



A LIGHT WEIGHT CNN MODEL FOR FIRE DETECTION

A PROJECT REPORT

Submitted by

SOWMIYA.S [REGISTER NO:211417104266]

SUSHMITHA.B [REGISTER NO:211414104279]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2021

BONAFIDE CERTIFICATE

Certified that this project report “**A LIGHT WEIGHT CNN MODEL FOR FIRE DETECTION**” is the bonafide work of “**SOWMIYA.S(211417104266), SUSHMITHA.B(211417104279)**” who carried out the project work under my supervision.

SIGNATURE

**Dr.S.MURUGAVALLI,M.E.,Ph.D.,
HEAD OF THE DEPARTMENT**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

**P.DEEPA M.E.
SUPERVISOR
ASSOCIATE PROF**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project

Viva-Voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like express our heartfelt and sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI, Dr.C.SAKTHIKUMAR, M.E., Ph.D.,** and **Tmt. SARANYASREE SAKTHIKUMAR B.E.,M.B.A.,** for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.Mani, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CSE Department, **Dr. S.MURUGAVALLI , M.E.,Ph.D.,** for the support extended throughout the project.

We would like to thank our **Project Guide Mrs.P.DEEPA M.E** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

SOWMIYA.S

SUSHMITHA.B

ABSTRACT

Vision based fire detection framework has lately picked up popularity when contrasted with customary fire recognition framework dependent on sensors. The need of video perception at private, Modern, business regions and woods areas has expanded the use of vision based fire acknowledgment system. Recently lots of fire related accidents has occurred due to improper Surveillance or unable to cover those uncertain regions like restricted areas in forest or any factory buildings. In order to overcome such accidents, this project provides a new method using Convolutional neural networks (RCNN).

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	LIST OF FIGURES	v
	LIST OF SYMBOLS, ABBREVIATIONS	vi
1.	INTRODUCTION	1
	1.1 Overview	1
	1.2 Problem Definition	2
2.	LITERATURE SURVEY	3
3.	SYSTEM ANALYSIS	6
	3.1 Existing System	6
	3.2 Proposed system	6
	3.3 Requirement Analysis and Specification	7
	3.3.1 Input Requirements	7
	3.3.2 Output Requirements	7
	3.3.3 Functional Requirements	7
	3.4 Technology Stack	7

CHAPTER NO.	TITLE	PAGE NO.
4.	SYSTEM DESIGN	10
	4.1. ER diagram	10
	4.2 Data dictionary	10
	4.3 UML Diagrams	11
5.	SYSTEM ARCHITECTURE	16
	5.1 Architecture Overview	16
	5.2 Module Design Specification	17
	5.3 Program Design Language	18
6.	SYSTEM IMPLEMENTATION	33
	6.1 Coding	33
7.	SYSTEM TESTING	40
	7.1 Unit Testing	40
	7.2 Integration Testing	40
	7.3 Test Cases & Reports / Performance Analysis	41
8.	CONCLUSION	43
	8.1 Conclusion and Future Enhancements	43
	APPENDICES	44
	A.1 Sample Screens	44
	A.2 Publications	49
	REFERENCES	54

LIST FIGURES

4.1 ER Diagram	10
4.2 Firebase Diagram... ..	11
4.3.1 Use Case Diagram.....	12
4.3.2 Sequence Diagram	13
4.3.3 Activity Diagram.....	14
4.3.4 Collaboration Diagram	14
4.3.5 Data Flow Diagram	15
5.1 System Architecture	16
A.1.1 Folders and Collected Datasets... ..	44
A.1.2 Fire and Non-Fire Images for Datasets.....	44
A.1.3 Screenshot for Anaconda Prompt.....	45
A.1.4 Fire Detection... ..	45
A.1.5 Non-Fire Detection... ..	46
A.1.6 Homepage of Firebase Console	46
A.1.7 Firebase Showing the Update Status as Fire.....	47
A.1.8 Firebase Showing the Update Status as Non-Fire	47
A.1.9 Android App for Fire Detection... ..	48
A.1.10 Notification from App as Fire Detected... ..	48

LIST OF SYMBOLS, ABBREVIATION

CNN -- Convolutional Neural Networks

ORM – Object-relation mapping. The set of rules for mapping objects in a programming language to records in a relational database, and vice versa.

DBMS – Database management system. Software for maintaining, querying and updating data and metadata in a database.

RGB – Red, Green, Blue.

CMYK – Cyan, Magenta, Yellow, Key.

HIS – Hue, Saturation, Intensity.

OS – Operating System.

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Fire detection is crucial task for the safety of people. To prevent damages caused by fire, several fire detection systems were developed. One can find different technical solutions. Most of them are based on sensors, which is also generally limited to indoors. However, those methods have a fatal flaw where they will only work on reaching a certain condition. In the worst-case scenario, the sensors are damaged or not being configured properly can cause heavy casualty in case of real fire. Those sensors detect the particles produced by smoke and fire by ionization, which requires a close proximity to the fire. Consequently, they cannot be used for covering large area. To get over such limitations video fire detection systems are used. Due to rapid developments in digital cameras and video processing techniques, there is a significant tendency to switch to traditional fire detection methods with computer vision-based systems. Video-based fire detection techniques are well suited for detecting fire in large and open spaces. Nowadays, closed circuit television surveillance systems are installed in most of the places monitoring indoors and outdoors. Under this circumstance, it would be an advantage to develop a video-based fire detection system, which could use these existing surveillance cameras without spending any extra cost. This type of systems offers various advantages over those standard detection methods. For example, the cost of using this type of detection is cheaper and the implementation of this type system is greatly simpler compare to those traditional methods. Secondly, fire detection system responds faster compared to any other traditional detection methods because a vision-based fire detection system does not require any type conditions to trigger the devices and it has the ability to monitor a large area.

1.2 PROBLEM DEFINITION

As we know, fire spreads very vastly and will cause a large destruction and ultimately there will be a huge loss to rescue. In order to eliminate this kind of situation and to reduce the running rate, we have designed a light weight CNN model for fire detection which reduces the false alarm rates as well as overcomes the following listed problems in the traditional methods. a) Dead battery Certain smoke alarms are not hardwired into home electrical system rely on battery power. So, it require regular checking and not function when battery dead. b) Steam interference False alarm also will trigger when steam (maybe from bathroom or steam room) interrupt the light beams of electrical current inside smoke detector. c) Dusty deception High dusty area or insects can trip sensor inside alarm and make false alarm. d) Residual smoke from burned food (stoves, toasters or oven) also can trigger wrong alarm. e) Delay trigger alarm Available smoke sometime will take long delay although already detect smoke before trigger alarm to people.

CHAPTER 2

LITERATURE SURVEY

JOURNAL NAME AND YEAR: IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL.25, NO. 9,SEPTEMBER 2015.

AUTHOR NAME: Pasquale Foggia, Alessia saggese, and Mario Vento

PAPER TITLE: Real-Time Fire Detection for Video-Surveillance Applications Using a Combination of Experts Based On Color, Shape, and Motion

MERITS: The main advantage deriving from this approach lies in the fact that the overall performance of the system significantly increase with a relatively small effort made by the designer.

We considered dataset; at the same time, although having a less true positive further reduces the false positive rate, conquering the best rank on this aspect .

DEMERITS: The thresholding of potential fire pixels is not based on a simple heuristics but instead on a support vector machine, able to provide a good generalization without requiring problem domain knowledge.

Algorithm is less sensitive to variations in the luminance of the environment, its main drawback compared with other color based approaches.

JOURNAL NAME AND YEAR: SIBGRAPI Conference on Graphics, Patterns and Images, 2015 -IEEE

AUTHOR NAME: Daniel Y. T. chino, Letricia P. S. Avalhais, Jose F. Rodrigues Jr.,Agma J.M. Traina

PAPER TITLE: BoWFire: Detection of fire in Still Images by Integrating Pixel Color and Texture Analysis

MERITS: Flames exhibit random behavior, and it is experimentally observed that the underlying random process can be considered as a wide-sense stationary process for a small video region.

The method works very well when the fire is clearly visible and in close range, such that the flicker and irregular nature of flames are easily observable

DEMERITS: The system does not use a background subtraction method to segment moving regions and can be used, to some extent, with non-stationary cameras.

Using larger spatial block sizes decreases the resolution of the algorithm such that smaller fire regions cannot be detected.

JOURNAL NAME AND YEAR: IEEE Transactions on Systems, Man, and Cybernetics: Systems .2018

AUTHOR NAME: Khan Muhammad, Jamil Ahmad, Zhihan Lv, Paolo bellavista, Po yang, Sung Wook Baik.

PAPER TITLE: Efficient Deep CNN-Based Fire Detection and Localization in Video Surveillance Applications.

MERITS: Various abnormal events such as accidents, medical emergencies, and fires can be detected using these smart cameras.

Understanding the objects and scenes under observation. Future studies may focus on making challenging and specific scene understanding datasets for fire detection methods and detailed experiments.

DEMERITS: These sensors are not well suited to critical environments and need human involvement to confirm a fire in the case of an alarm, involving a visit to the location of the fire.

Such systems cannot usually provide information about the size, location, and burning degree of the fire.

JOURNAL NAME AND YEAR: IEEE - 2016

AUTHOR NAME: Sebastien Frizzi Rabeb Kaabi Moez Bouchouicha Jean-Marc Ginoux Eric Moreau Farhat Fnaiech

PAPER TITLE: Convolutional Neural Network for Video Fire and Smoke Detection

MERITS: The confusion matrix and ROC curves indicates a very good overall accuracy for the detection stage.

Convolutional neural network are shown to perform very well in the area of object classification.

DEMERITS: These features are difficult to define and depend largely on the kind of fire observed.

A different approach for this problem is to use a learning algorithm to extract the useful features instead of using an expert to build them

JOURNAL NAME AND YEAR: International Journal of Recent Technology and Engineering (IJRTE) December 2018

AUTHOR NAME: Senthil Vadivu, M.N. Vijayalakshmi

PAPER TITLE: An Efficient Multi-Feature Best Decision Based Forest Fire Detection (MF-BD-FFD) From Still Images

MERITS: Compared with two other methods in the literature and the performance improvement of the proposed method both in the correct fire detection rate and false alarm rate is demonstrated.

The system is commercially used in parallel to standard smoke detectors to reduce the false alarms caused by the smoke detectors.

DEMERITS: The statistical image features are not considered to be used as part of a standalone fire detection system.

The illumination of image changes, the fire pixel classification rules cannot perform well.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

- * To detect fire, researchers have presented both traditional and learned representation based fire detection methods.
- * In literature, the traditional methods use either color or motion characteristics for fire detection.
- * For instance, used color features for fire detection by exploring different color models including HSI, YUV, YCbCr, RGB, and YUC.
- * The major issue with these methods is their high rate of false alarms.
- * Several attempts have been made to solve this issue by combining the color information with motion and analyses of fire's shape and other characteristics

3.2 PROPOSED SYSTEM

- * The project provides an efficient RCNN based system for fire detection in videos captured in uncertain surveillance scenarios.
- * Our approach uses light-weight deep neural networks with no dense fully connected layers, making it computationally inexpensive.
- * Once detect a fire the information will pass through the firebase. Firebase is a one type of database.
- * Then the firebase to send a notification in android smartphone.

3.3 REQUIRIMENT ANALYSIS AND SPECIFICATION

3.3.1 INPUT REQUIRIMENT

The input requirements are Web cam, processor-I3 core and hard disk consist of 500 GB then the RAM should be 4 GB and higher version of the above mentioned requirements are needed to be used.

