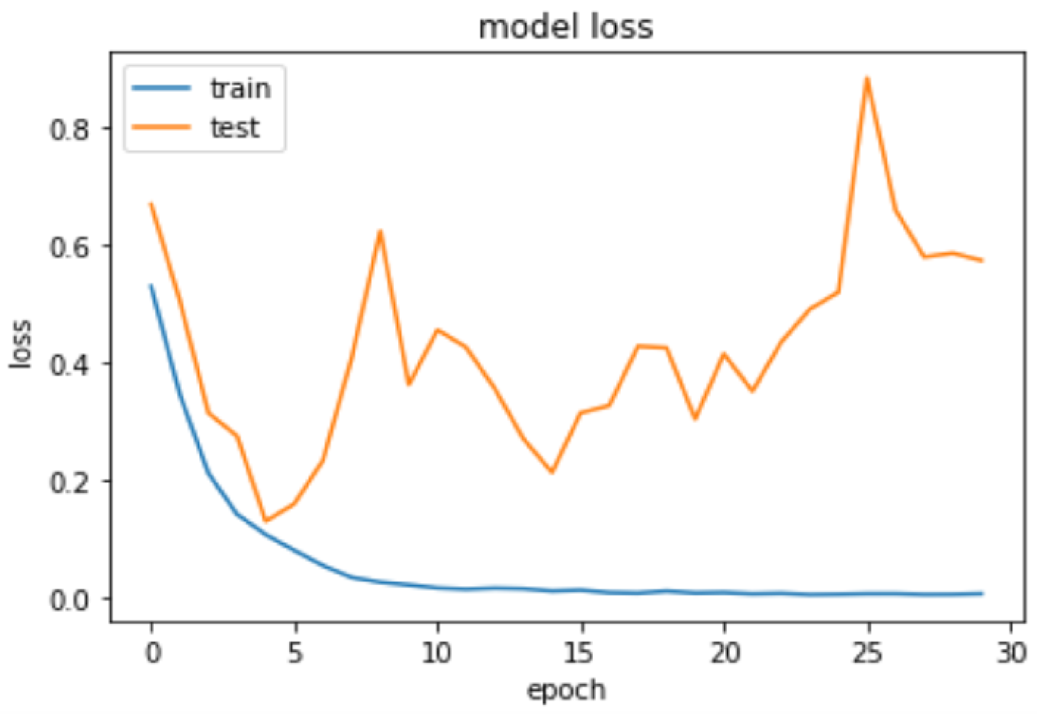


Figure 4.1 (c): Loss Curve for InceptionV3-GRU model

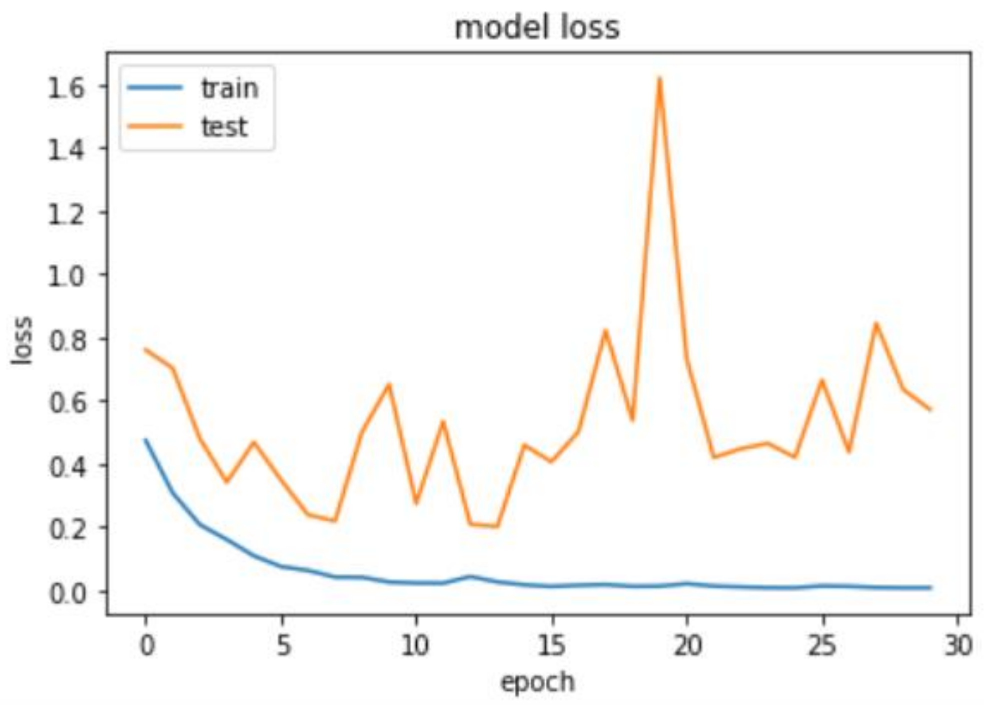


Figure 4.1 (d): Loss Curve for InceptionV3-LSTM model

4.2. Performance Metrics for Validation Phase

The performance parameters used to determine accuracy based on the classification results are shown in the table below. Performance analysis is done to identify the best model having the highest detection rate. The general evaluation metrics such as Accuracy, Precision, Recall, F1 score and confusion Matrix are used. High accuracy here indicates the enhanced detection rate and reduced false alarm rate. The performance indicators include: -

- ❖ True Positive (TP) is the number of correct classifications of attack category.
- ❖ True Negative (TN) is the number of correct classifications of normal category.
- ❖ False Positive (FP) is the number of incorrect classifications of attack category i.e., normal category wrongly classified as intrusive.
- ❖ False Negative (FN) is the number of incorrect classifications of normal category i.e., attack category wrongly classified as normal.

The most important performance matrix is precision. Precision is the weightage of relevant examples among the data returned by categorization methods. The suggested GRU's accuracy value outperforms that of the LSTM model.

