

BlazeGuard: A Deep Learning Based Automated System for Fire Detection and Classification



Group Members

Muhammad Arij Kamran	19I-0795
Aahad Yousaf Raja	19I-0835
Subata Khan	19I-0842

Project Supervisor

Dr. Mukhtar Ullah

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Islamabad
2019

Developer's Submission

“This report is being submitted to the Department of Electrical Engineering of the National University of Computer and Emerging Sciences in partial fulfillment of the requirements for the degree of BS in Electrical Engineering”

Developer's Declaration

“We take complete responsibility for the project work conducted during the Final Year Project (FYP) titled “**BlazeGuard: A Deep Learning Based Automated System for Fire Detection and Classification**”. We solemnly declare that the entire project including its due tasks were executed by only the three individuals present in this group, no external assistance was taken except for that mentioned. The project report was also written by us. Moreover, we have not presented this FYP (or substantially similar project work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

We understand that the management of the Department of Electrical Engineering of National University of Computer and Emerging Sciences has a zero-tolerance policy towards plagiarism. Therefore, we as an author of the above-mentioned FYP report solemnly declare that no portion of our report has been plagiarized and any material used in the report from other sources is properly referenced. Moreover, the report does not contain any literal citing of more than 70 words (total) even by giving a reference unless we have obtained the written permission of the publisher to do so. Furthermore, the work presented in the report is our own work and we have positively cited the related work of the other projects by clearly differentiating our work from their relevant work.

We further understand that if we are found guilty of any form of plagiarism in our FYP report even after our graduation, the University reserves the right to withdraw our BS degree. Moreover, the University will also have the right to publish our names on its website that keeps a record of the students who committed plagiarism in their FYP reports.”

M. Arij Kamran

BS(EE) 2019-0795

Aahad Yousaf Raja

BS(EE) 2019-0835

Subata Khan

BS(EE) 2019-0842

Certified by Supervisor

Verified by Plagiarism Cell Officer



Dated: _____

Abstract

The risk of potential fires in closed systems like warehouses is excruciatingly high. Moreover, the lack of workers actively present in the vicinity ensures the need of an automated system that could detect and prevent fire outrages. The currently installed fire alarm systems in warehouses have a wide margin of error. Most warehouses cannot monitor heat and temperature 24/7 and neither can a business afford a long-term staff just for the fire hazard. Hence, to prevent any major losses this system will be installed. The real-time data collected on the hardware nodes will be processed to derive the root cause of the fire. This will be depicted in the shape of the 5 classes of fire: Class A, B, C, D, or K. Each class has a distinguishing feature depending on the material that is subject to the fire. This classification of fire is done through a trained convolutional neural network. This, along with the amount of smoke spread, will be displayed on a user-friendly web application, with alerts about the fire. This is how early detection of fire will prevent major damages from occurring.

Acknowledgements

We would like to express our sincere gratitude to our supervisors, Dr. Mukhtar Ullah for providing their invaluable guidance, comments and suggestions throughout the course of this project. We thank him for constantly motivating us to work hard and teaching us about the importance of group work.

We would also like to mention the people in our FYP panel who made critical evaluations of our final year project thus giving us room for improvement and making our project better every time.

Table of Contents

DEVELOPER’S SUBMISSION-----	2
DEVELOPER’S DECLARATION-----	3
ABSTRACT-----	4
ACKNOWLEDGEMENTS-----	5
TABLE OF CONTENTS-----	6
List of Figures-----	7
LIST OF TABLES-----	7
CHAPTER 1 INTRODUCTION-----	8
MOTIVATION-----	8
1.1 PROBLEM STATEMENT-----	8
1.2 LITERATURE REVIEW-----	8
1.4 REPORT OUTLINE-----	9
CHAPTER 2 SOLUTION DESIGN & IMPLEMENTATION-----	10
2.1 BLOCK DIAGRAM-----	10
2.2 FLOW CHART-----	13
2.3 SOFTWARE IMPLEMENTATION-----	14
2.4 HARDWARE IMPLEMENTATION-----	26
CHAPTER 3 RESULT AND RECOMMENDATIONS-----	29
3.1. Results:-----	29
3.2. Project Management:-----	32
3.3 Budget-----	33
3.4 CONCLUSIONS-----	34
3.5 RECOMMENDATIONS / FUTURE WORK-----	34
Chapter 4 Societal Impacts-----	35
4.1 Sustainable Development Goals-----	35
4.2 Lifelong Learning-----	35
4.3 Benefits to the society-----	35
APPENDIX-A: PROJECT CODES-----	36
BIBLIOGRAPHY-----	38