3.3.2 OUTPUT REQUIRIMENT

A computer pc/laptop and a android phone to use the android app are needed to get the fire detected notification.

3.3.3 FUNCTIONAL REQUIRIMENT

- User/client should have a android phone with latest version of android .To get a notification message in the android app.
- The webcam installed for capturing the video have to be working in good condition.

3.4 TECHNOLOGY STACK

3.4.1 HARDWARE REQUIREMENTS

- ❖ Hard Disk : 500GB and Above
- ❖ RAM : 4GB and Above
- ❖ Processor : I3 and Above
- ❖ Webcam 1

3.4.2 SOFTWARE REQUIREMENTS

- ❖ Operating System : Windows 7 , 8, 10 (64 bit)
- ❖ Software : Python 3.7
- ❖ Tools : Anaconda (Jupyter Note Book IDE)

Introduction to Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Python Syntax compared to other programming languages

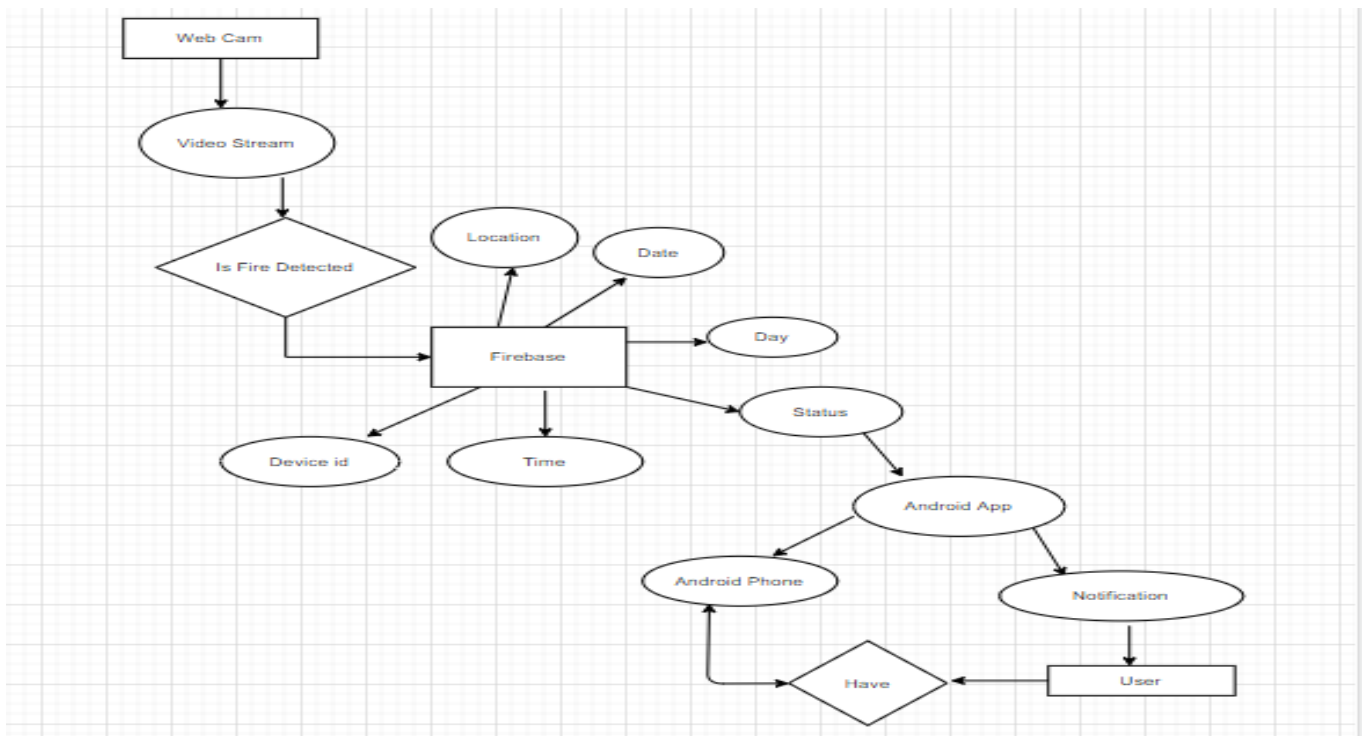
- Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

CHAPTER 4

SYSTEM DESIGN

4.1 ER DIAGRAM

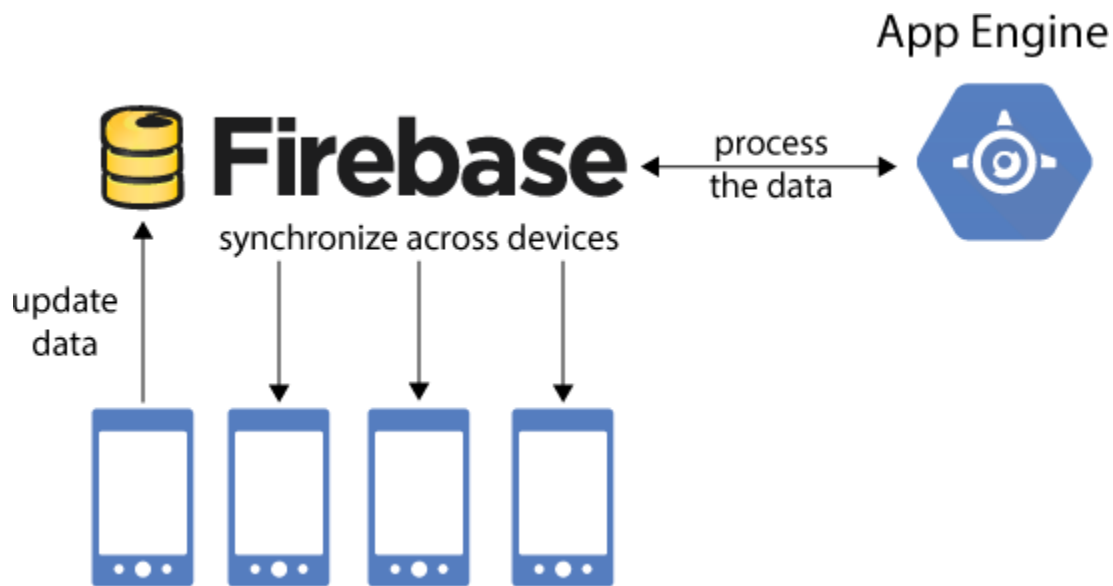
Entity Relationship Diagram (ERDs) illustrate the logical structure of databases. An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database.



4.2 DATA DICTIONARY

The Firebase is a Real-time Database it is a cloud-hosted , No-SQL database that allows to store and sync between your users in real-time.

The Real-time Database is really just one big JSON object that the developers can manage in real-time.



Cloud Storage for Firebase lets you upload and share user generated content, such as images and video, which allows you to build rich media content into your apps. Your data is stored in a Google Cloud Storage bucket — an exabyte scale object storage solution with high availability and global redundancy.

It is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality

4.3 UML DIAGRAMS

4.3.1 Use Case Diagram:

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify,

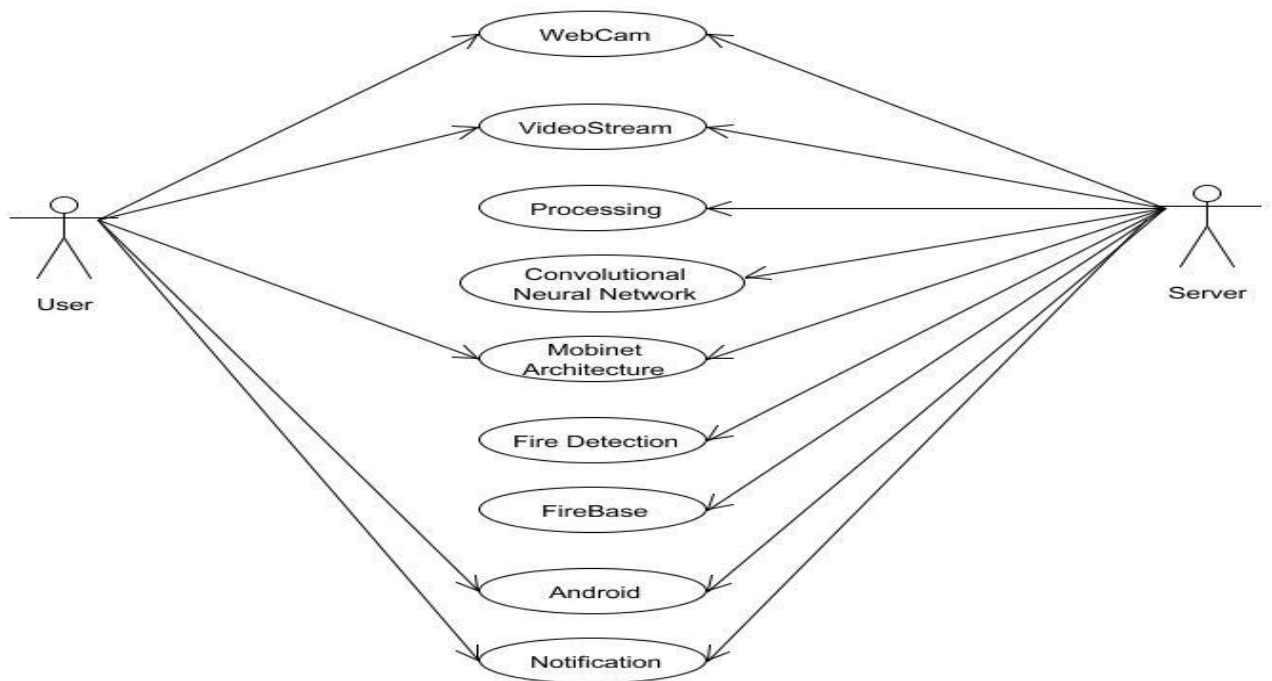
visualize, modify, construct and document the artifacts of an object oriented software intensive system under development.

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

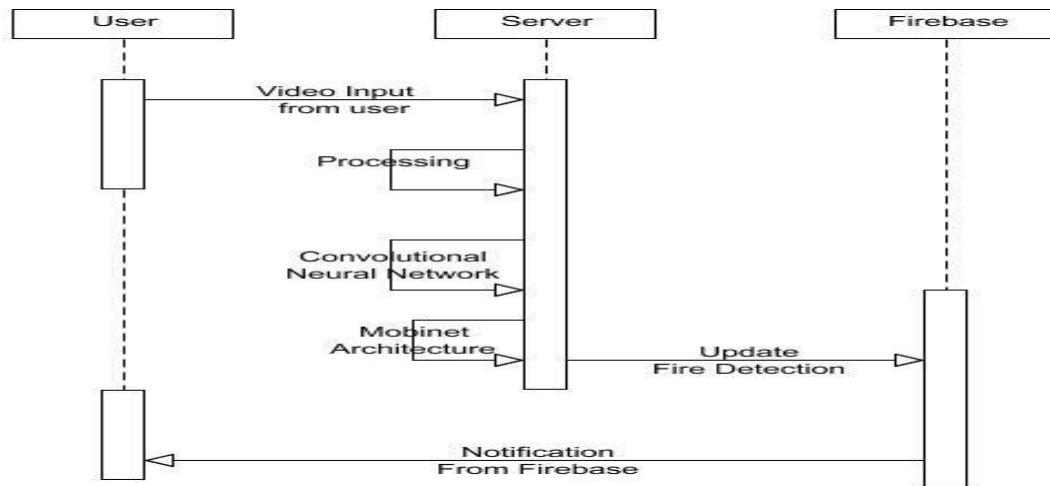
Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse.