Precision = T P/(T P+FP)

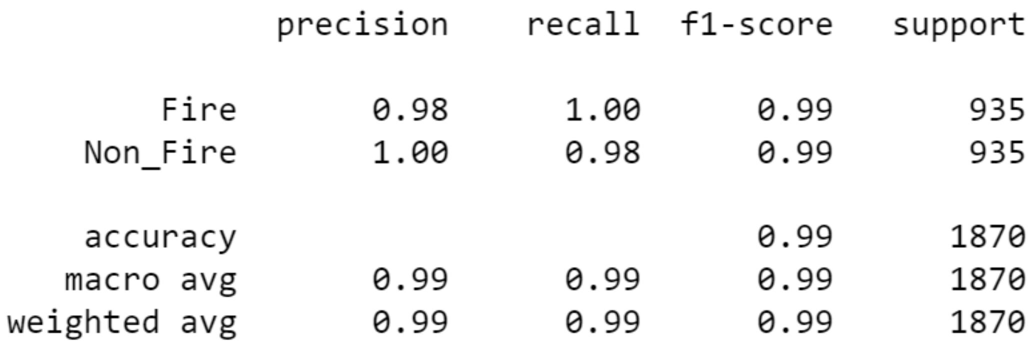
One of the most widely used evaluation metrics for gauging the efficacy of the categorization model is recall. The percentage of all relevant instances that were really the recovered instance by the classification method is known as recall.

Recall = T P/(T P+TN)

The f1 score is applied in statistical analysis of binary classification as a means of testing the accuracy levels. To calculate the f1 score, the classification algorithm's accuracy and recall are factored.

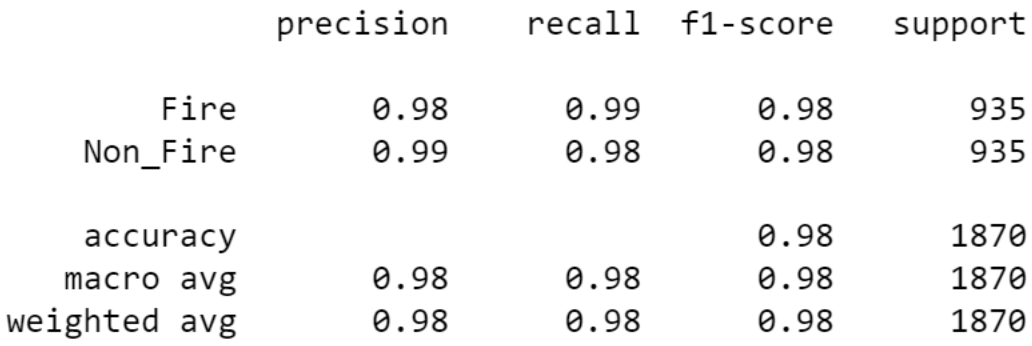
F1score = precision Recall/ (Precision + Recall)

The relevant performance metrics for the GRU and LSTM models are shown in tables 4.2 (a), (b).



```
              precision    recall  f1-score   support

        Fire       0.98      1.00      0.99       935
    Non_Fire       1.00      0.98      0.99       935

    accuracy                           0.99      1870
   macro avg       0.99      0.99      0.99      1870
weighted avg       0.99      0.99      0.99      1870
```

Figure 4.2(a): Performance Metrics-GRU



```
              precision    recall  f1-score   support

        Fire       0.98      0.99      0.98       935
    Non_Fire       0.99      0.98      0.98       935

    accuracy                           0.98      1870
   macro avg       0.98      0.98      0.98      1870
weighted avg       0.98      0.98      0.98      1870
```

Figure 4.2(b): Performance Metrics-LSTM

4.2.1. Confusion Matrix

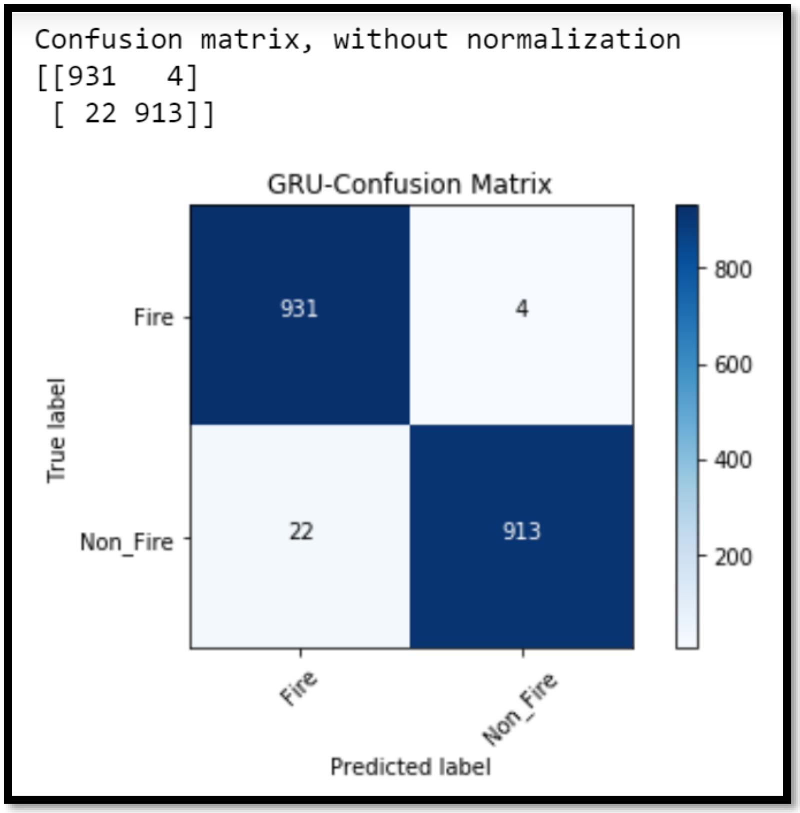
A confusion matrix is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. It allows easy identification of confusion between classes e.g.; one class is commonly mislabeled as the other.

