

# Early Fire Detection using Deep Learning



A Minor Project Report

in partial fulfillment of the degree

## **Bachelor of Technology** in **Computer Science & Engineering**

**By**

Roll.No 19K41A05A2

Name KOLETI SRILATHA

Roll.No 19K41A0592

Name ANUGAM SAIKIRAN

Roll.No 19K41A05B6

Name THABETI PHANI PRIYA

Roll.No 19K41A0575

Name MD RAHAIL PASHA

**Under the Guidance of**

**Dr.J.Srinivas sir**

**Submitted to**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**  
**S.R.ENGINEERING COLLEGE(A),ANANTHASAGAR, WARANGAL**  
**(Affiliated to JNTUH, Accredited by NBA)**

**May, 2022.**



## **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

### **CERTIFICATE**

This is to certify that the Minor Project Report entitled “EARLY FIRE DETECTION” is a record of bonafide work carried out by the student(s) Koleti Srilatha, Anugam Saikiran, Thabeti Phani Priya, MD Rahail Pasha, bearing Roll No(s) 19K41A05A2, 19K41A0592, 19K41A05B6, 19K41A0575 during the academic year 2021-22 in partial fulfillment of the award of the degree of *Bachelor of Technology* in **Computer Science & Engineering** by the Jawaharlal Nehru Technological University, Hyderabad.

**Supervisor**

**Head of the Department**

**External Examiner**

## **Acknowledgement**

It brings us great pleasure to present the report of the Project Work completed during the third year of B.Tech. Dr.M.Sheshikala, Associate Professor,Head,CSE,SR Engineering College(A), Ananthasagar, Warangal, owes us a special debt of gratitude for his unwavering support and advice throughout my efforts. Our endeavours have only seen the light of day because of his conscientious efforts.

Dr.J.Srinivas, Assistant Professor, Department of Computer Science and Engineering, SR Engineering College(A),Ananthasagar,Warangal, the project's guide, deserves our heartfelt gratitude for patiently guiding and editing our different documents.

## **Abstract**

Fire is a flame, whether it is small or large, an undesirable place, situation and time. In general, every place has the potential to experience a fire. But at this time, the smoke sensors are the most widely used devices to detect fires. Where the smoke sensors can only detect fires if the fire is large. So that a system is needed to detect early fires. Fire is an abnormal event which can cause significant damage to lives and property. In this paper, we propose a deep learning-based fire detection method using a video sequence, which imitates the human fire detection process. In this project, a video-based fire alarm system is designed, using a laptop and webcam as the main equipment. The method for using Convolutional Neural Networks (CNN) to identify fire. The system created has an accuracy rate of 99.49%.

# CONTENTS

<b>Chapter No.</b>	<b>Title</b>	<b>Page No.</b>
<b>1.</b>	<b>Introduction</b>	<b>01</b>
1.1	Existing System	01
1.2	Proposed System	01-02
<b>2.</b>	<b>Literature Survey</b>	<b>03</b>
2.1	Related Work	03
<b>3.</b>	<b>Design</b>	<b>04</b>
3.1	Requirement Specifications (S/W & H/W)	04
3.2	Data Flow Diagram	04-05
<b>4.</b>	<b>Implementation</b>	<b>06</b>
4.1	Modules	06-08
4.2	Overview Technology	09-15
<b>5.</b>	<b>Testing</b>	<b>16</b>
5.1	Test Cases	16
5.2	Test Results	16-17
<b>6.</b>	<b>Results</b>	<b>18</b>
6.1	Comparitive Results Analysis	18
<b>7.</b>	<b>Conclusion</b>	<b>19</b>
<b>8.</b>	<b>Future Scope</b>	<b>20</b>
	<b>Bibliography</b>	<b>21</b>

# 1. INTRODUCTION

Fire accidents pose a serious threat to industries, crowded events, social gatherings, and densely populated areas that are observed across India. These kinds of incidents may cause damage to property, environment, and pose a threat to human and animal life. According to the recent National Risk Survey Report, Fire stood at the third position overtaking corruption, terrorism, and insurgency thus posing a significant risk to our country's economy and citizens. The recent forest-fires in Australia reminded the world, the destructive capability of fire and the impending ecological disaster, by claiming millions of lives resulting in billions of dollars in damage. Early detection of fire-accidents can save innumerable lives along with saving properties from permanent infrastructure damage and the consequent financial losses. In order to achieve high accuracy and robustness in dense urban areas, detection through local surveillance is necessary and also effective. Traditional opto-electronic fire detection systems have major disadvantages: Requirement of separate and often redundant systems, fault-prone hardware systems, regular maintenance, false alarms and so on. Usage of sensors in hot, dusty industrial conditions is also not possible. Thus, detecting fires through surveillance video stream is one of the most feasible, cost-effective solution suitable for replacement of existing systems without the need for large infrastructure installation or investment. The existing video-based machine learning models rely heavily on domain knowledge and feature engineering to achieve detection therefore, have to be updated to meet new threats. We aim to develop a classification model using Deep learning and Transfer Learning to recognise fires in images/video frames, thus ensuring early detection and save manual work.

## 1.1 EXISTING SYSTEM

In conventional fire detection, approaches require domain knowledge of fires in captured images essential to explore hand-crafted features and cannot reflect the information spatially and temporally involved in fire environments well. In addition, almost all methods using the conventional approach only use a still image or consecutive pairs of frames to detect fire. Therefore, they only consider the short-term dynamic behavior of fire, whereas a fire has a longer-term dynamic behavior.

## 1.2 PROPOSED SYSTEM

By the great potential of CNNs, we detect fire from videos at an early stage. Our system has two custom models for fire detection. Our model classifies the real-time videos such that the whole video frame turns into black and white when it detects fire in it. It detects the fire early thus making us to alert and take action. Considering the fair fire detection accuracy of the CNN

model, it can be of assistance to disaster management teams in managing fire disasters on time, thus preventing huge losses.