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.



4.3.2 Sequence Diagram:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.

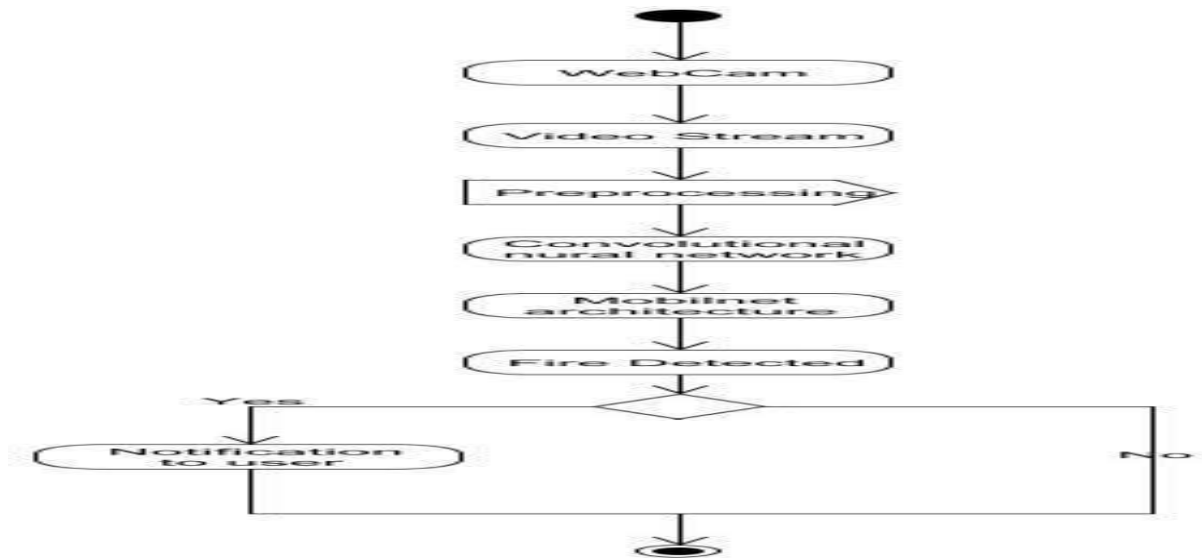


4.3.3 Activity Diagram:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

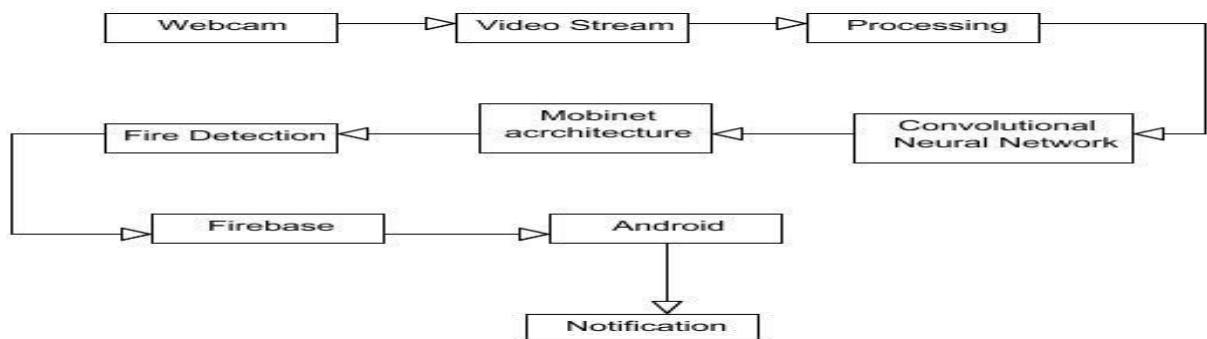
The most important shape types:

- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.



4.3.4 Collaboration Diagram:

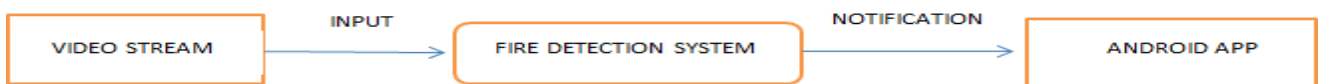
UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects



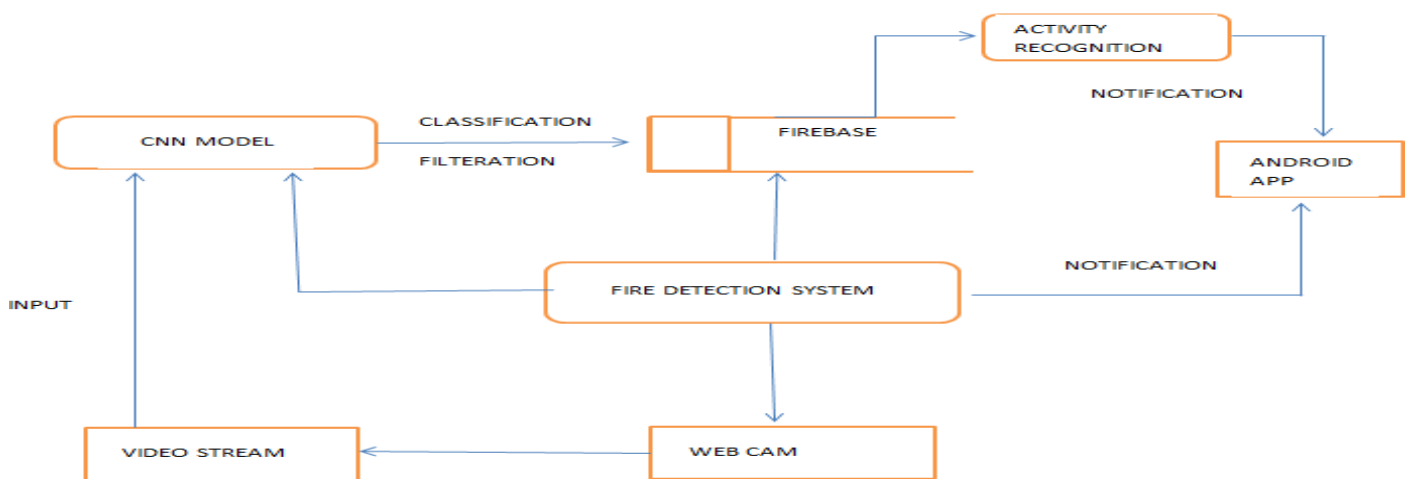
4.3.5 Data flow diagram:

A data-flow diagram is a way of representing a flow of data through a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart.

LEVEL 0:



LEVEL 1:

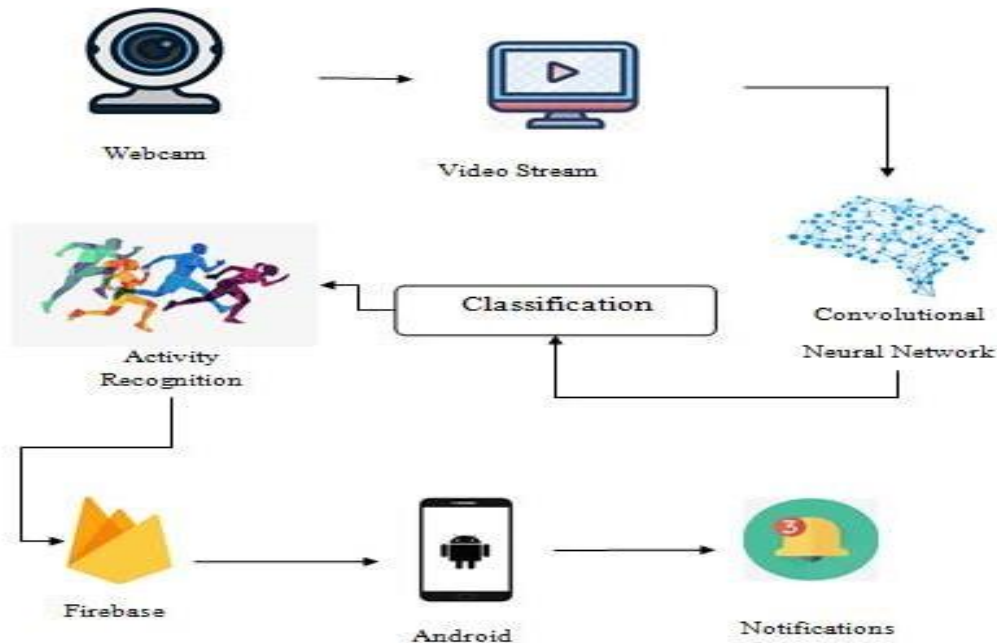


CHAPTER 5

SYSTEM ARCHITECTURE

5.1 Architecture Overview:

Architecture Diagram:



Details of the Proposed Architecture for Fire Detection

Firstly, video stream is given as a input to the CNN algorithm , here frame by frame detection takes places and also it classifies and identifies the object ,the recognition will be send to firebase, values for fire and non fire are already initiated .

After the detection from the firebase, notifications will be sent through the created Android App.

MobileNet (v2) version better than other models such as Alex Net, Google Net and Squeeze Net. Thus, we use a model with similar architecture to MobileNet and modify it according to fire detection problem in uncertain surveillance environment. To this end, we kept the number of neurons to two instead of 1000 in the final layer of our architecture, enabling classification into fire and non-fire.

5.2 Module Design Specification:

- * **DATASET CREATION**
- * **FRAME BY FRAME DETECTION**
- * **INPUT IDENTIFICATION**
- * **DATABASE CREATION**
- * **ANDROID APP FOR NOTIFICATION**

DATASET CREATION:

To Collect images of fire and non- fire images and train both the fire and non-fire images separately. Now there created a trained dataset with well determined fire images which compares with the video sequence.

FRAME BY FRAME DETECTION

The captured images are splitted into frame by frame and the frame is again splitted into pixels with the help of cnn algorithm from the surveillance

INPUT IDENTIFICATION:

- Now a input data is given to the CNN model which identifies whether the input is fire or non -fire.
- Then indicates a red warning in the background for few seconds ,if the input is fire.
- Else for a non-fire it indicates a green in the background

DATABASE CREATION:

FIREBASE:

- The Firebase Real-time Database is a cloud-hosted No-SQL database that allows to store and sync data between the users in real-time
- Using a gmail account , signed up into a firebase and added our project
- If the CNN model detects fire from the input data then, firebase will get updated.
- It clearly shows the date , day , time ,location once the fire detected and also the status as “FIRE”

ANDROID APP FOR NOTIFICATION

- The android app is created to get a notification if the fire is detected.
- The Android App “Fire detection” can be installed in users Mobile devices using APK file.
- Once fire is detected, firebase gets updated and the status will be send to the app.
- The created Android app will give a notification message as “FIRE DETECTED”

5.3 Program Design Language:

Algorithm used:

Convolutional Neural Networks (CNN)

Introduction

Convolutional neural networks (CNN) sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year’s ImageNet

competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

However, the classic, and arguably most popular, use case of these networks is for image processing. Within image processing, let's take a look at how to use these CNNs for image classification.

The Problem Space

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.