Most performance measures are computed from the confusion matrix. The important terms included in confusion matrix are as following:

- True Positive (TP): Observed positive, and predicted as positive.
- False Negative (FN): Observed positive, but predicted as negative.
- True Negative (TN): Observed negative, and predicted as negative.
- False Positive (FP): Observed negative, but predicted as positive.

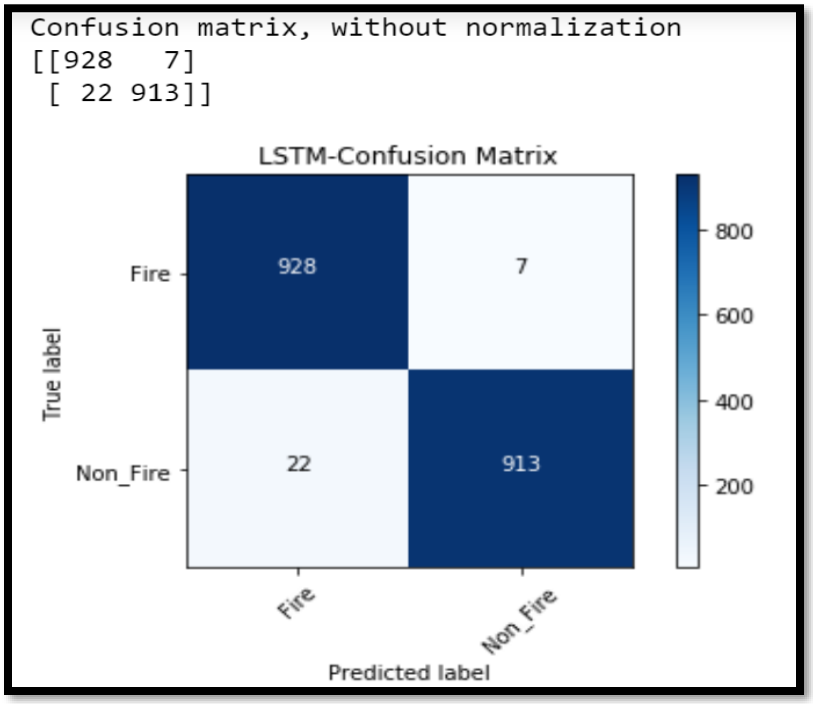
Though the LSTM model showed hiked accuracy levels than GRU model, the number of false positive and false negative predictions are comparatively lesser for GRU as compared with LSTM.

The confusion matrices of InceptionV3+GRU and InceptionV3+LSTM generated using Fire and Non-Fire samples from the dataset are shown in Figure 4.3 and Figure 4.4 respectively:



```
Confusion matrix, without normalization
[[931   4]
 [ 22 913]]
```

Figure 4.3: Confusion Matrix of GRU



```
Confusion matrix, without normalization
[[928   7]
 [ 22 913]]
```

Figure 4.4: Confusion Matrix of LST

Chapter 5
CONCLUSION

As a result, it can be said that the suggested model accurately categorizes photos and videos for fire detection. The CNN-RNN network is used with a fire detection system to evaluate fire in both the spatial and temporal domains. In order to provide unique outcomes for producing the probabilities of fire and non-fire data, the model's loss is reduced during training and the accuracy is concurrently increased during each epoch step. The pre-processing steps guarantee that convolutional neural networks and recurrent neural networks execute without being prone to overfitting, ensuring that the outputs are consistently coherent.

The primary focus of this study is an investigation of the two unique types of recurrent neural networks' efficiency. According to the experimental findings, LSTM performs more accurately on smaller datasets than GRU. GRU employs fewer training parameters than LSTM, which consumes less memory and runs quicker. However, LSTM performs better on bigger datasets.

5.1. Advantages of the proposed model

As the system is capable of processing image/video data from a video surveillance equipment by algorithms to determine the presence of a fire, the technique has certain advantages such as:

- ✓ early fire detection
- ✓ less complex network architecture
- ✓ high accuracy
- ✓ flexibility in system installation
- ✓ the capacity to quickly identify flames even in uncertain surveillance environmental conditions.

5.1. Future Enhancement

This project has an immense scope in the field of fire and safety and can be continued for other such insightful innovations. In the future works, a fire candidate segmentation model will be integrated to eliminate the instability of basic flame features. In order to improve the overall detection results, more video data will be collected and used.

REFERENCES

[1] Pasquale Foggia, Alessia Saggese, and Mario Vento "Real-Time Fire Detection for Video-Surveillance Applications Using a Combination of Experts Based on Color, Shape, and Motion", IEEE Transactions on Circuits and Systems for Video Technology (Volume: 25, Issue: 9, Sept. 2015 pp. 1545–1556.
[Online] Available: https://ieeexplore.ieee.org/document/7014233

[2] Khan Muhammad, Salman Khan, Mohamed Elhoseny , Syed Hassan Ahmed and Sung Wook Baik , "Efficient Fire Detection for Uncertain Surveillance Environment", IEEE Transactions on Industrial Informatics (Volume: 15, Issue: 5, May 2019).
[Online]. Available: https://ieeexplore.ieee.org/document/8635329

[3] Jamil Ahamad, Irfan Mehmood, Khan Muhammed, Sung Wook Baik, Seungmin Rho, "Convolutional Neural Networks Based Fire Detection in Surveillance Videos", IEEE Access (Vol:6, 06 March 2018), pp. 18174-18183.
[Online]. Available: https://ieeexplore.ieee.org/document/8307064

[4] Yakun Xie, Jun Zhu, Yungang Cao, Yunhao Zhang, Dejun Feng, Yuchun Zhang, Min Chen, "Efficient Video Fire Detection Exploiting Motion-Flicker-Based Dynamic Features and Deep Static Features," IEEE Access, Vol. 8, 29 April 2020, pp. 81904 – 81917
[Online]. Available: https://ieeexplore.ieee.org/document/9081979

[5] Manh Dung Nguyen, Hoai Nam Vu, Duc Cuong Pham, Bokgil Choi and Soonghwan Ro, "Multistage Real-Time Fire Detection Using Convolutional Neural Networks and Long Short-Term Memory Networks," IEEE Access (Volume: 9), 22 October 2021, pp. 146667 – 146679
[Online]. Available: https://ieeexplore.ieee.org/document/9584840

[6] P.-H. Huang, J.-Y. Su, Z.-M. Lu, and J.-S. Pan, "A fire-alarming method based on video processing," in Proc. Int. Conf. Intell. Inf. Hiding Multimedia, Dec. 2006, pp. 359–364.