## 2.LITERATURE SURVEY

### 2.1 RELATED WORK

In comparison to conventional object detection, there are few publications on fire detection in computer vision. The vast majority of the literatures use motion and colour data [1, 2] or spatial temporal features [3] since fire is a non-rigid object with dynamic shapes. Aside from that, the majority of the work assumes fire detection in a surveillance situation, in which a camera is fixed in one location and is used to detect flames and/or smokes.

As a result, background removal is commonly used to increase accuracy and robustness [4]. We are not assuming a surveillance situation in this paper; instead, our technology can be used in more complex dynamic environments, such as placing cameras on UAVs. Background subtraction and motion analysis become difficult as a result. The classic fire detection pipeline can be characterised as follows, based on earlier literature: (1) Exaction of moving pixels and regions [5, 4]. (2) Extraction of flame and/or smoke region candidates using a colour model, such as the HIS colour model utilised in [5, 4]. (3) Additional research into the candidate regions, such as foreground region analysis [5] and fire dynamic behaviour analysis [6]. [2]. We employ CNN to learn feature representations and fire classifiers instead of the standard vision-based fire detection process. For several computer vision tasks, such as object identification [7], detection [8], and semantic segmentation [9], deep convolutional neural networks have demonstrated state-of-the-art performance. Learning-based approaches are, in reality, hardly examined in past research.[10] suggested a covariance descriptor based on colour, spatial, and temporal information, as well as training an SVM as a classifier in a recent paper.



### 3.DESIGN

#### 3.1 REQUIREMENT SPECIFICATION(S/W & H/W)

##### Hardware Requirements

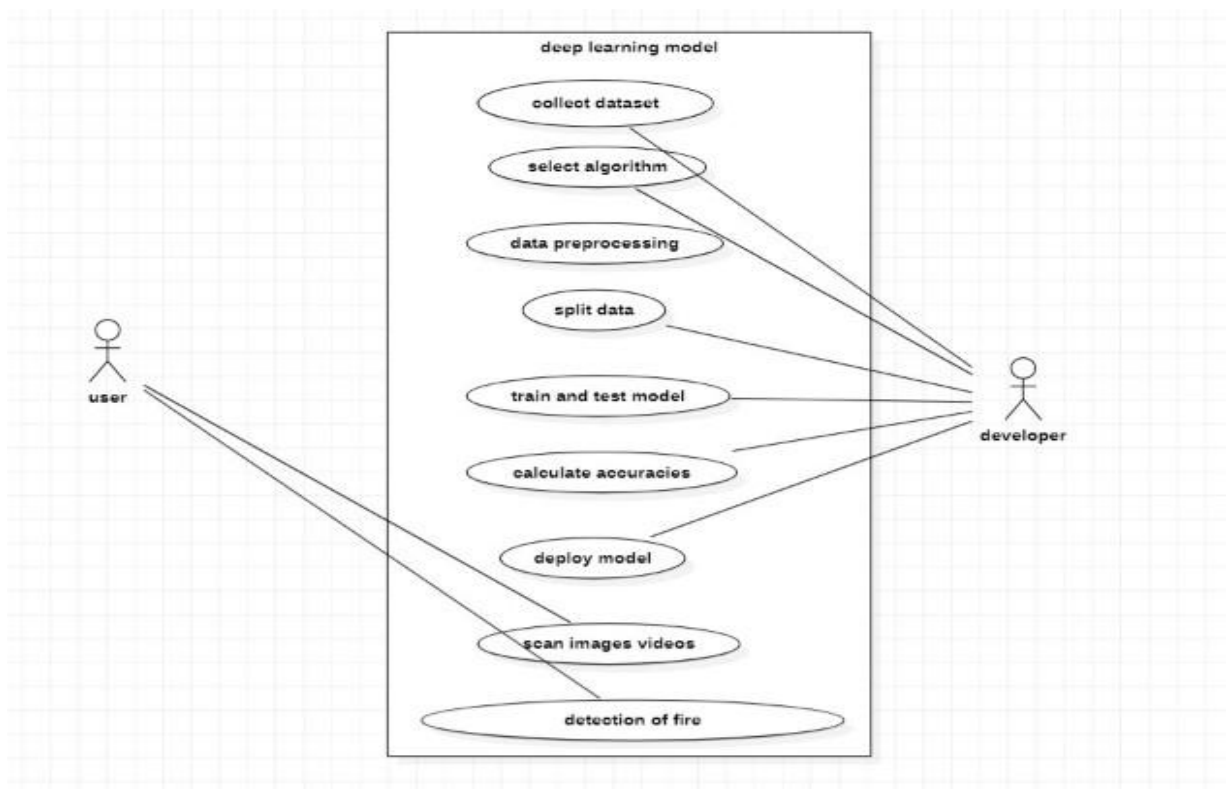
- ✓ **System** : Pentium 4, Intel Core i3, i5, i7 and 2GHz Minimum
- ✓ **RAM** : 4GB or above
- ✓ **Hard Disk** : 10GB or above
- ✓ **Input** : Keyboard and Mouse
- ✓ **Output** : Monitor or PC

##### Software Requirements

- ✓ **OS** : Windows 8 or Higher Versions
- ✓ **Platform** : Jupiter Notebook
- ✓ **Program Language** : Python