What We See

```

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 81 08
49 49 99 40 17 81 18 57 40 87 17 40 98 43 49 48 04 56 42 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 43 08 40 91 66 49 94 21
24 55 58 05 66 73 99 24 97 17 78 78 96 83 14 88 34 89 43 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 48 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 48 87 57 62 20 72 03 46 33 67 46 55 12 22 63 93 53 49
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 49 36 41 72 30 23 88 34 42 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 97 05 54
01 70 54 71 83 51 54 49 16 92 33 48 61 43 52 01 89 19 47 48

```

What Computers See

Inputs and Outputs

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describe the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for bird, etc).

What We Want the Computer to do

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process that goes on in our minds

subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does. Let's get into the specifics.

Biological Connection

But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 (Video) where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs.

Structure

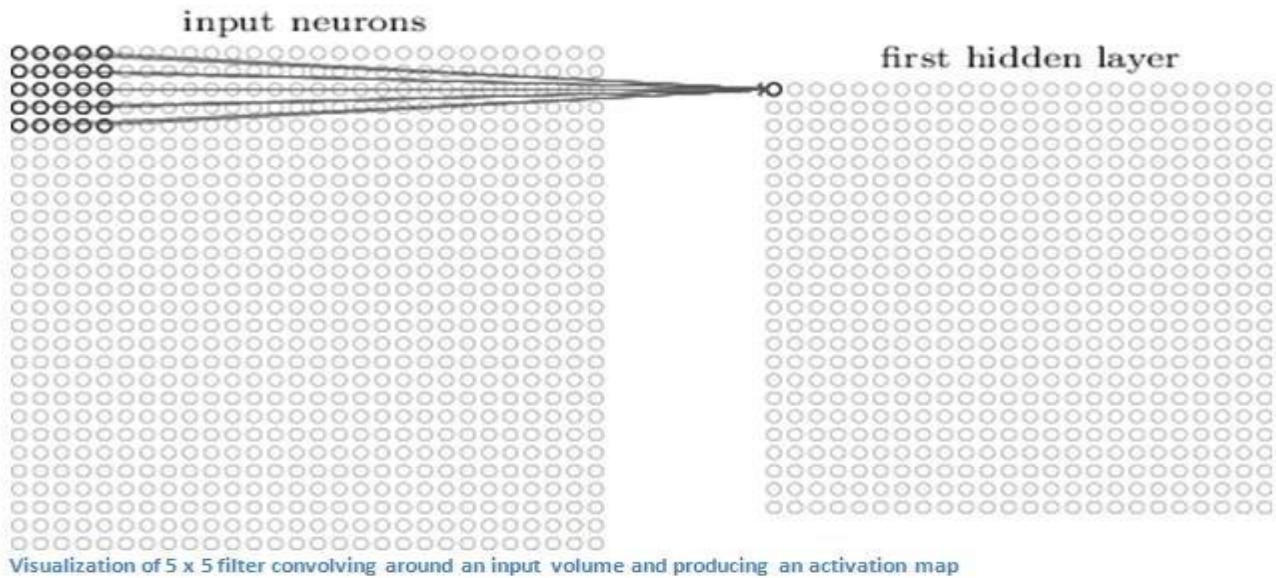
Back to the specifics. A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now,

the hard part understands what each of these layers do. So let's get into the most important one.

First Layer – Math Part

The first layer in a CNN is always a Convolutional Layer. First thing to make sure you remember is what the input to this conv (I'll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a $32 \times 32 \times 3$ array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5×5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a filter (or sometimes referred to as a neuron or a kernel) and the region that it is shining over is called the receptive field. Now this filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimension of this filter is $5 \times 5 \times 3$. Now, let's take the first position the filter is in for example. It would be the top left corner. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing element wise multiplications). These multiplications are all summed up (mathematically speaking, this would be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you're left with is a $28 \times 28 \times 1$ array of numbers, which we call an activation map or feature map. The reason you get a 28×28 array is that there are 784 different locations that a 5×5 filter

can fit on a 32 x 32 input image. These 784 numbers are mapped to a 28 x 28 array.



Let's say now we use two 5 x 5 x 3 filters instead of one. Then our output volume would be 28 x 28 x 2. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

First Layer – High Level Perspective

However, let's talk about what this convolution is actually doing from a high level. Each of these filters can be thought of as **feature identifiers**. When I say features, I'm talking about things like straight edges, simple colors, and curves. Think about the simplest characteristics that all images have in common with each other. Let's say our first filter is 7 x 7 x 3 and is going to be a curve detector. (In this section, let's ignore the fact that the filter is 3 units deep and only consider the top depth slice of the filter and the image, for simplicity.) As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve (Remember, these filters that we're talking about as just numbers!).

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

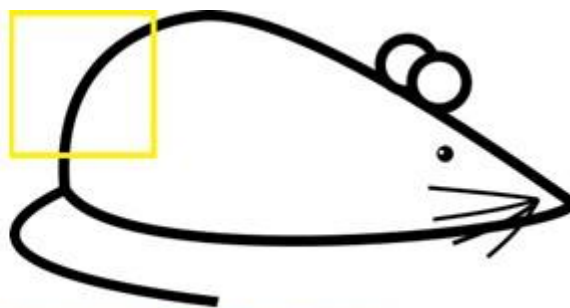


Visualization of a curve detector filter

Now, let's go back to visualizing this mathematically. When we have this filter at the top left corner of the input volume, it is computing multiplications between the filter and pixel values at that region. Now let's take an example of an image that we want to classify, and let's put our filter at the top left corner.



Original image



Visualization of the filter on the image

Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$ (A large number!)

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

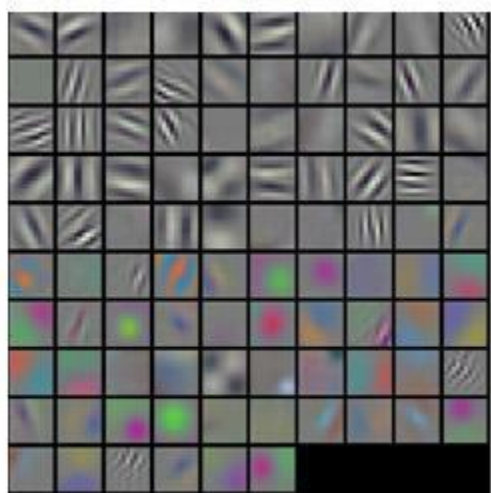
Multiplication and Summation = 0

The value is much lower! This is because there wasn't anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas in which there are mostly likely to be curves in the picture. In this example, the top left value of our 26 x 26 x 1 activation map (26 because of the 7x7 filter instead of 5x5) will be 6600. This high value means that it is likely that there is some sort of curve in the input volume that caused the

filter to activate. The top right value in our activation map will be 0 because there wasn't anything in the input volume that caused the filter to activate (or more simply said, there wasn't a curve in that region of the original image). Remember, this is just for one filter. This is just a filter that is going to detect lines that curve outward and to the right. We can have other filters for lines that curve to the left or for straight edges. The more filters, the greater the depth of the activation map, and the more information we have about the input volume.

Disclaimer:

The filter I described in this section was simplistic for the main purpose of describing the math that goes on during a convolution. In the picture below, you'll see some examples of actual visualizations of the filters of the first conv layer of a trained network. Nonetheless, the main argument remains the same. The filters on the first layer convolve around the input image and "activate" (or compute high values) when the specific feature it is looking for is in the input volume.



Visualizations of filters

Going Deeper Through the Network

Now in traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I'd strongly encourage those interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting. A classic CNN architecture would look like this.

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool -> Fully Connected

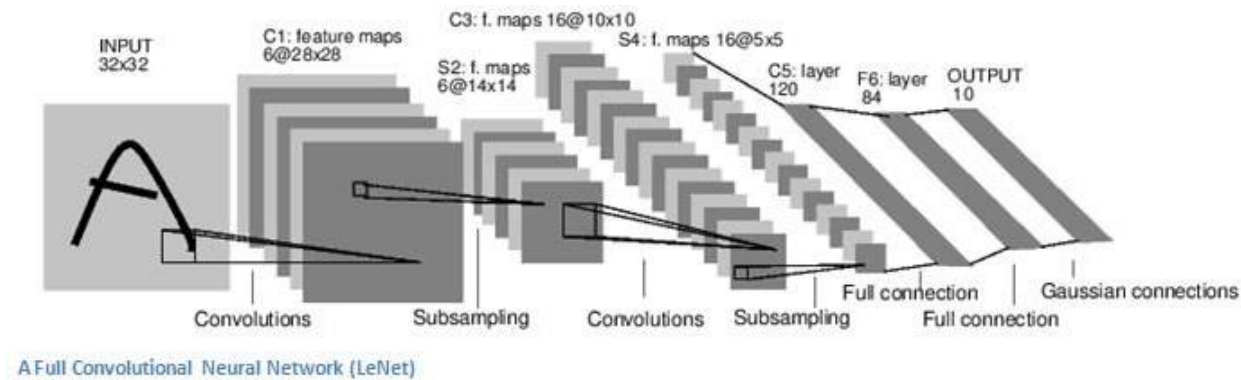
The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would imagine, in order predicting whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So let's think about what the output of the network is after the first conv layer. It would be a $28 \times 28 \times 3$ volume (assuming we use three $5 \times 5 \times 3$ filters). When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer. Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the output will be activations that represent higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have

some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. If you want more information about visualizing filters in ConvNets, Matt Zeiler and Rob Fergus had an excellent research paper discussing the topic. Jason Yosinski also has a video on YouTube that provides a great visual representation. Another interesting thing to note is that as you go deeper into the network, the filters begin to have a larger and larger receptive field, which means that they are able to consider information from a larger area of the original input volume (another way of putting it is that they are more responsive to a larger region of pixel space).

Fully Connected Layer

Now that we can detect these high level features, the icing on the cake is attaching a **fully connected layer** to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. For example, if the resulting vector for a digit classification program is [0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but I am just showing the softmax approach). The way this fully connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class. For example, if the program is predicting that some image is a dog, it will have high values in the activation maps that represent high level features like a paw or 4 legs, etc. Similarly, if the program is predicting that some image is a bird, it will have high values in the activation maps that

represent high level features like wings or a beak, etc. Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.



Training (AKA: What Makes this Stuff Work)

Now, this is the one aspect of neural networks that I purposely haven't mentioned yet and it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know what activation maps to look at? How do the filters in each layer know what values to have? The way the computer is able to adjust its filter values (or weights) is through a training process called **backpropagation**.

Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values are randomized. The filters don't know to look for edges and curves. The filters in the higher layers don't know to look for paws and beaks. As we grew older however, our parents and teachers showed us different

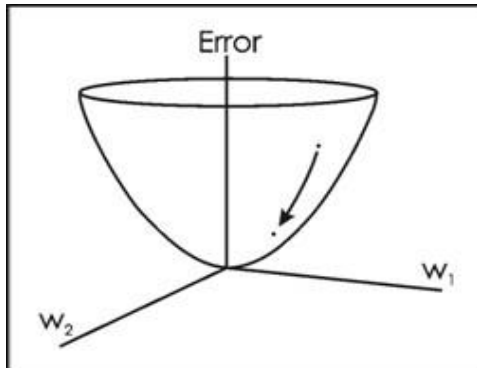
pictures and images and gave us a corresponding label. This idea of being given an image and a label is the training process that CNNs go through. Before getting too into it, let's just say that we have a training set that has thousands of images of dogs, cats, and birds and each of the images has a label of what animal that picture is. Back to backprop.