[7] G. Marbach, M. Loepfe, and T. Brupbacher, "An image processing technique for fire detection in video images," Fire Saf. J., vol. 41, no. 4, pp. 285–289, Jun. 2006.

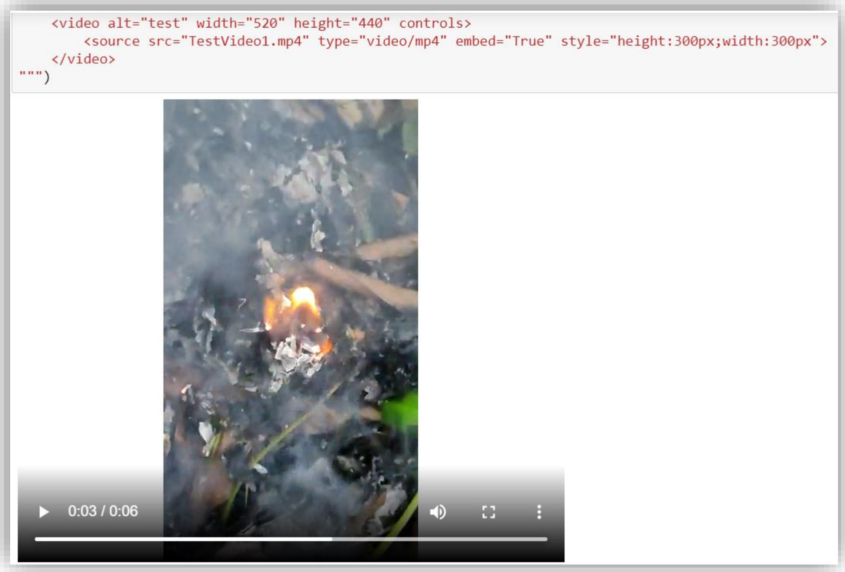
[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778

[9] J. Sharma, O.-C. Granmo, M. Goodwin, and J. T. Fidje, "Deep convolutional neural networks for fire detection in images," in Proc. Int. Conf. Eng. Appl. Neural Netw. Cham, Switzerland: Springer, 2017, pp. 183–193.

[10] A. Rafiee, R. Tavakoli, R. Dianat, S. Abbaspour, and M. Jamshidi, "Fire and smoke detection using wavelet analysis and disorder characteristics," in Proc. 3rd Int. Conf. Comput. Res. Develop. (ICCRD), vol. 3. Mar. 2011, pp. 262–265.

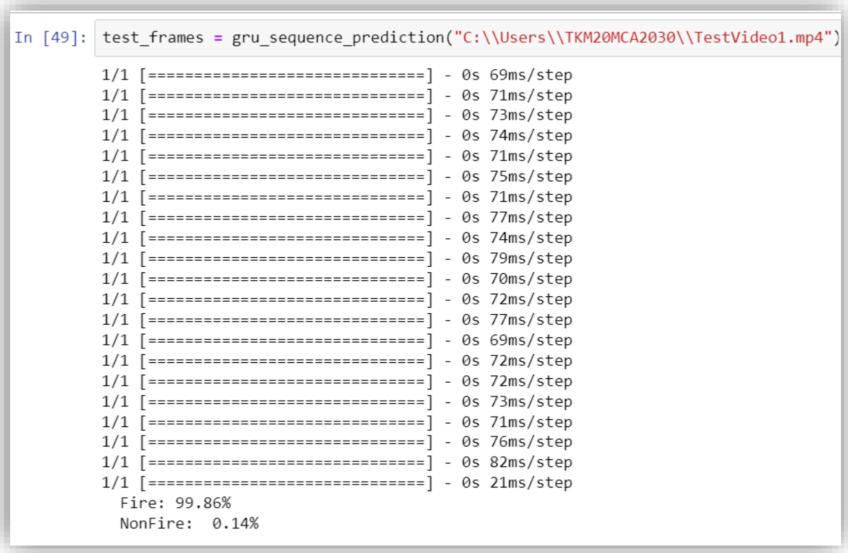
APPENDICES

Screenshots



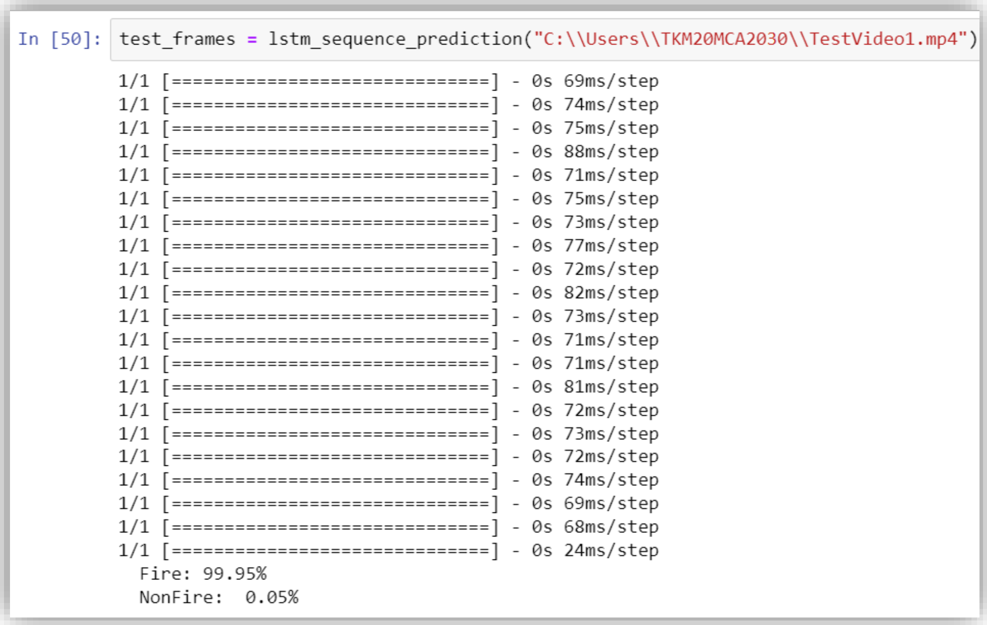
```
    <video alt="test" width="520" height="440" controls>
        <source src="TestVideo1.mp4" type="video/mp4" embed="True" style="height:300px;width:300px">
    </video>
""")
```