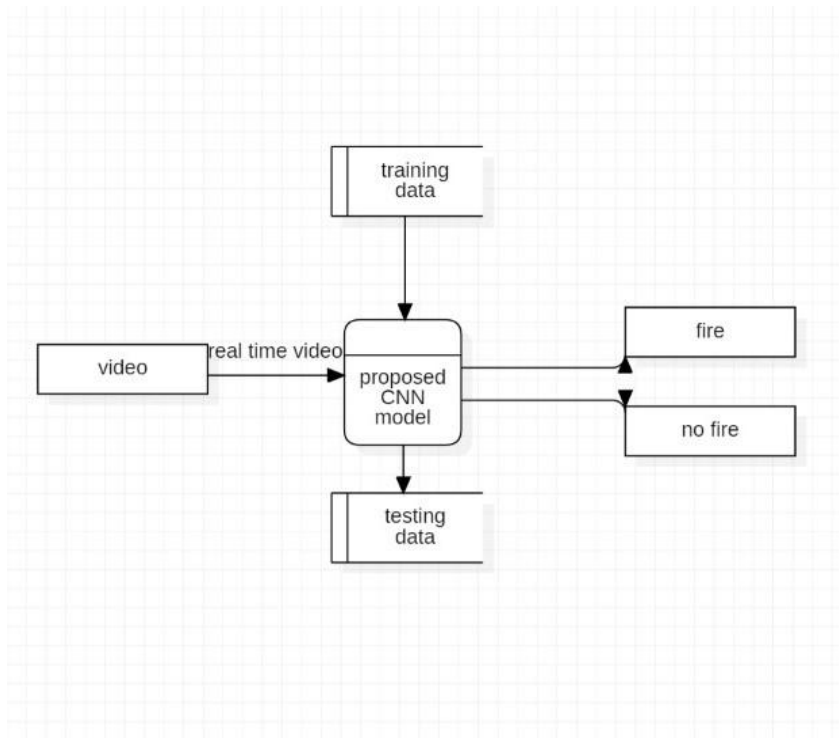
#### 3.2 UML DIAGRAMS

##### USE CASE DIAGRAM



**Fig.1** Usecase diagram of Fire Detection

## DATA FLOW DIAGRAM



**Fig.2** Dataflow diagram of Fire Detection

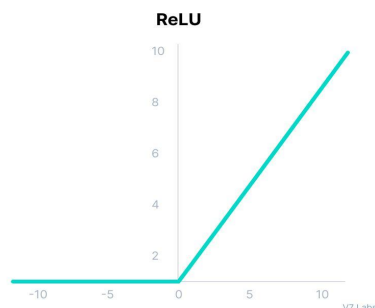
## 4.IMPLEMENTATION

### 4.1. MODULES

#### ACTIVATION FUNCTIONS

##### ReLU activation function :

ReLU stands for Rectified Linear Unit. ReLU has a derivative function and enables for back propagation while being computationally efficient, even though it seems to be a linear function. The key caveat is that the ReLU function does not simultaneously stimulate all of the neurons. Only if the linear transformation output is less than 0 will the neurons be silenced.



**Fig.3 : ReLU Activation Function**

Mathematically it can be represented as:

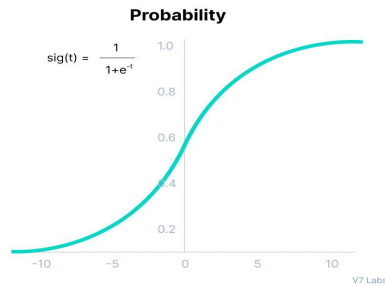
$$\text{ReLU}$$
$$f(x) = \max(0, x)$$

The following are some of the benefits of utilizing ReLU as an activation function:

The ReLU function is significantly more computationally efficient than the sigmoid and tanh functions since only a small number of neurons are engaged. Due to its linear, non-saturating characteristic, ReLU speeds the convergence of gradient descent towards the global minimum of the loss function. The Dying ReLU issue is one of the function's limitations.

##### Softmax output function

Before delving into the details of the Softmax activation function, let's look at one of its components: the sigmoid/logistic activation function, which calculates probability values.



**Fig.4 : Probability using softmax**

The sigmoid function's output ranged from 0 to 1, which may be interpreted as a probability. However, there are several issues with this feature. Let's say we have five different output values: 0.8, 0.9, 0.7, 0.8, and 0.6. What are our options for moving forward? 32

We are unable to do so. The figures above are incorrect since the total of all the classes/output probabilities should be 1. As you can see, the Softmax function is made up of many sigmoids. The relative probability is calculated. The SoftMax function, like the sigmoid/logistic activation function, returns the probability of each class.

Mathematically it can be represented as:

$$\text{Softmax}$$

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

V7 Labs

## OPTIMIZERS

### RMSprop Optimizer:

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9. If you are not interested in the math behind the optimizer, you can just skip the following equations.

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw$$

$$v_{db} = \beta \cdot v_{db} + (1 - \beta) \cdot db$$

$$W = W - \alpha \cdot v_{dw}$$

$$b = b - \alpha \cdot v_{db}$$

Gradient descent with momentum

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{db} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

RMSprop optimizer

Sometimes the value of  $v_{dw}$  could be really close to 0. Then, the value of our weights could blow up. To prevent the gradients from blowing up, we include a parameter epsilon in the denominator which is set to a small value.

### Stochastic Gradient Descent (SGD):

The word ‘*stochastic*’ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets get big. Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima are reached. Hence, it becomes computationally very expensive to perform.

This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

### SGD algorithm:

for  $i$  in range ( $m$ ) :

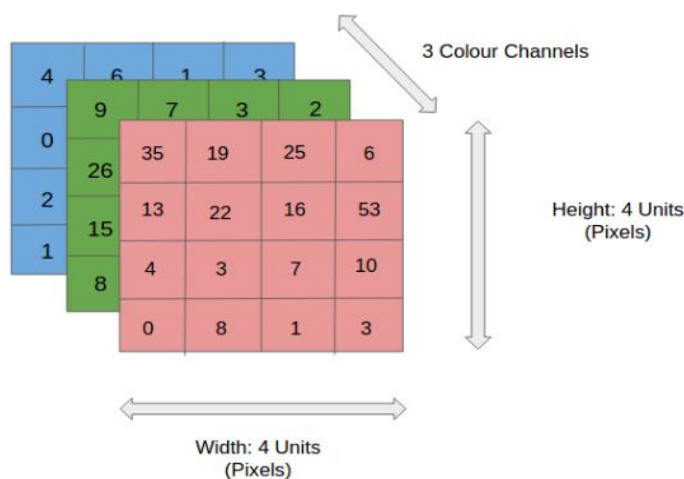
$$\theta_j = \theta_j - \alpha (\hat{y}^i - y^i) x_j^i$$