So backpropagation can be separated into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the **forward pass**, you take a training image which as we remember is a 32 x 32 x 3 array of numbers and pass it through the whole network. On our first training example, since all of the weights or filter values were randomly initialized, the output will probably be something like [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1], basically an output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be. This goes to the **loss function** part of backpropagation. Remember that what we are using right now is training data. This data has both an image and a label. Let's say for example that the first training image inputted was a 3. The label for the image would be [0 0 0 1 0 0 0 0 0 0]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is $\frac{1}{2}$ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Let's say the variable L is equal to that value. As you can imagine, the loss will be extremely high for the first couple of training images. Now, let's just think about this intuitively. We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label (This means that our network got its prediction right). In order to get there, we want to minimize the amount of loss we have. Visualizing this as just an optimization problem in calculus, we want to find out which inputs (weights in

our case) most directly contributed to the loss (or error) of the network.



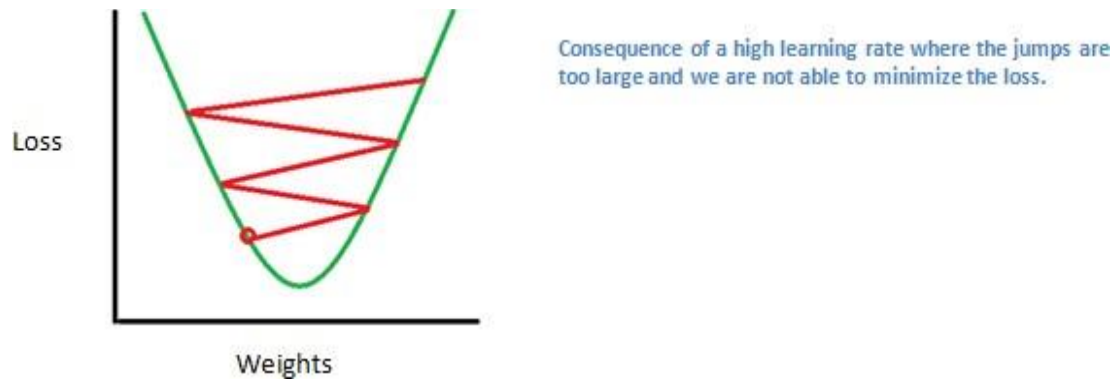
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

This is the mathematical equivalent of a $\mathbf{dL/dW}$ where W are the weights at a particular layer. Now, what we want to do is perform a **backward pass** through the network, which is determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the **weight update**. This is where we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
 w_i = Initial Weight
 η = Learning Rate

The **learning rate** is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.



The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch). Once you finish the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

Testing

Finally, to see whether or not our CNN works, we have a different set of images and labels (can't double dip between training and test!) and pass the images through the CNN. We compare the outputs to the ground truth and see if our network works!

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 CODING

firenet.py

```
import cv2
import os
import sys
import math
import datetime
import time

from firebase import firebase
import tflearn
from tflearn.layers.core import *
from tflearn.layers.conv import *
from tflearn.layers.normalization import *
from tflearn.layers.estimator import regression
fixefixed_interval = 3
firebase = firebase.FirebaseApplication('https://fire-85db0-default-rtdb.firebaseio.com/',
None)
count=1
def construct_firenet (x,y, training=False):

    # Build network as per architecture in [Dunnings/Breckon, 2018]

    network = tflearn.input_data(shape=[None, y, x, 3], dtype=tf.float32)

    network = conv_2d(network, 64, 5, strides=4, activation='relu')

    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)

    network = conv_2d(network, 128, 4, activation='relu')
```

```

network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)

network = conv_2d(network, 256, 1, activation='relu')

network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)

network = fully_connected(network, 4096, activation='tanh')
network = dropout(network, 0.5)

network = fully_connected(network, 4096, activation='tanh')
network = dropout(network, 0.5)

network = fully_connected(network, 2, activation='softmax')

# if training then add training hyperparameters

if(training):
    network = regression(network, optimizer='momentum',
                        loss='categorical_crossentropy',
                        learning_rate=0.001)

# constuct final model

model = tflearn.DNN(network, checkpoint_path='firenet',
                    max_checkpoints=1, tensorboard_verbose=2)

return model

```

```

if __name__ == '__main__':

```

```

    # construct and display model

```

```

model = construct_firenet (224, 224, training=False)
print("Constructed FireNet ...")

model.load(os.path.join("models/FireNet", "firenet"),weights_only=True)
print("Loaded CNN network weights ...")


# network input sizes

rows = 224
cols = 224


# display and loop settings

windowName = "Live Fire Detection - FireNet CNN";
keepProcessing = True;


if len(sys.argv) == 2:

    # load video file from first command line argument

    video = cv2.VideoCapture(sys.argv[1])
    print("Loaded video ...")

    # create window

    cv2.namedWindow(windowName, cv2.WINDOW_NORMAL);

    # get video properties

    width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH));
    height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))

```

```

fps = video.get(cv2.CAP_PROP_FPS)
frame_time = round(1000/fps);

while (keepProcessing):

    # start a timer (to see how long processing and display takes)

    start_t = cv2.getTickCount();

    # get video frame from file, handle end of file

    ret, frame = video.read()
    if not ret:
        print("... end of video file reached");
        break;

    # re-size image to network input size and perform prediction

    small_frame = cv2.resize(frame, (rows, cols), cv2.INTER_AREA)
    output = model.predict([small_frame])

    # label image based on prediction

    if round(output[0][0]) == 1:
        cv2.rectangle(frame, (0,0), (width,height), (0,0,255), 50)
        cv2.putText(frame,'FIRE',(int(width/16),int(height/4)),
            cv2.FONT_HERSHEY_SIMPLEX, 4,(255,255,255),10,cv2.LINE_AA);
        #print("Fire")
        if count==1:
            datetime1=datetime.datetime.now()
            date=datetime1.strftime("%x")
            time=datetime1.strftime("%X")
            day=datetime1.strftime("%A")
            device = "42"
            status="Fire"

```

```

        data={"Device
ID":device,"Status":status,"Date":date,"Time":time,"Day":day}
        firebase.put(", 'FIRE DETECTION/Location 1', data)
        #time.sleep(10)
        status="clear"
        data={"Device
ID":device,"Status":status,"Date":date,"Time":time,"Day":day}
        firebase.put(", 'FIRE DETECTION/Location 1', data)
        count=0

```

else:

```

cv2.rectangle(frame, (0,0), (width,height), (0,255,0), 50)
cv2.putText(frame,'CLEAR',(int(width/16),int(height/4)),
cv2.FONT_HERSHEY_SIMPLEX, 4,(255,255,255),10,cv2.LINE_AA);
#print("Clear")
if count==1:

```

```

    device = "42"
    status="Clear"
    data={"Device ID":device,"Status":status}
    firebase.put(", 'FIRE DETECTION/Location 1', data)
    count=0

```

stop the timer and convert to ms. (to see how long processing and display takes)

```
stop_t = ((cv2.getTickCount() - start_t)/cv2.getTickFrequency()) * 1000;
```

image display and key handling

```
cv2.imshow(windowName, frame);
```

```
# wait fps time or less depending on processing time taken (e.g. 1000ms / 25 fps =  
40 ms)
```

```
key = cv2.waitKey(max(2, frame_time - int(math.ceil(stop_t)))) & 0xFF;  
if (key == ord('x')):  
    keepProcessing = False;  
elif (key == ord('f')):  
    cv2.setWindowProperty(windowName, cv2.WND_PROP_FULLSCREEN,  
cv2.WINDOW_FULLSCREEN);  
else:  
    print("usage: python firenet.py videofile.ext");
```

video_processing.py

```
import cv2  
import os  
import sys  
import math
```

```
rows=224  
cols=224
```

```
keepProcessing=True  
windowName='fire'
```

```
video = cv2.VideoCapture('models/test.mp4')  
print("Loaded video ...")
```

```
cv2.namedWindow(windowName, cv2.WINDOW_NORMAL);
```

```
width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH));  
height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

```
fps = video.get(cv2.CAP_PROP_FPS)  
frame_time = round(1000/fps);
```

```

while (keepProcessing):

    start_t = cv2.getTickCount();
    ret, frame = video.read()
    if not ret:
        print("... end of video file reached");
        break;

    small_frame = cv2.resize(frame, (rows, cols), cv2.INTER_AREA)
    stop_t = ((cv2.getTickCount() - start_t)/cv2.getTickFrequency()) * 1000;
    cv2.imshow(windowName, frame);

    key = cv2.waitKey(max(2, frame_time - int(math.ceil(stop_t)))) & 0xFF;

    if (key == ord('x')):
        keepProcessing = False;

    elif (key == ord('f')):
        cv2.setWindowProperty(windowName, cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN);