Figure A.1: Real-time Test_Video_1



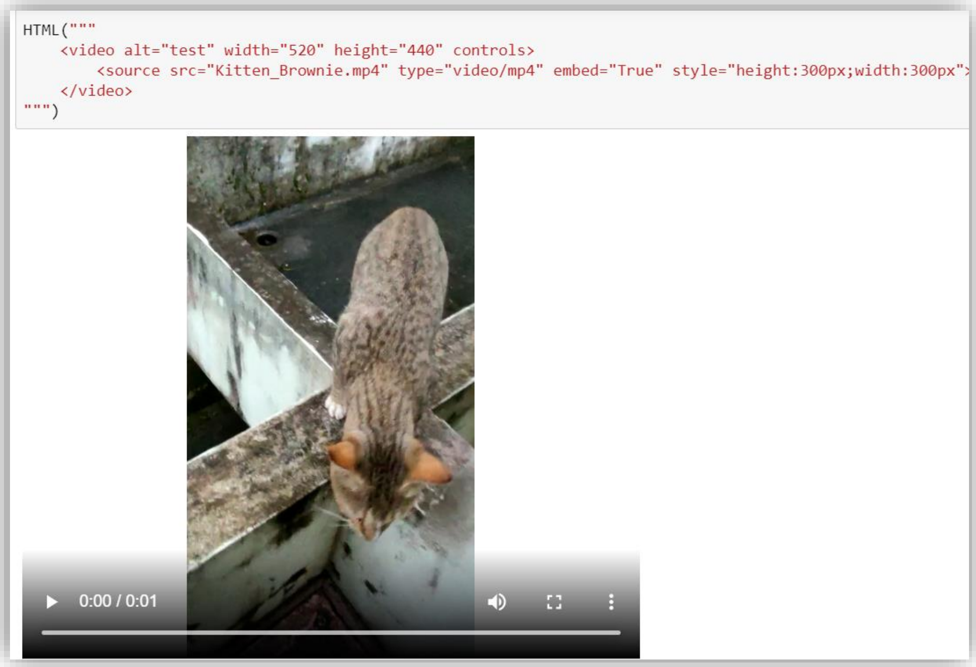
```
In [49]: test_frames = gru_sequence_prediction("C:\\Users\\TKM20MCA2030\\TestVideo1.mp4")

         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 79ms/step
         1/1 [==============================] - 0s 70ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 82ms/step
         1/1 [==============================] - 0s 21ms/step
           Fire: 99.86%
           NonFire:  0.14%
```

Figure A.2: Real-time Test_Video_1 GRU Prediction Result



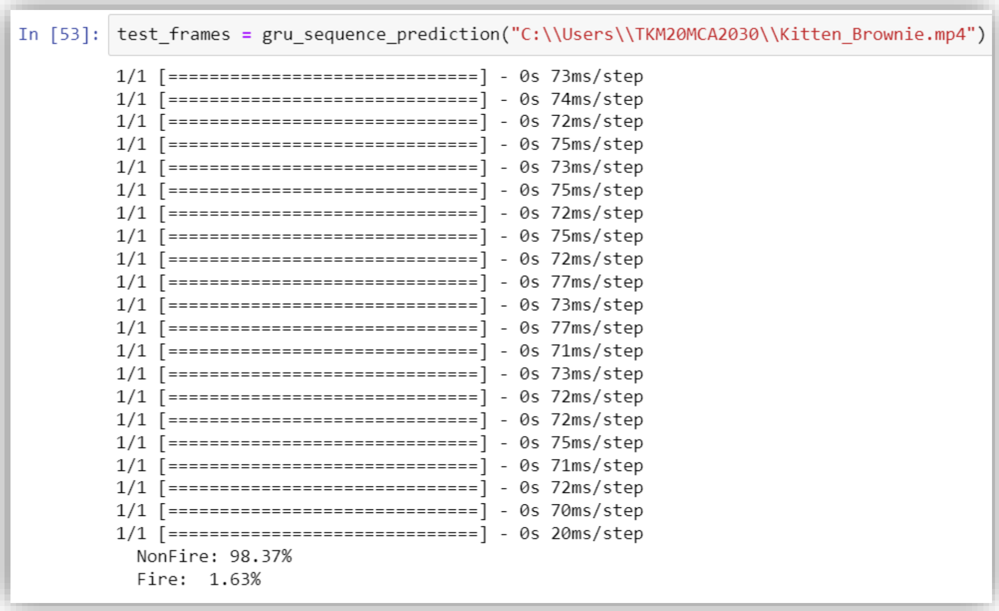
```
In [50]: test_frames = lstm_sequence_prediction("C:\\Users\\TKM20MCA2030\\TestVideo1.mp4")

         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 88ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 82ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 81ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 68ms/step
         1/1 [==============================] - 0s 24ms/step
           Fire: 99.95%
           NonFire:  0.05%
```