## 4.2. OVERVIEW TECHNOLOGY

### Convolutional Neural Networks (CNN)

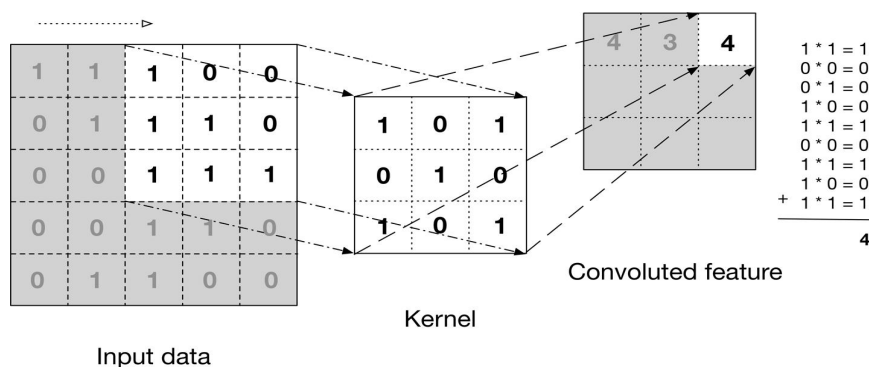
A convolutional neural network (CNN/ConvNet) is a type of deep neural network used to evaluate visual images in deep learning. When we think about neural networks, we usually think of matrix multiplications, but this isn't the case with ConvNet. It employs a method known as Convolution. Convolution is a mathematical procedure that creates a third function that indicates how the form of one is affected by the other.

An RGB image is nothing more than a three-dimensional matrix of pixel values, whereas a grayscale image is the same but only contains one plane. To learn more, check out the image below.



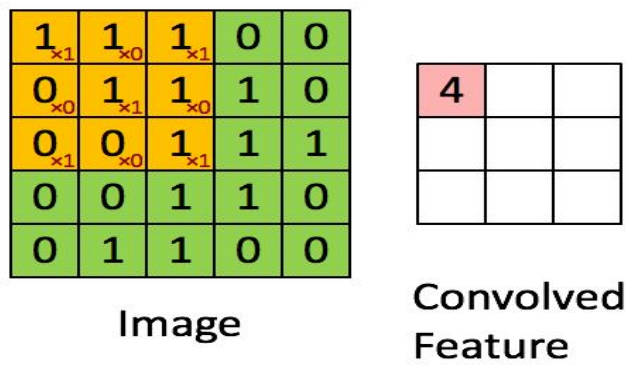
**Fig.5-**Matrix of an RGB image

To keep things simple, we'll use grayscale photos to explain how CNNs function.



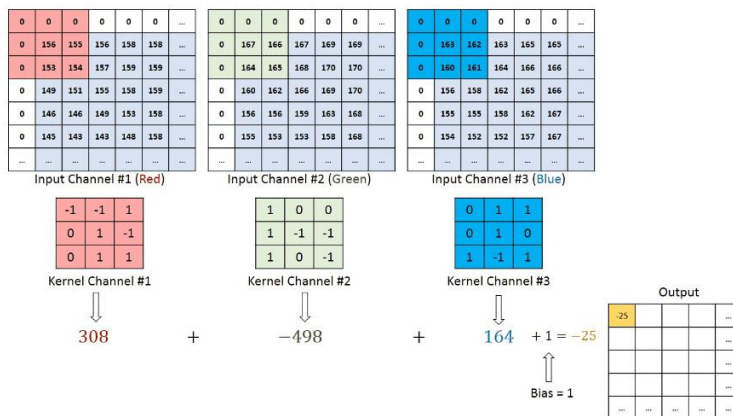
**Fig.6-**Image representing convolution for a filter size 3\*3

A convolution is seen in the figure above. To obtain the convolved feature, we apply a filter/kernel (3x3 matrix) to the input picture. The following layer receives this convolved feature.



**Fig.7-** image and a convolved feature

In the case of RGB color, channel take a look at this animation to understand its working.



**Fig.8-**Working of a convolutional layer

Multiple layers of artificial neurons make up convolutional neural networks. Artificial neurons are mathematical functions that calculate the weighted sum of various inputs and output an activation value, similar to their biological counterparts. Each layer creates many activation functions that are passed on to the next layer when you input an image into a ConvNet.

Basic characteristics such as horizontal or diagonal edges are generally extracted by the first layer. This information is passed on to the next layer, which is responsible for detecting more complicated characteristics like corners and combinational edges. As we go further into the network, it can recognize increasingly more complex elements like objects, faces, and so on.

The classification layer generates a series of confidence ratings (numbers between 0 and 1) based on the activation map of the final convolution layer, which indicates how likely the picture is to belong to a "class".

### Pooling layer

The Pooling layer, like the Convolutional Layer, is responsible for shrinking the

Convolved Feature's spatial size. By lowering the size, the computer power required to process the data is reduced. Average pooling and maximum pooling are the two forms of pooling. So far, I've just worked with Max Pooling and haven't run across any issues.

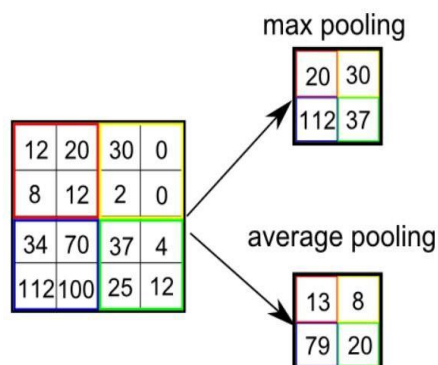
3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

**Fig.9-Pooling layer**

We determine the largest value of a pixel from a region of the picture covered by the kernel using Max Pooling. Max Pooling works as a Noise Suppressant as well. It removes all noisy activations and conducts de-noising and dimensionality reduction at the same time.

Average Pooling, on the other hand, returns the average of all values from the Kernel's covered region of the picture. As a noise-suppressing strategy, Average Pooling merely reduces dimensionality. As a result, Max Pooling outperforms Average Pooling significantly.



**Fig.10-Pooling layer with max pooling and average pooling**

## InceptionV3

Inception V3 by Google is the 3rd version in a series of Deep Learning Convolutional Architectures. Inception V3 was trained using a dataset of 1,000 classes (See the list of classes [here](#)) from the original ImageNet dataset which was trained with over 1 million training images, the Tensorflow version has 1,001 classes which is due to an additional "background" class not used in the original ImageNet. Inception V3 was trained for the ImageNet Large Visual Recognition Challenge where it was a first runner up.