    else:
        print("usage: python firenet.py videofile.ext");

```

CHAPTER 7

SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

7.1 UNIT TESTING

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

7.2 INTERGRATION TESTING

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as

early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

7.3 TEST CASES &REPORTS/PERFORMANCE ANALYSIS

TEST CASE 1

S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT	TEST COMMENTS
1.	Fire detection& notification	Test.mp4	Fire detection& notification to app	Fire is detected& notification is sent to the app	Pass	Status will indicate as Fire detected

TEST CASE 2

S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT	TEST COMMENTS
2.	Fire detection& notification	Test1.mp4	Fire detection & notification to app	No fire detected	Fail	Status will indicate as clear.

TEST CASE 3

S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT	TEST COMMENTS
3.	No-Fire detected	Test1.mp4	No-Fire is detected	No-Fire is detected	Pass	Status will indicate as clear

TEST CASE 4

S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT	TEST COMMENTS
4.	No-Fire detected	Test.mp4	No-Fire is detected	Fire is detected& Notification Is sent to the app	Fail	Status will indicate as Fire detected

CHAPTER 8

8.1 CONCLUSION AND FUTURE ENHANCESMENT

Fire is the most dangerous abnormal event, as failing to control it at an early stage can result in huge disasters leading to human, ecological and economic losses. Fire accidents can be detected using the cameras. So that, here we proposed a CNN approach for fire detection using cameras. Our approach can identify the fire under the camera surveillance. Furthermore, our proposed system balances the accuracy of fire detection and the size of the model using fine-tuning of datasets. We have obtained an accuracy of 94%. Also the F-measure value is 0.95. These values shows that the model gives a better prediction. We conduct experiments using datasets collected from recording of fire and verified it to our proposed system. In view of the CNN model's reasonable accuracy for fire detection, its size, and the rate of false alarms, the system can be helpful to disaster management teams in controlling fire disasters in a short time. Thus, avoiding huge losses. This work mainly focuses on the detection of fire scenes under observation. Future studies may focus on deploying the model into raspberry pi and using necessary support packages to detect the real time fire by making challenging and specific scene understanding datasets for fire detection methods and detailed experiments.

APPENDICES

A.1 SAMPLE SCREENS

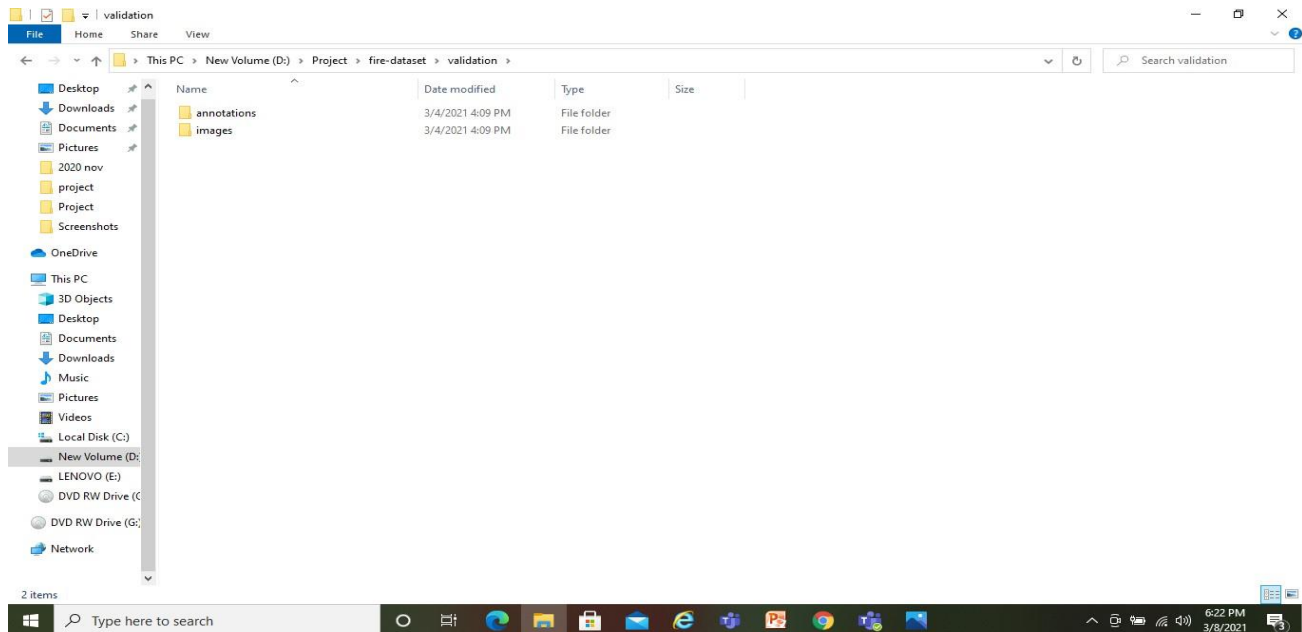


FIG A.1.1- FOLDERS FOR COLLECTED DATASETS

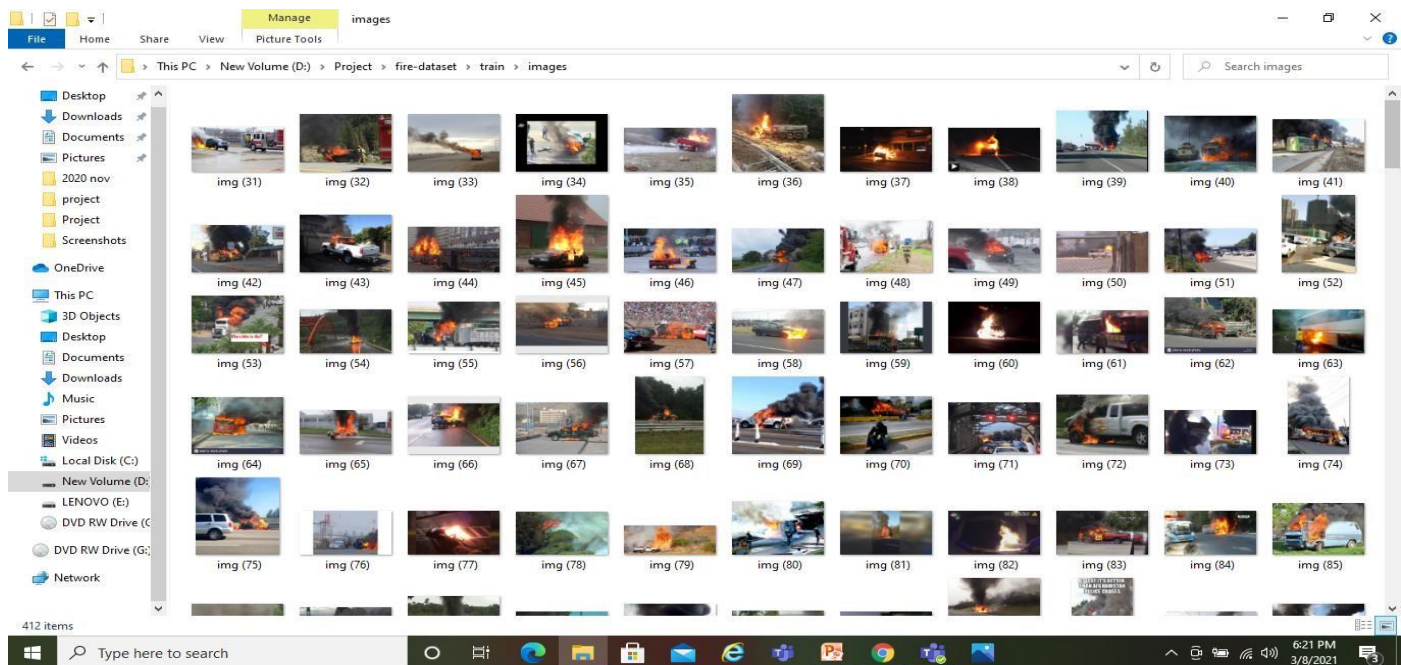


FIG A.1.2- FIRE AND NON-FIRE IMAGES FOR DATASETS

```
Anaconda Prompt (Anaconda3) - python firenet.py models/test.mp4

(base) C:\Users\Sowmiya>d:

(base) D:\>Project
'Project' is not recognized as an internal or external command,
operable program or batch file.