Figure A.3: Real-time Test_Video_1 LSTM Prediction Result



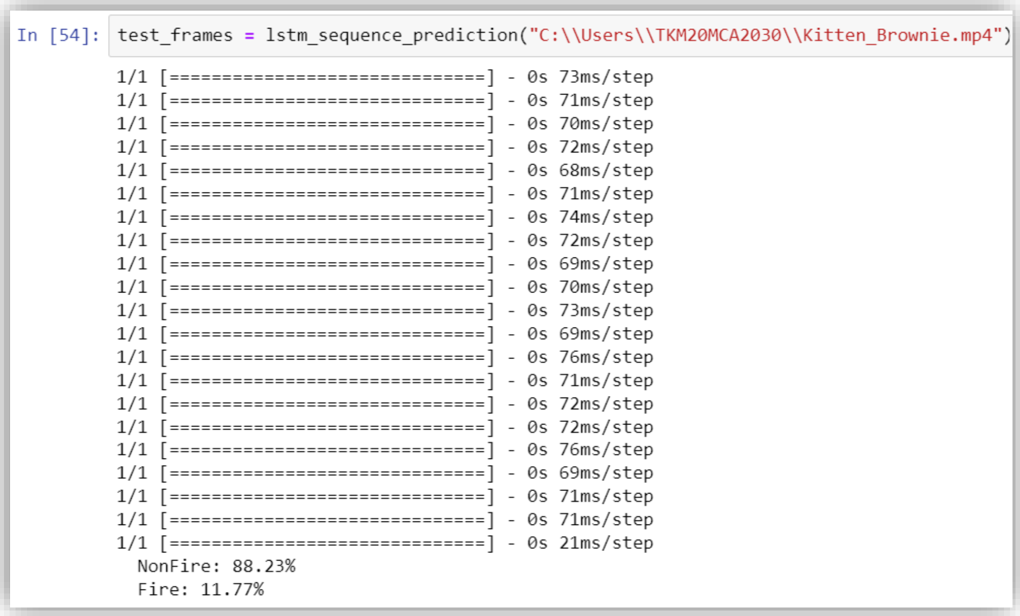
```
HTML("""
    <video alt="test" width="520" height="440" controls>
        <source src="Kitten_Brownie.mp4" type="video/mp4" embed="True" style="height:300px;width:300px">
    </video>
""")
```

Figure A.4: Real-time Test_Video_2



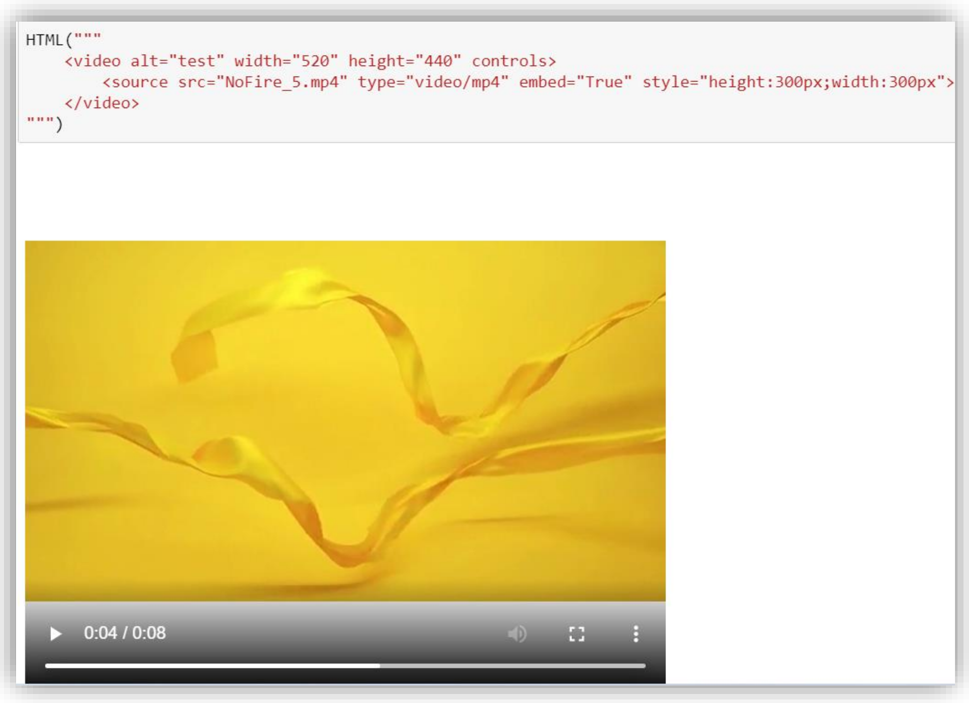
```
In [53]: test_frames = gru_sequence_prediction("C:\\Users\\TKM20MCA2030\\Kitten_Brownie.mp4")

         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 70ms/step
         1/1 [==============================] - 0s 20ms/step
           NonFire: 98.37%
           Fire:  1.63%
```

Figure A.5: Real-time Test_Video_2 GRU Prediction Result



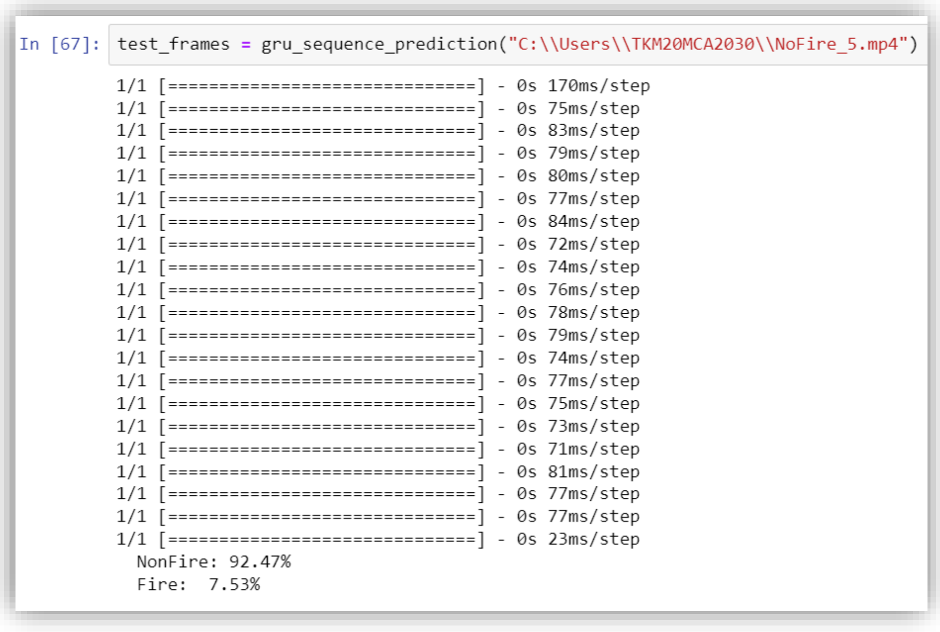
```
In [54]: test_frames = lstm_sequence_prediction("C:\\Users\\TKM20MCA2030\\Kitten_Brownie.mp4")

         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 70ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 68ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 70ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 21ms/step
           NonFire: 88.23%
           Fire: 11.77%
```