List of Figures

Figure 2.1: Block diagram of the project.....	10
Figure 2.2a MQ-2 Sensor.....	11
Figure 2.2b ESP-32.....	12
Figure 2.3: Flow Chart of the project.....	13
Figure 2.4: CNN Layers.....	15
Figure 2.5: Training and validation accuracies.....	16
Figure 2.6 Flow chart of Client Configuration.....	17
Figure 2.7 Sample testing model.....	21
Figure 2.8a: Setting up IoT on AWS.....	21
Figure 2.8b: Creating a Thing.....	22
Figure 2.9 a) IoT Rule to connect to Timestream. b) Attributes for the Database.....	24
Figure 2.9 c) SQL Query to upload the data.....	24
Figure 2.10a: Web interface Home page.....	25
Figure 2.10b: App interface Home page.....	25
Figure 2.11: Login Page.....	25
Figure 2.12: Table displaying Sensor data and Class of fire.....	26
Figure 2.13a AutoCAD view of the model.....	28
Figure 2.13b Physical view of the model.....	28
Figure 2.14: Serial monitor of the ESP32.....	29
Figure 2.15: AWS IoT Client testing.....	30
Figure 2.16a: Test Image of no fire.....	31
Figure 2.16b: Test Image of fire.....	31

List of Tables

Table 2.1: MQ-2 Pin Configurations.....	11
Table 2.2: ESPCAM pin configuration.....	27

Chapter 1 Introduction

Fire hazards have been known to cause severe damages to infrastructure. According to NFPA U.S. fire departments responded to an estimated average of 1,450 structure fires in warehouse properties per year, \$283 million in direct property damage, two civilian deaths, and 16 civilian injuries[1]. With the passage of time, these numbers have a low chance of reducing, hence, early detection of fire in real time is proposed as a solution.

Motivation

The motivation behind this project was to cater to the fire incidents that occur in our country on a daily basis. We aim to build a system that is inexpensive and provides robust fire detection and prevention, such that owners are easily able to afford it, thus saving a lot of lives from being at risk every second. Our mission is also to detect the fire at its earliest stage, so that appropriate actions can be taken readily and there is less wastage of the resources.

1.1 Problem statement

“Design an automated system for the detection and classification of fire using a deep learning algorithm ”

The proposed system is designed to detect fire at its earliest stage, using real-time monitoring of data. With the detection of fire, it is also classified to derive the reason behind the fire: the type of fire. This classification is done with the help of feature extraction of 5 types of fires that distinguish from each other, depending on the material it is caught on.

1.2 Literature review

Review of deep learning: concepts, CNN architectures, challenges, applications, future directions

This paper talks about the deep learning neural network. It goes through the background and the reasons why deep learning is an efficient approach. It also highlights the types of neural networks and the basic principles of their working. We connected this paper with our project by learning about the types of classification of data and the challenges faced while observing and executing deep learning algorithms. [2]

Exploring the Internet of Things with AWS IoT Core — Part I: Overview, Provisioning Single and Bulk Nodes

This blog talks about the entirety of setting up the infrastructure of IoT. It elaborates upon a core concept being used in our project which is sending the data and values gathered through a sensor to the cloud to store them. The basics of initiating an AWS Project are briefed in this article, which assisted us in using Amazon IoT Core and developing “things” for our hardware devices. Furthermore adding queries and proceeding steps was also discussed in this blog. [3]

1.4 Report Outline

This report is further divided into multiple chapters as listed below.

In Chapter 2, the proposed solution is discussed in detail. It includes details of the block diagram, a Flow chart of the process, and interfacing of the sensors with the processing module.

Chapter 3 discusses the results acquired by testing the device on multiple subjects in order to improve the device's accuracy. It further discusses the conclusions drawn from the obtained results and the recommendations/future work that is proposed for further enhancements

Chapter 2 Solution Design & Implementation

This chapter discusses the complete design and implementation of the proposed device. Section 2.1 discusses the block diagram and module specifications. The details of the flowchart are presented in section 2.2. Section 2.3 discusses in detail all of the software configurations done for the project and finally, the hardware implementation is presented in Section 2.4.

2.1 Block Diagram

Figure 2.1 shows the complete block diagram of the project.