(base) D:\>cd Project

(base) D:\Project>python firenet.py models/test.mp4
WARNING:tensorflow:From C:\Users\Sowmiya\Anaconda3\lib\site-packages\tensorflow\python\compat\v2_compat.py:61: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
curses is not supported on this machine (please install/reinstall curses for an optimal experience)
WARNING:tensorflow:From C:\Users\Sowmiya\Anaconda3\lib\site-packages\tflearn\initializations.py:110: calling UniformUnitScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From C:\Users\Sowmiya\Anaconda3\lib\site-packages\tensorflow\python\util\deprecation.py:507: UniformUnitScaling.__init__ (from tensorflow.python.ops.init_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.initializers.variance_scaling instead with distribution-uniform to get equivalent behavior.
WARNING:tensorflow:From C:\Users\Sowmiya\Anaconda3\lib\site-packages\tflearn\initializations.py:165: calling TruncatedNormal.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From C:\Users\Sowmiya\Anaconda3\lib\site-packages\tensorflow\python\training\saver.py:247: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use 'rate' instead of 'keep_prob'. Rate should be set to 'rate = 1 - keep_prob'.
2021-03-08 18:30:04.377308: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2021-03-08 18:30:06.294864: W tensorflow/core/framework/allocation.cc:107] Allocation of 205520896 exceeds 10% of system memory.
Constructed FireNet ...
2021-03-08 18:30:08.556080: W tensorflow/core/framework/allocation.cc:107] Allocation of 205520896 exceeds 10% of system memory.
WARNING:tensorflow:From C:\Users\Sowmiya\Anaconda3\lib\site-packages\tensorflow\python\training\saver.py:1276: checkpoint_exists (from tensorflow.python.training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
2021-03-08 18:30:10.757362: W tensorflow/core/framework/allocation.cc:107] Allocation of 205520896 exceeds 10% of system memory.
Loaded CNN network weights ...
Loaded video ...
```

FIG A.1.3- SCREENSHOT OF ANACONDA PROMPT



FIG A.1.4- FIRE DETECTION



FIG A.1.5-NON-FIRE DETECTION

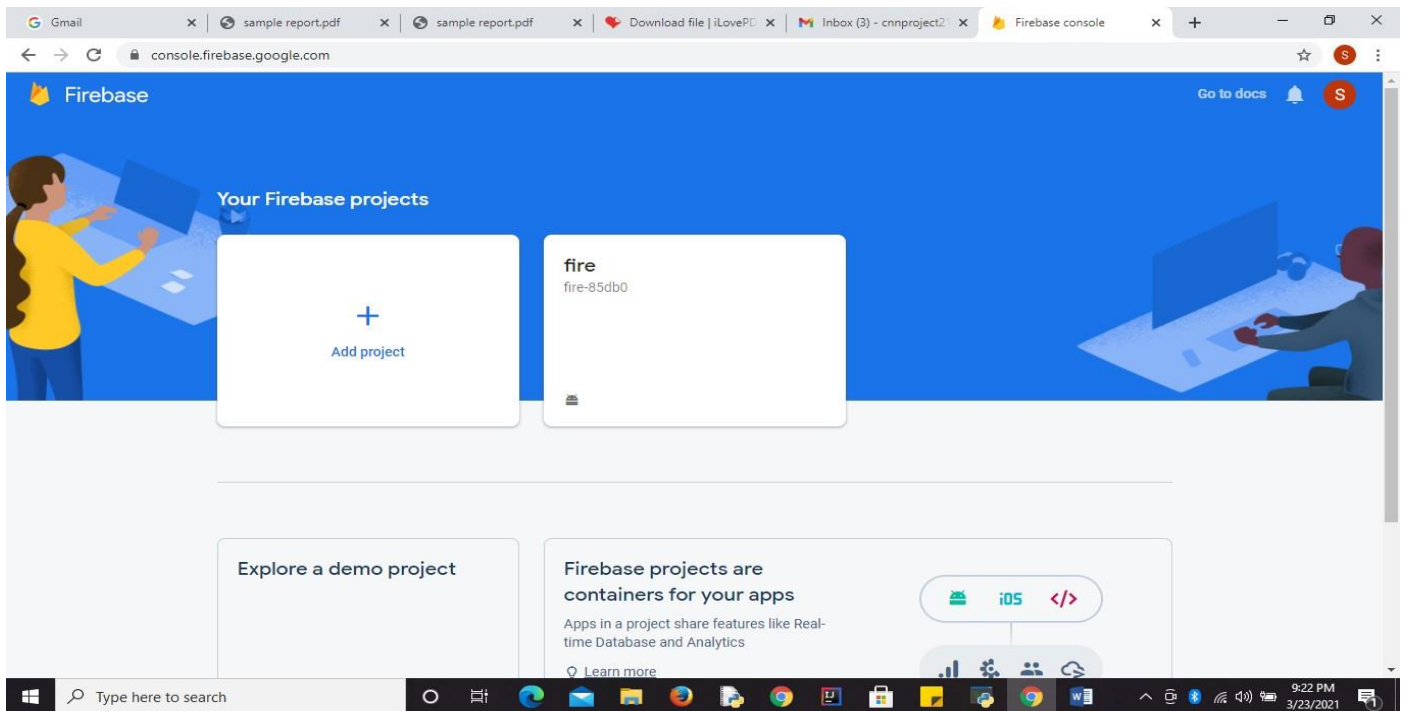


FIG A.1.6-HOMEPAGE OF FIREBASE CONSOLE

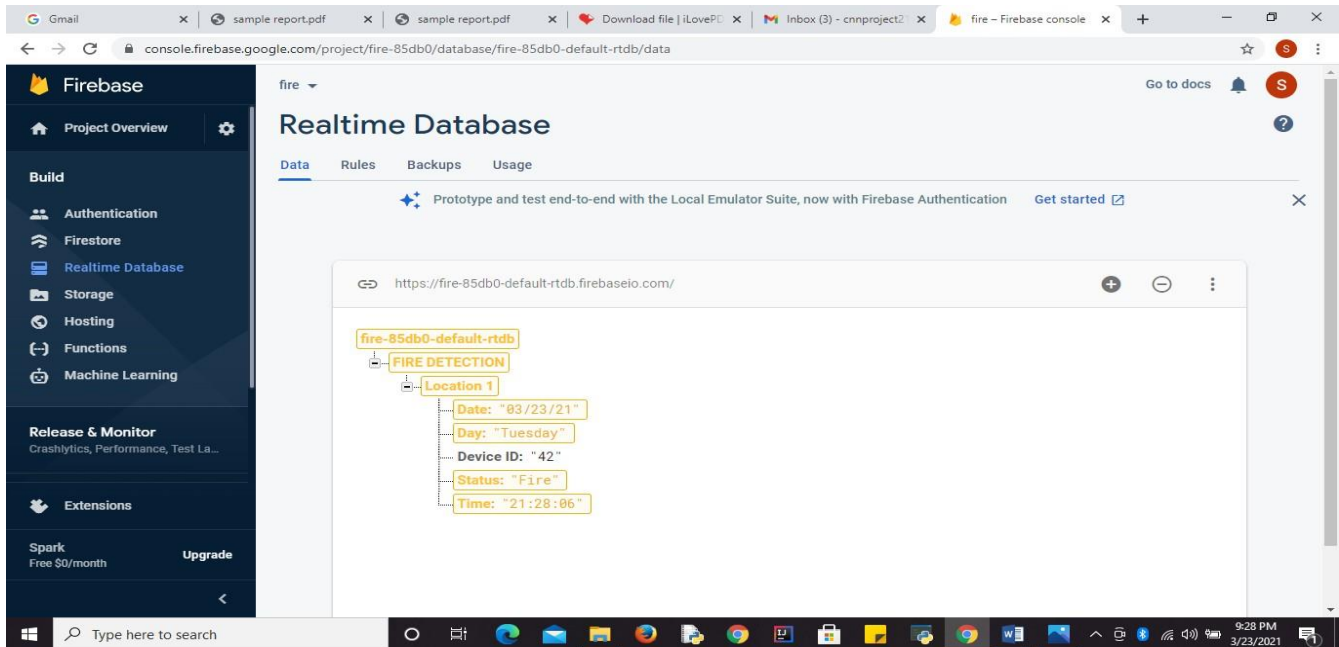


FIG A.1.7-FIREBASE SHOWING THE UPDATED STATUS AS “FIRE”

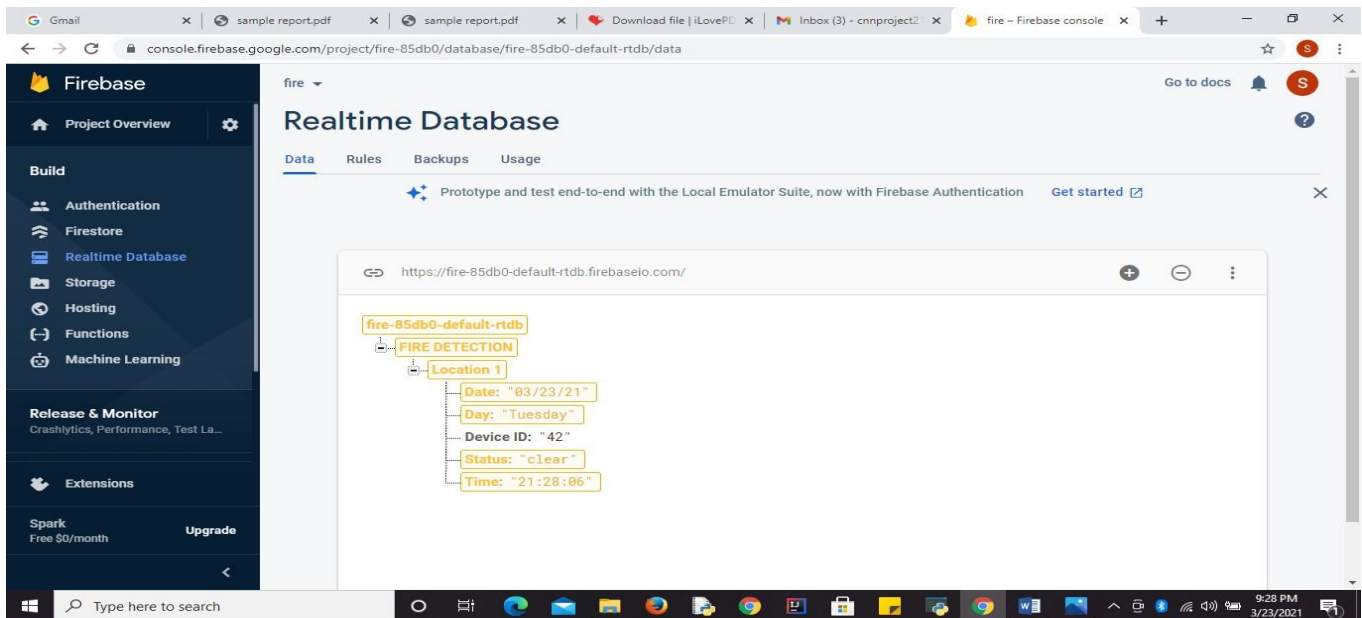
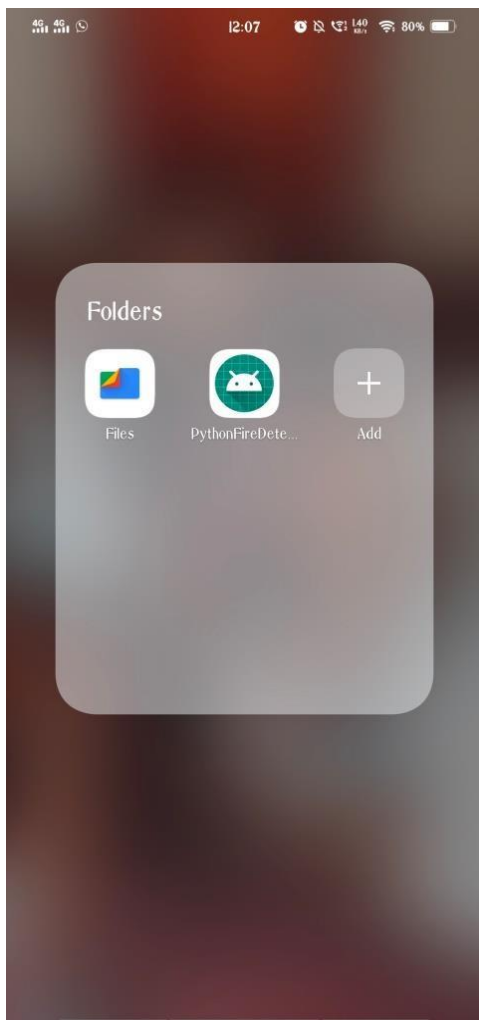
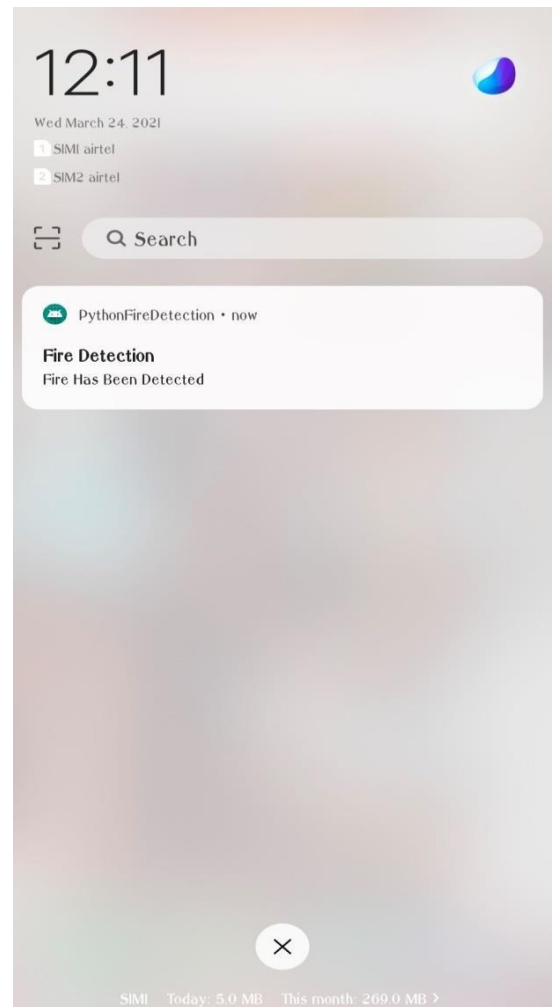


FIG A.1.8-FIREBASE SHOWING THE UPDATED STATUS AS “NON-FIRE”



**FIG A.1.9-ANDROID APP
FOR FIRE DETECTION**



**FIG A.1.10-NOTIFICATION FROM
APP AS FIRE DETECTED**

A2.PUBLICATION

DESIGN AND IMPLEMENTATION OF A LIGHT WEIGHT CNN MODEL FOR FIRE DETECTION

P.DEEPA¹, B.SUSHMITHA², S. SOWMIYA³

Associate Professor¹, UG Scholar^{2,3}

^{1,2,3}Dept. of CSE, Panimalar Engineering College, Chennai.

mca_deepa@yahoo.com¹, sushmi1024@gmail.com²

Abstract

When compared to traditional fire recognition frameworks based on sensors, vision-based fire detection frameworks have recently gained popularity. The need for video perception in private, modern, business areas, and wooded areas has expanded the application of a vision-based fire system. Recently, a number of fire-related accidents have occurred as a result of inadequate surveillance or the inability to cover those uncertain regions such as restricted areas in forests or factory buildings. To avoid such mishaps, we propose a method based on Convolution neural networks.

Keywords – Fire detection, Image classification.CNN, Video analysis.

In 2018, 1,318,500 fires have been reported by the NFPA report, resulting in a total of 3,655 civilian fire killings, 15,200 civilian fire wounds, and a direct loss of \$25.6 billion. A fire victim was reported every 2 hours and 24 minutes and a fire injury was reported every 35 minutes. Domestic fires killed 2,720 people, making up close to 74% of all deaths. Of the 36,746,500 calls, 4 fires were reported, 8% were reported to be false alarms. Modern smoke detectors are mostly ionizing, photoelectric or laser based. In an outdoor environment, however, they are not very reliable. The availability of software open source helps in the quick delivery of CNN deployments.

1. INTRODUCTION

The first to recognize checking figures were evolutionary neural networks and deep learning in the late 1990s. In the last decade, on the other hand, CNNs have been widely used. The primary reason for this is the increase and decrease in the performance of GPUs. Initially, GPUs were designed to perform countless millions of matrix operations per second. As a large number of matrix operations are required for neural networks. GPUs are better optimized than central processing units for neural networks (CPUs). We chose just to detect fires and smoke for this article, because early detection is one of the most common cases in which the number of victims is reduced and fire-related disasters prevented.

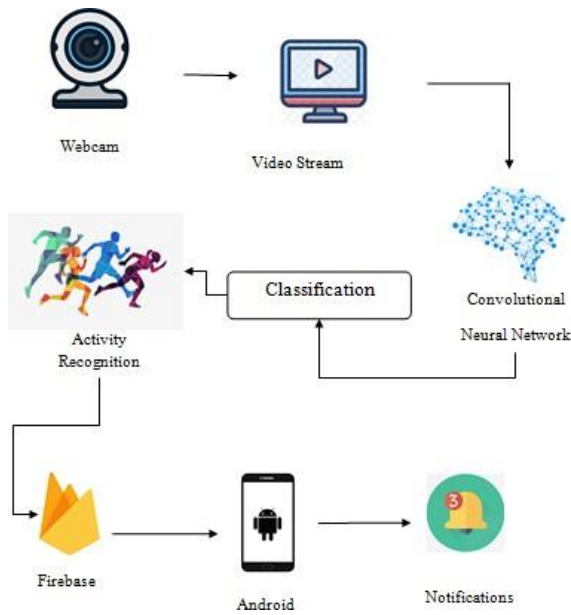
2. EXISTING SYSTEM

Researchers introduced traditional as well as learned fire detection methods for the detection of fire. In literature, both color and motion characteristics are used in traditional methods for fire detection. Use color detection features such as HSI, YUV, YCbCr, RGB, and YUC to explore different color models for Fire detection. High levels of false alarms are the major problem with these methods. Several efforts were made to solve this problem by combining color information with motion, fire form analyses and other features.

3. PROPOSED SYSTEM

The work proposed offers a cost-effective RCNN-based fire detection system for videos captured in uncertain scenarios. Our approach uses lightweight, deep neural networks with no dense completely connected layers, which makes them computer-costly. The information passes through the firebase once a fire is detected. Firebase's a database type. The firebase then sends a message to an android smart phone.

4. SYSTEM ARCHITECTURE



5. MODULE DESCRIPTION

The proposed system consists of the following modules

- ❖ Dataset Creation
- ❖ Frame By Frame Detection
- ❖ Input Identification

5.1 CREATION OF DATASET

Collect fire and non fire images and train fire images separately as well as non-fire images. There is now a trained dataset, compared to the video sequence and with well defined fire pictures.

5.2 DETECTION OF FRAMES

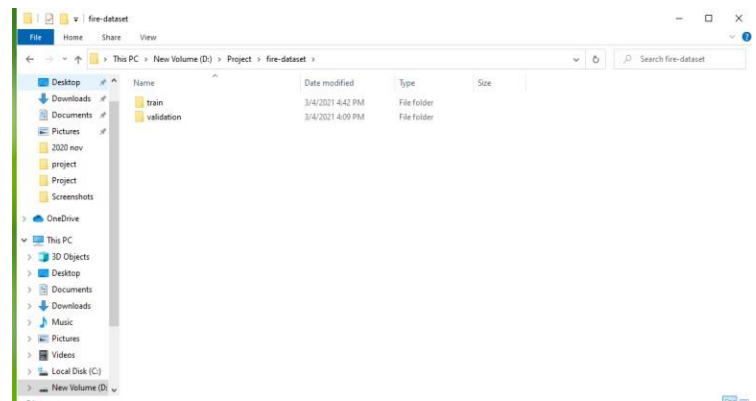
The images are divided into frames and the frame is again divided into pixels by means of the CNN monitoring algorithm.

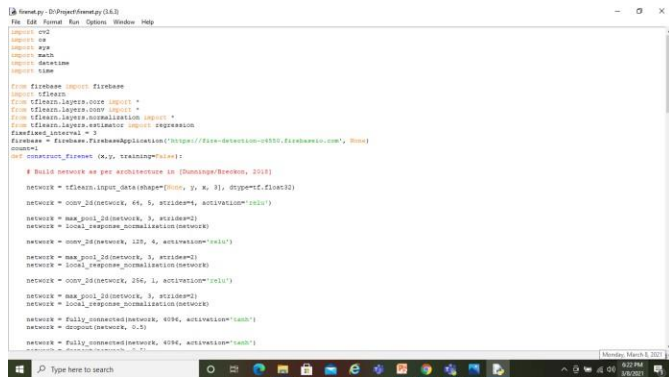
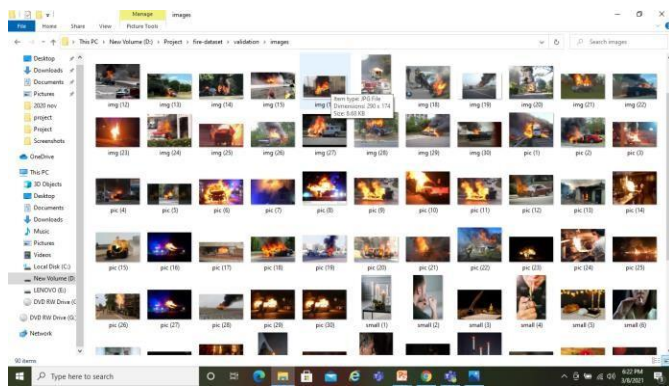