Figure A.6: Real-time Test_Video_2 LSTM Prediction Result



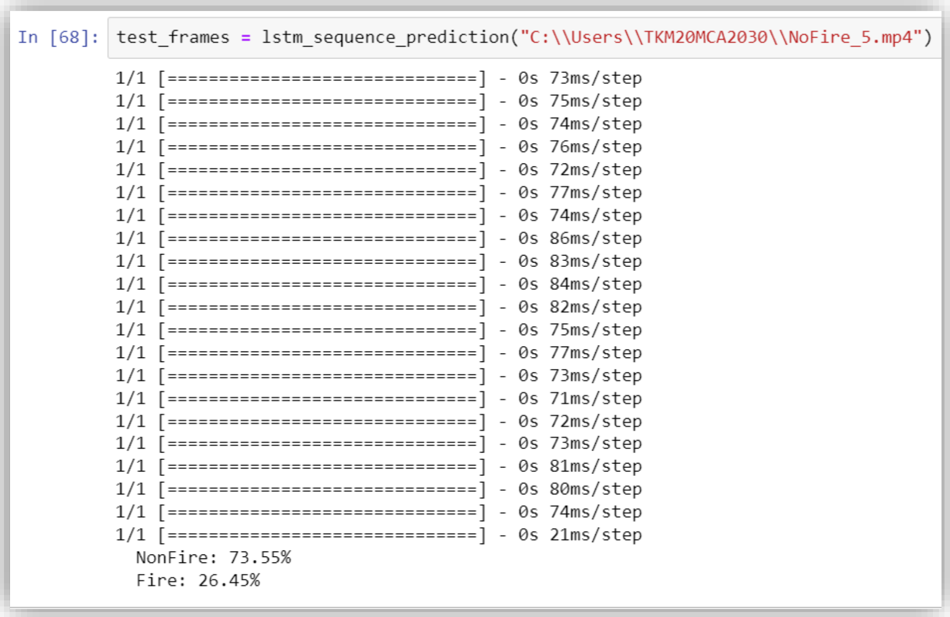
```
HTML("""
    <video alt="test" width="520" height="440" controls>
        <source src="NoFire_5.mp4" type="video/mp4" embed="True" style="height:300px;width:300px">
    </video>
""")
```

Figure A.7: Test_Video_D1 from Dataset



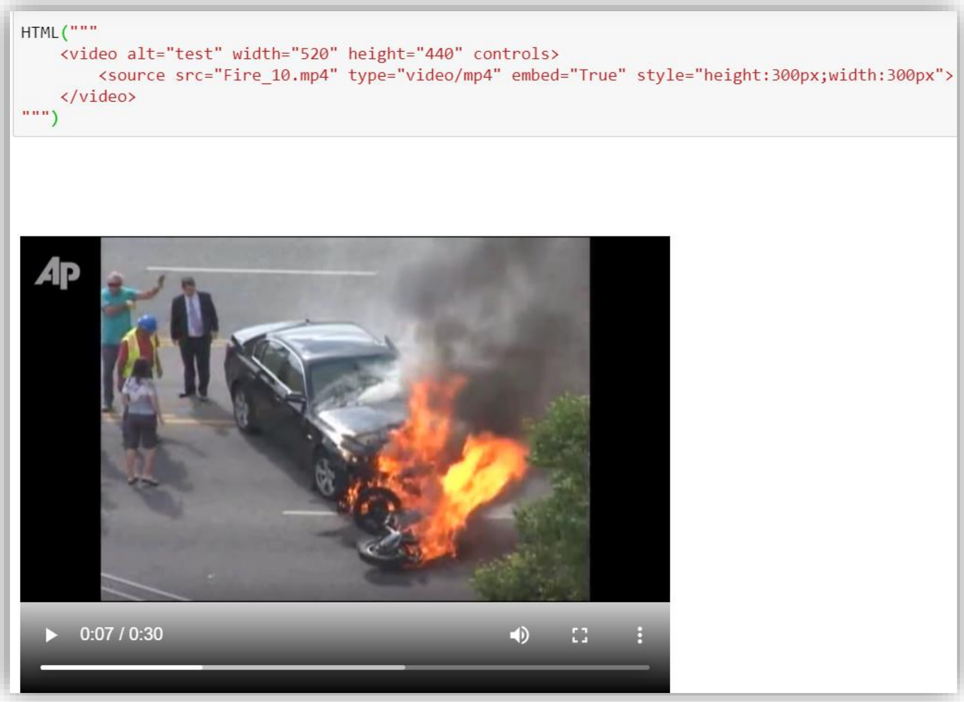
```
In [67]: test_frames = gru_sequence_prediction("C:\\Users\\TKM20MCA2030\\NoFire_5.mp4")

         1/1 [==============================] - 0s 170ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 83ms/step
         1/1 [==============================] - 0s 79ms/step
         1/1 [==============================] - 0s 80ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 84ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 78ms/step
         1/1 [==============================] - 0s 79ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 81ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 23ms/step
           NonFire: 92.47%
           Fire:  7.53%
```