The inception V3 is just the advanced and optimized version of the inception V1 model. The Inception V3 model used several techniques for optimizing the network for better model adaptation.



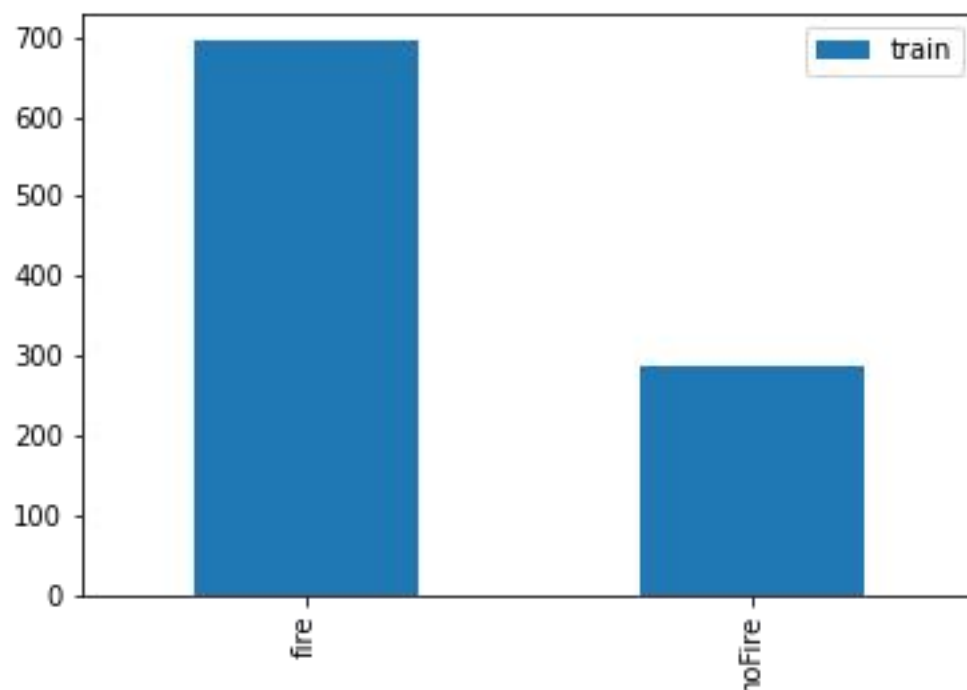
- It has higher efficiency
- It has a deeper network compared to the Inception V1 and V2 models, but its speed isn't compromised.
- It is computationally less expensive.
- It uses auxiliary Classifiers as regularizes.
- Inception V3 Model Architecture
- The inception v3 model was released in the year 2015, it has a total of 42 layers and a lower error rate than its predecessors. Let's look at what are the different optimizations that make the inception V3 model better.

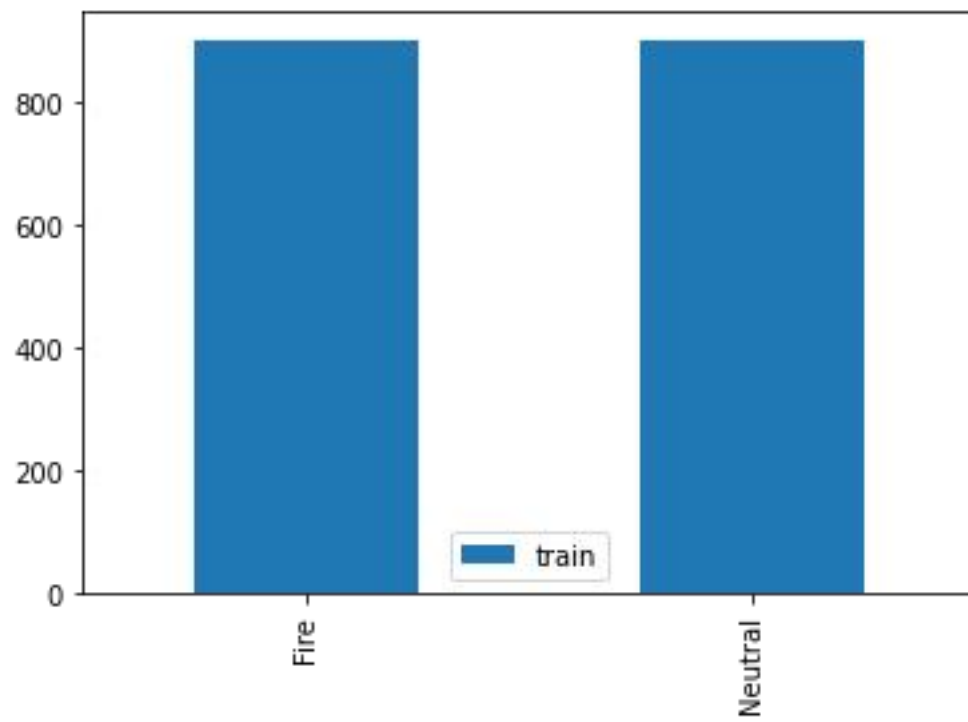
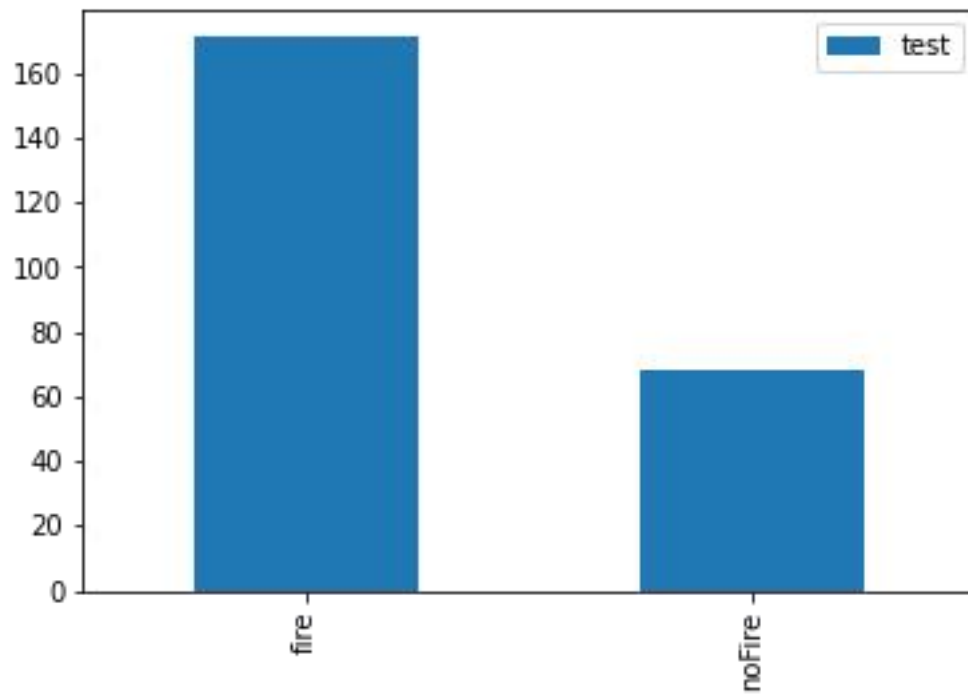
The major modifications done on the Inception V3 model are