5.3 IDENTIFICATION OF INPUTS

A CNN model is now provided with data indicating whether or not the input is fire. In the background a red warning will be given if the input is a fire for a few seconds. In the background it shows a green otherwise for a non fire.

5. SYSTEM IMPLEMENTATION

The work has been implemented in python 3.7, operating system used is windows 10 (64bits) and tool is Anaconda (JUPYTER IDE).





6. CONCLUSION

In this paper, we present and compare our fire detection network with existing CNNs. The CNNs were compared using two different data sets for the training of CNNs for fire detection from the images.

A lightweight CNN model with high precision results was designed. After training, the fire detection network model was smaller. Two data sets were used to measure the accuracy of the model fire detection network and found to be between 77 and 81 percent. In terms of run time the best time for each image processing on both data sets was shown by our fire detection network model. The measured fire detection network time for the dataset1 reached 0,052s per picture. DataSet2 showed a 0.109s per picture fire-detection network model. We have also shown potential future development steps for fire detection systems in this paper.

REFERENCES

- [1] R. Raina , A. Madhavan and A. Y. Ng, "Largescale deep un-supervised learning using graphics processors,"Computer Science Department, Stanford University, Stanford CA 94305 USA.
- [2] K. Muhammad, R. Hamza, J. Ahmad, J. Lloret, H. H. Ge Wang, and S. W. Baik, "Secure surveillance framework for IoT systems using probabilistic image encryption,"IEEE Trans. Ind. Inform., to be published.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Con-volutional Neural Networks,"InNIPS, 2012.
- [4] Y. LeCun et al., "Handwritten digit recognition with a back-propagation network," inProc. Adv. Neural Inf. Process. Syst., 1990, pp.396–404
- [5] A. Ullah et al., "Action recognition in movie scenes using deep features of key frames,"J. Korean Inst. Next Generation Comput., vol. 13, pp. 7–14, 2017.
- [6] R. Zhang, J. Shen, F. Wei, X. Li, and A. K. Sangaiah, "Medical image classification based on multi-scale non-negative sparse cod-ing,"Artif.Intell. Med., vol. 83, pp. 44–51, Nov. 2017.
- [7] F. N. Iandola1, S. Han2, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters

and; 0.5 mb model size,”arXiv preprint arXiv:1602.07360 (2016).

[8] GitHub-DeepQuestAI/Fire-Smoke-Dataset: An image dataset for training fire and frame detection AI, [Online]. <https://github.com/DeepQuestAI/Fire-Smoke-Dataset>.

[9] GitHub-cair/Fire-Detection-Image-Dataset, [Online]. <https://github.com/cair/Fire-Detection-Image-Dataset>.

[10] Supervisely - Web platform for computer vision. Annotation, training and deploy, [Online]. <https://supervise.ly/>

CERTIFICATE



Certificate of Participation
★★★★★



PANIMALAR ENGINEERING COLLEGE
A Christian Minority Institution-Jaisakthi Educational Trust
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
Accredited by National Board of Accreditation(NBA)
Approved by UGC for 2(f) & 12(B) Status
Bangalore Trunk Road, Poonamallee, Chennai-600123.



ICoNIC


PECTEAM 2K21

This is to certify that Dr./Mr./Ms **B.sushmitha** of CSE from Panimalar Engineering College has presented paper entitled Design and implementation of light weight CNN model for Fire detection in the **4th International Conference on Intelligent Computing (ICoNIC)** held on 26th & 27th March 2021.




CONVENER
Dr.S.MALATHI,M.E.,Ph.D







CO-CONVENER
Dr.V.D.Ambeth Kumar,M.E.,Ph.D.,




CO-PATRON
DR.K.MANI,M.E.,Ph.D.,






Certificate of Participation
★★★★★



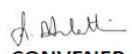
PANIMALAR ENGINEERING COLLEGE
A Christian Minority Institution-Jaisakthi Educational Trust
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
Accredited by National Board of Accreditation(NBA)
Approved by UGC for 2(f) & 12(B) Status
Bangalore Trunk Road, Poonamallee, Chennai-600123.




ICoNIC


PECTEAM 2K21

This is to certify that Dr./Mr./Ms **S.Sowmiya** of CSE from Panimalar Engineering College has presented paper entitled Desgin And Implementation Of A Light Weight CNN Model For Fire Detection in the **4th International Conference on Intelligent Computing (ICoNIC)** held on 26th & 27th March 2021.




CONVENER
Dr.S.MALATHI,M.E.,Ph.D







CO-CONVENER
Dr.V.D.Ambeth Kumar,M.E.,Ph.D.,



CO-PATRON
DR.K.MANI,M.E.,Ph.D.,



We presented the paper in International Conference ICONIC 2K21 held at Panimalar Engineering College on 27th Saturday 2021.

REFERENCES

- [1] M. Maier, M. Chowdhury, B. P. Rimal, and D. P. Van, “The tactile internet: Vision, recent progress, and open challenges,” *IEEE Commun. Mag.*, vol. 54, no. 5, pp. 138–145, May 2016.
- [2] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, “5G-enabled tactile internet,” *IEEE J. Sel. Areas Commun.*, vol. 34, no. 4, pp. 460–473, Mar. 2016.
- [3] K. Muhammad, R. Hamza, J. Ahmad, J. Lloret, H. H. G.Wang, and S.W. Baik, “Secure surveillance framework for IoT systems using probabilistic image encryption,” *IEEE Trans. Ind. Inform.*, vol. 14, no. 8, pp. 3679–3689, Aug. 2018.
- [4] N. Pogkas, G. E. Karastergios, C. P. Antonopoulos, S. Koubias, and G. Papadopoulos, “Architecture design and implementation of an ad-hoc network for disaster relief operations,” *IEEE Trans. Ind. Inform.*, vol. 3, no. 1, pp. 63–72, Feb. 2007.
- [5] A. Filonenko, D. C. Hern´andez, and K.-H. Jo, “Fast smoke detection for video surveillance using CUDA,” *IEEE Trans. Ind. Inform.*, vol. 14, no. 4, pp. 725–733, Feb. 2018.

- [6] U. L. Puvvadi, K. Di Benedetto, A. Patil, K.-D. Kang, and Y. Park, “Costeffective security support in real-time video surveillance,” *IEEE Trans. Ind. Inform.*, vol. 11, no. 6, pp. 1457–1465, Dec. 2015
- [7] S. Mumtaz, A. Bo, A. Al-Dulaimi, and K.-F. Tsang, “5G and beyond mobile technologies and applications for industrial IoT (IIoT),” *IEEE Trans. Ind. Inform.*, vol. 14, no. 6, pp. 2588–2591, Jun. 2018.
- [8] D. Guha-Sapir and P. Hoyois, “Estimating populations affected by disasters: A review of methodological issues and research gaps,” Centre Res. Epidemiol. Disast., Inst. Health Soc., Univ. Catholique de Louvain, Brussels, Belgium, 2015.
- [9] D. Han and B. Lee, “Development of early tunnel fire detection algorithm using the image processing,” in *Proc. Int. Symp. Vis. Comput.*, 2006, pp. 39–48.
- [10] T.-H. Chen, P.-H. Wu, and Y.-C. Chiou, “An early fire-detection method based on image processing,” in *Proc. Int. Conf. Image Process.*, 2004, pp. 1707–1710.