Figure A.8: Test_Video_D1 GRU Prediction Result



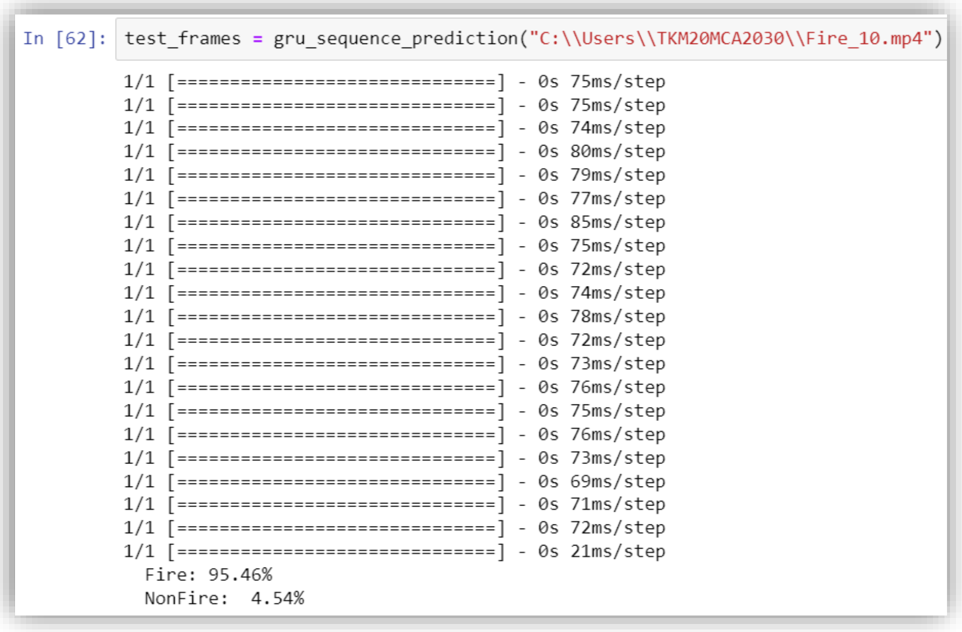
```
In [68]: test_frames = lstm_sequence_prediction("C:\\Users\\TKM20MCA2030\\NoFire_5.mp4")

         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 86ms/step
         1/1 [==============================] - 0s 83ms/step
         1/1 [==============================] - 0s 84ms/step
         1/1 [==============================] - 0s 82ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 81ms/step
         1/1 [==============================] - 0s 80ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 21ms/step
           NonFire: 73.55%
           Fire: 26.45%
```

Figure A.9: Test_Video_D1 LSTM Prediction Result



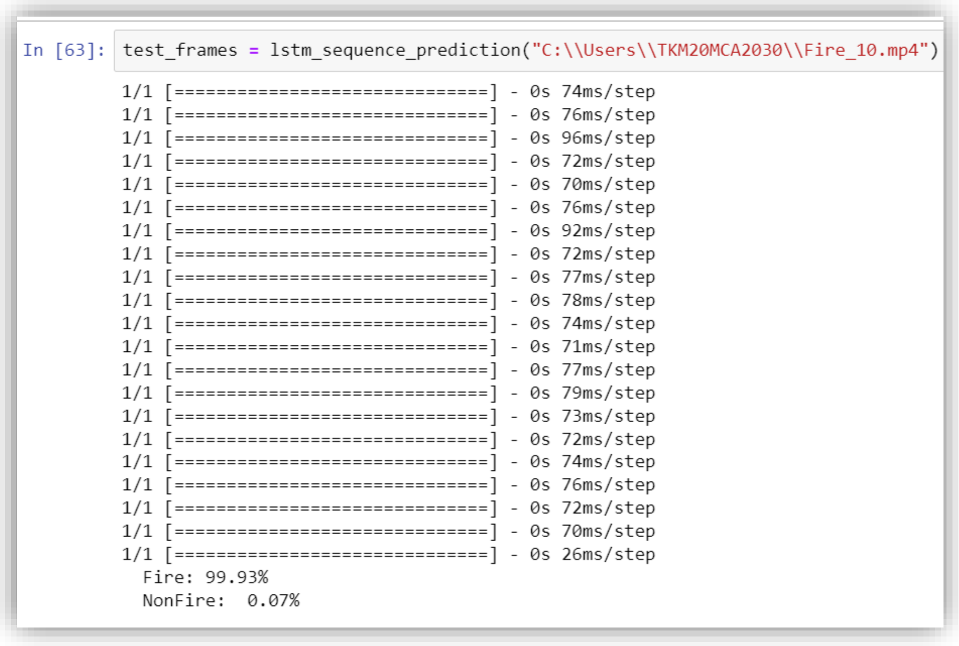
```
HTML("""
    <video alt="test" width="520" height="440" controls>
        <source src="Fire_10.mp4" type="video/mp4" embed="True" style="height:300px;width:300px">
    </video>
""")
```

Figure A.10: Test_Video_D2 from Dataset



```
In [62]: test_frames = gru_sequence_prediction("C:\\Users\\TKM20MCA2030\\Fire_10.mp4")

         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 80ms/step
         1/1 [==============================] - 0s 79ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 85ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 78ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 75ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 69ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 21ms/step
           Fire: 95.46%
           NonFire:  4.54%
```

Figure A.11: Test_Video_D2 GRU Prediction Result



```
In [63]: test_frames = lstm_sequence_prediction("C:\\Users\\TKM20MCA2030\\Fire_10.mp4")

         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 96ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 70ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 92ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 78ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 71ms/step
         1/1 [==============================] - 0s 77ms/step
         1/1 [==============================] - 0s 79ms/step
         1/1 [==============================] - 0s 73ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 74ms/step
         1/1 [==============================] - 0s 76ms/step
         1/1 [==============================] - 0s 72ms/step
         1/1 [==============================] - 0s 70ms/step
         1/1 [==============================] - 0s 26ms/step
           Fire: 99.93%
           NonFire:  0.07%
```

Figure A.12: Test_Video_D2 LSTM Prediction Result