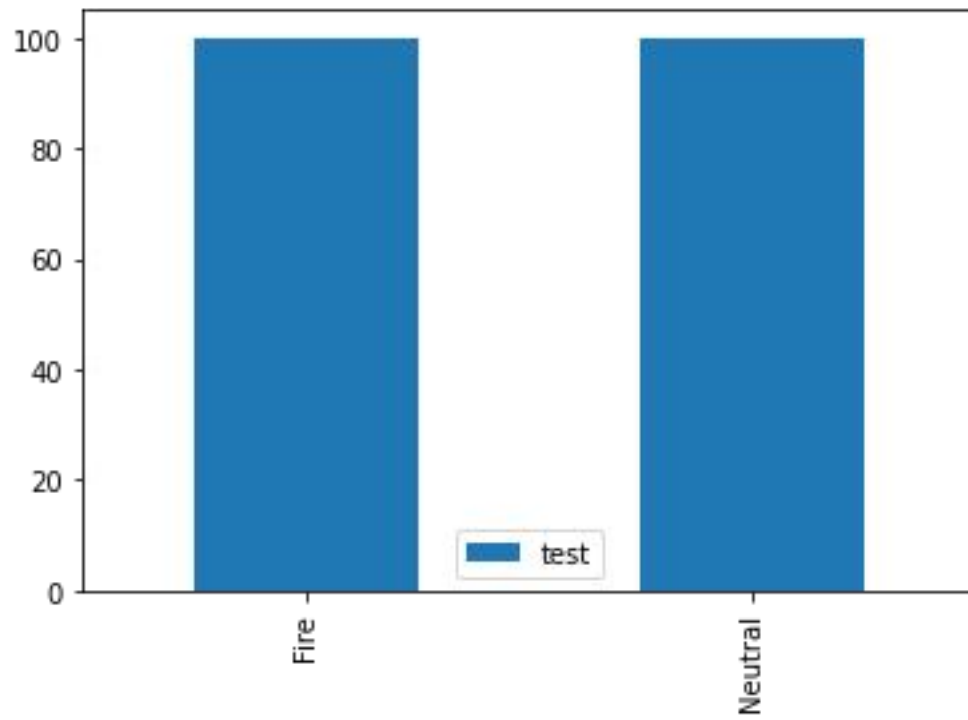
1. Factorization into Smaller Convolutions
2. Spatial Factorization into Asymmetric Convolutions
3. Utility of Auxiliary Classifiers
4. Efficient Grid Size Reduction

### **Data Visualization using Graphs**

In order to understand the data more deeply we have used histograms to analyze the image data . In the Figures we have shown the histograms for all datasets. Through this we can understand the data more efficiently.

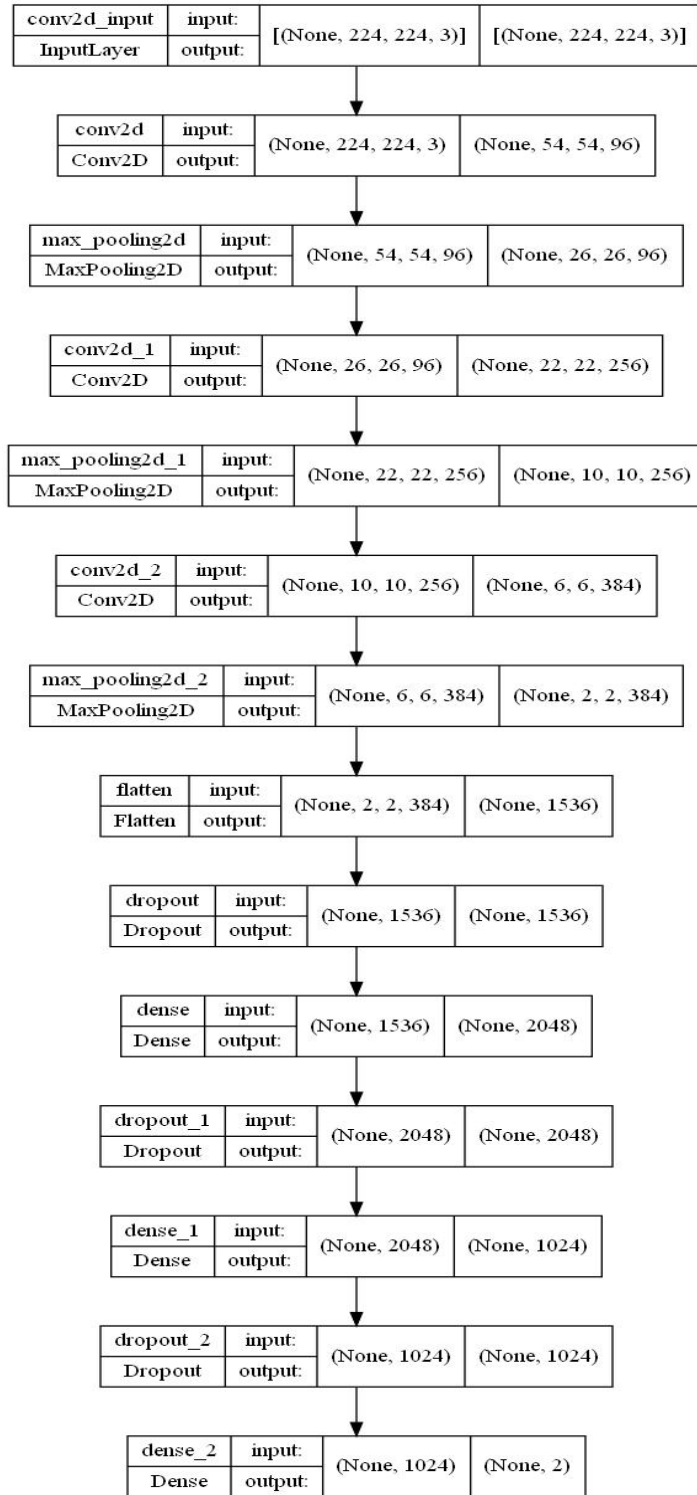






**Fig. 11** Visualization of Data using Graphs

## System Architecture



*Fig. 12 Discriminator Architecture*

## 5.TESTING

### 5.1.TEST CASES

- As shown in the figure ,the data that the model takes are video frames or live web feed or an image.The screen turns to black and white from color when it detects fire in the data.
- The screen remains same as color when it does not detect fire.

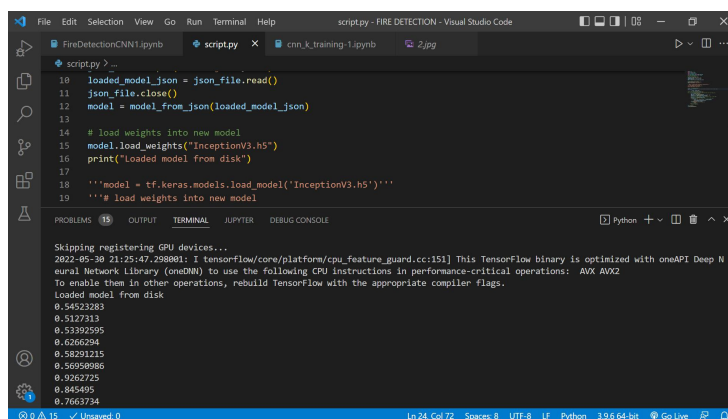
### 5.2.TEST RESULTS



*Fig. 13 Image with fire*

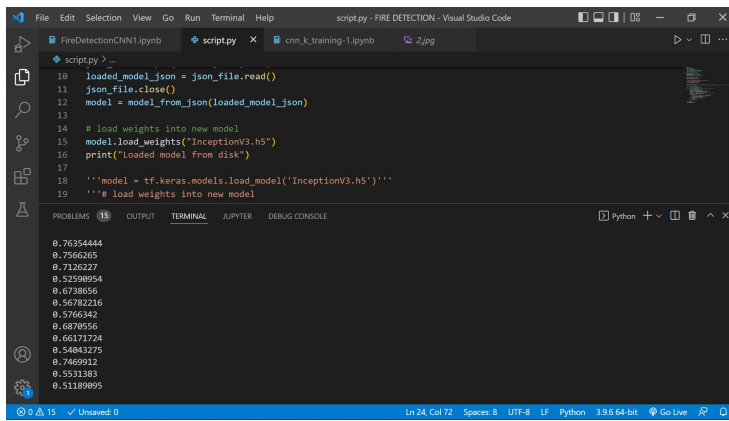


*Fig. 14 Detected fire in image*



```
script.py - FIRE DETECTION - Visual Studio Code
FireDetectionCNN1.ipynb script.py x cnn_k_training-1.ipynb 2.jpg
script.py > _
18 loaded_model_json = json_file.read()
19 json_file.close()
12 model = model_from_json(loaded_model_json)
13
14 # load weights into new model
15 model.load_weights("InceptionV3.h5")
16 print("Loaded model from disk")
17
18 '''model = tf.keras.models.load_model('InceptionV3.h5')'''
19 '''# load weights into new model
20
21 # skip registering GPU devices...
22 2022-05-30 21:25:47.298081: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following GPU instructions in performance-critical operations: AVX AVX2
23 To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
24 Loaded model from disk
25 0.54523283
26 0.5127213
27 0.53392595
28 0.6266294
29 0.58291215
30 0.56959085
31 0.9262725
32 0.845495
33 0.7663734
```

*Fig.15 Accuracies generated during fire detection*



The screenshot shows a Visual Studio Code editor with a terminal window open. The terminal displays the output of a Python script. The code in the background is as follows:

```
10 loaded_model_json = json_file.read()
11 json_file.close()
12 model = model_from_json(loaded_model_json)
13
14 # load weights into new model
15 model.load_weights("InceptionV3.h5")
16 print("Loaded model from disk")
17
18 '''model = tf.keras.models.load_model('InceptionV3.h5')'''
19 '''# load weights into new model
```

The terminal output shows a list of 10 floating-point numbers, which are the accuracies generated during fire detection:

```
0.76354444
0.7566265
0.7126227
0.52928954
0.6738656
0.56782216
0.5766342
0.6870556
0.66171724
0.54843275
0.7469912
0.55131383
0.51189095
```

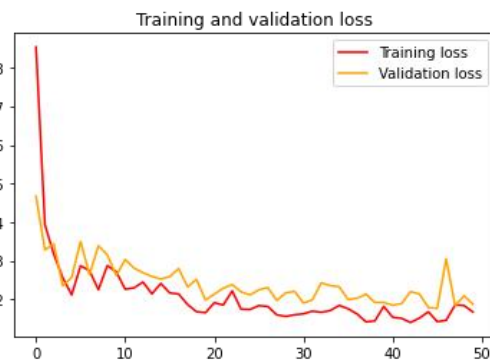
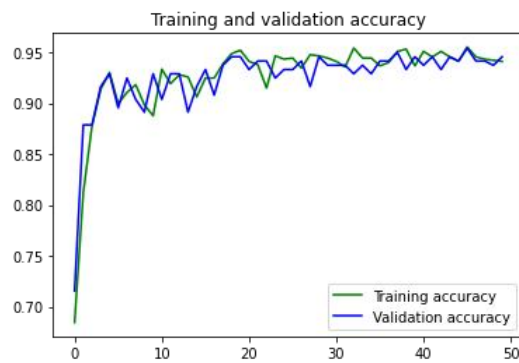
**Fig.16** *Accuracies generated during fire detection*

## 6.RESULTS

### 6.1 COMPARATIVE RESULTS ANALYSIS

OPTIMIZER	TRAINING ACCURACY	VALIDATION ACCURACY
ADAM	96.83%	94.98%
RMSPROP	97.65%	99.49%
SGD	98.04%	96.43%

**Table.1** Comparative results



**Fig.17** Training and Validation Accuracy graph

**Fig.18** Training and Validation Loss graph

```

87916544/87910968 [=====] - 92s 1us/step
87924736/87910968 [=====] - 92s 1us/step
Epoch 1/20
14/14 [=====] - 165s 11s/step - loss: 0.3515 - acc: 0.7590 - val_loss: 0.0532 - val_acc: 0.9949
Epoch 2/20
14/14 [=====] - 158s 11s/step - loss: 0.2497 - acc: 0.9079 - val_loss: 0.0815 - val_acc: 0.9847
Epoch 3/20
14/14 [=====] - 155s 11s/step - loss: 0.0697 - acc: 0.9743 - val_loss: 0.0985 - val_acc: 0.9541
Epoch 4/20
14/14 [=====] - 147s 10s/step - loss: 0.2984 - acc: 0.9133 - val_loss: 0.0342 - val_acc: 0.9898
Epoch 5/20
14/14 [=====] - 123s 8s/step - loss: 0.1611 - acc: 0.9510 - val_loss: 0.0406 - val_acc: 0.9745
Epoch 6/20
14/14 [=====] - 123s 9s/step - loss: 0.0571 - acc: 0.9815 - val_loss: 0.1528 - val_acc: 0.9286
Epoch 7/20
14/14 [=====] - 167s 12s/step - loss: 0.2076 - acc: 0.9270 - val_loss: 0.0852 - val_acc: 0.9592
Epoch 8/20
14/14 [=====] - 151s 11s/step - loss: 0.0888 - acc: 0.9719 - val_loss: 0.4210 - val_acc: 0.7908
Epoch 9/20
14/14 [=====] - 143s 10s/step - loss: 0.1552 - acc: 0.9390 - val_loss: 0.0308 - val_acc: 0.9949
Epoch 10/20
14/14 [=====] - 141s 10s/step - loss: 0.0774 - acc: 0.9761 - val_loss: 0.1520 - val_acc: 0.9082
Epoch 11/20
14/14 [=====] - 140s 10s/step - loss: 0.0454 - acc: 0.9850 - val_loss: 0.0342 - val_acc: 0.9949
...
Epoch 19/20
14/14 [=====] - 144s 10s/step - loss: 0.0482 - acc: 0.9815 - val_loss: 0.0072 - val_acc: 0.9949
Epoch 20/20
14/14 [=====] - 141s 10s/step - loss: 0.0124 - acc: 0.9952 - val_loss: 0.0094 - val_acc: 0.9949

```

**Fig.19** Training and Validation Accuracy(RMSprop)

## 7.CONCLUSION

Using smart cameras you can identify various suspicious incidents such as collisions, medical emergencies, and fires. Of such, fire is the most dangerous abnormal occurrence, because failure to control it at an early stage can lead to huge disasters, leading to human, ecological and economic losses. The successful implementation of convolutional neural networks enhances object detection ability dramatically. Finally, our model has obtained the highest accuracy with RMSprop optimizer of 99.49% which helps us to detect the fire.



## 8.FUTURE SCOPE

- This fire detection system can be extended adding a messaging technique which alters all the fire stations and police stations which are in a particular radius and at the same time adding alarm for the people in that particular area.
- We'd like to expand the benchmark and make finer-grained annotations in the future, for example, by leveraging the compute graphic engine to create synthetic data and generate pixel-wise annotations for free. We also intend to use UAVs to identify forest fires in the real world using the suggested fire detection system.

## BIBLIOGRAPHY

- [1] H. O. H. U. M. Celik, T.; Demirel, “Fire detection in video sequences using statistical color model,” in IEEE International Conference on Acoustics, Speech and Signal Processing, 2006.
- [2] I. K. Martin Mueller, Peter Karasev and A. Tannenbaum, “Optical flow estimation for flame detection in videos,” IEEE Trans. on Image Processing, vol. 22, no. 7, 2013.
- [3] N. A. Che-Bin Liu, “Vision based fire detection,” in Int. Conf. in Pattern Recognition, 2004.
- [4] P. Gomes, P. Santana, and J. Barata, “A vision-based approach to fire detection,” International Journal of Advanced Robotic Systems, 2014.
- [5] X. Z. Chunyu Yu, Zhibin Mei, “A real-time video fire flame and smoke detection algorithm,” in Asia-Oceania Symposium on Fire Science and Technology, 2013.
- [6] A. E. C. B. Ugur Toreyin, “Online detection of fire in video,” in IEEE Conf. on Computer Vision and Pattern Recognition, 2007.
- [7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks.” In Advances in Neural Information Processing Systems, Lake Tahoe, Nevada, USA, 2012.
- [8] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in IEEE Conf. on Computer Vision and Pattern Recognition, Columbus, Ohio, USA, 2014.
- [8] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in IEEE Conf. on Computer Vision and Pattern Recognition, 2015.
- [10] Y. Habibolu, O. Gnay, and A. etin, “Covariance matrix-based fire and flame detection method in video,” Machine Vision and Applications, vol. 23, no. 6, pp. 1101113, 2011.