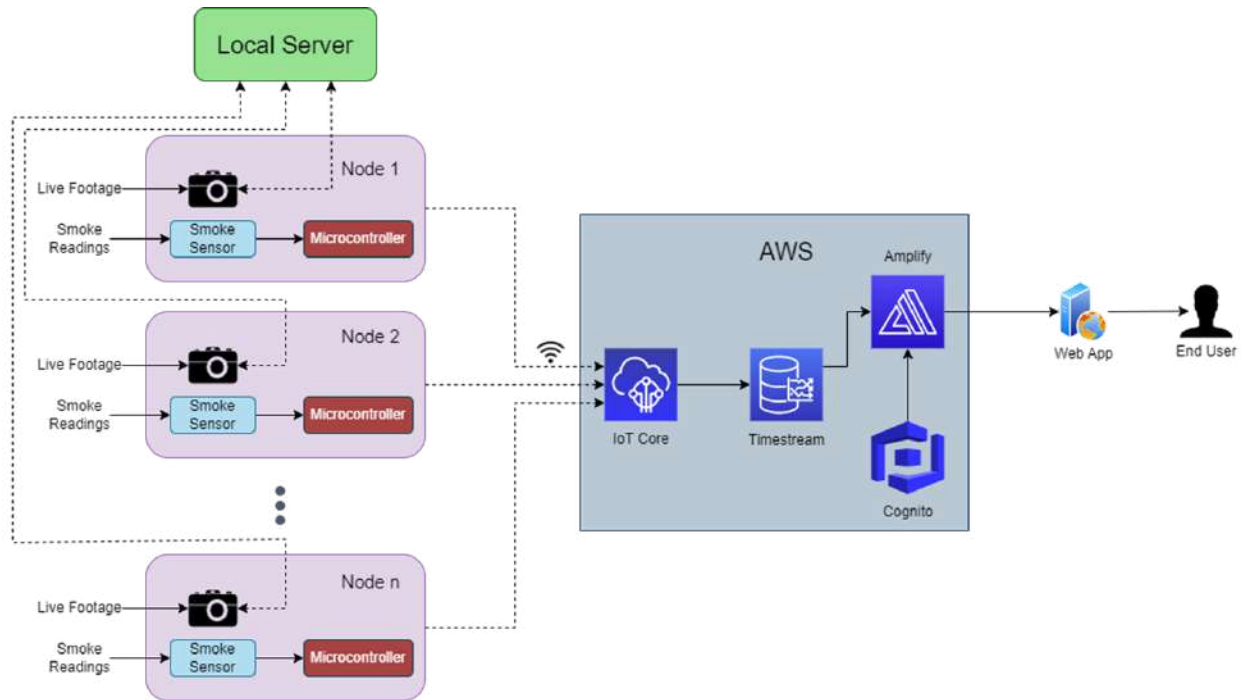


Figure 2.1: Block diagram of the project

Following are the details of the hardware used for the scope of this project.

ESP32 CAM:

An FTDI programmer refers to a programming device or module that utilizes an FTDI (Future Technology Devices International) USB-to-serial converter chip to facilitate programming and communication between a computer and another device. FTDI programmers are commonly used in the field of embedded systems and electronics for programming microcontrollers, configuring devices, and debugging.

FTDI Programmer:

An FTDI programmer refers to a programming device or module that utilizes an FTDI (Future Technology Devices International) USB-to-serial converter chip to facilitate programming and communication between a computer and another device. FTDI programmers are commonly used in the field of embedded systems and electronics for programming microcontrollers, configuring devices, and debugging.

MQ2 Sensor:

An MQ2 sensor is a smoke sensor that detects the presence of various gasses in the vicinity it is placed. It has 4 pins; their names and functions are mentioned below.


Pins	Functionality	 <p>Figure 2.2a MQ-2 Sensor</p>
Vcc	To provide power supply to the sensor module	
Gnd	For common ground between the sensor and MCU	
D0	A digital pin for a boolean indication of presence of smoke	
A0	An analog pin to determine the amount of smoke in the surroundings	

Table 2.1: MQ-2 Pin Configurations

We didn't use the digital pin since we just needed to send the concentration of smoke in the surrounding and our requirement was fulfilled by using the analog pin only.

ESP32:

ESP32 is a RISC based microcontroller that either uses the Tensilica Xtensa or LX7 microprocessor. It is well known for its specifications. It has a built in WiFi and Bluetooth module and also supports communication through various protocols i.e. UART, SPI, I2C, MQTT etc. It features 34 physical GPIO pins which can be used to collect and communicate digital and analog data.



Figure 2.2b ESP-32

2.2 Flow Chart

Figure 2.3 shows the complete flow chart of the project. The details are discussed below.

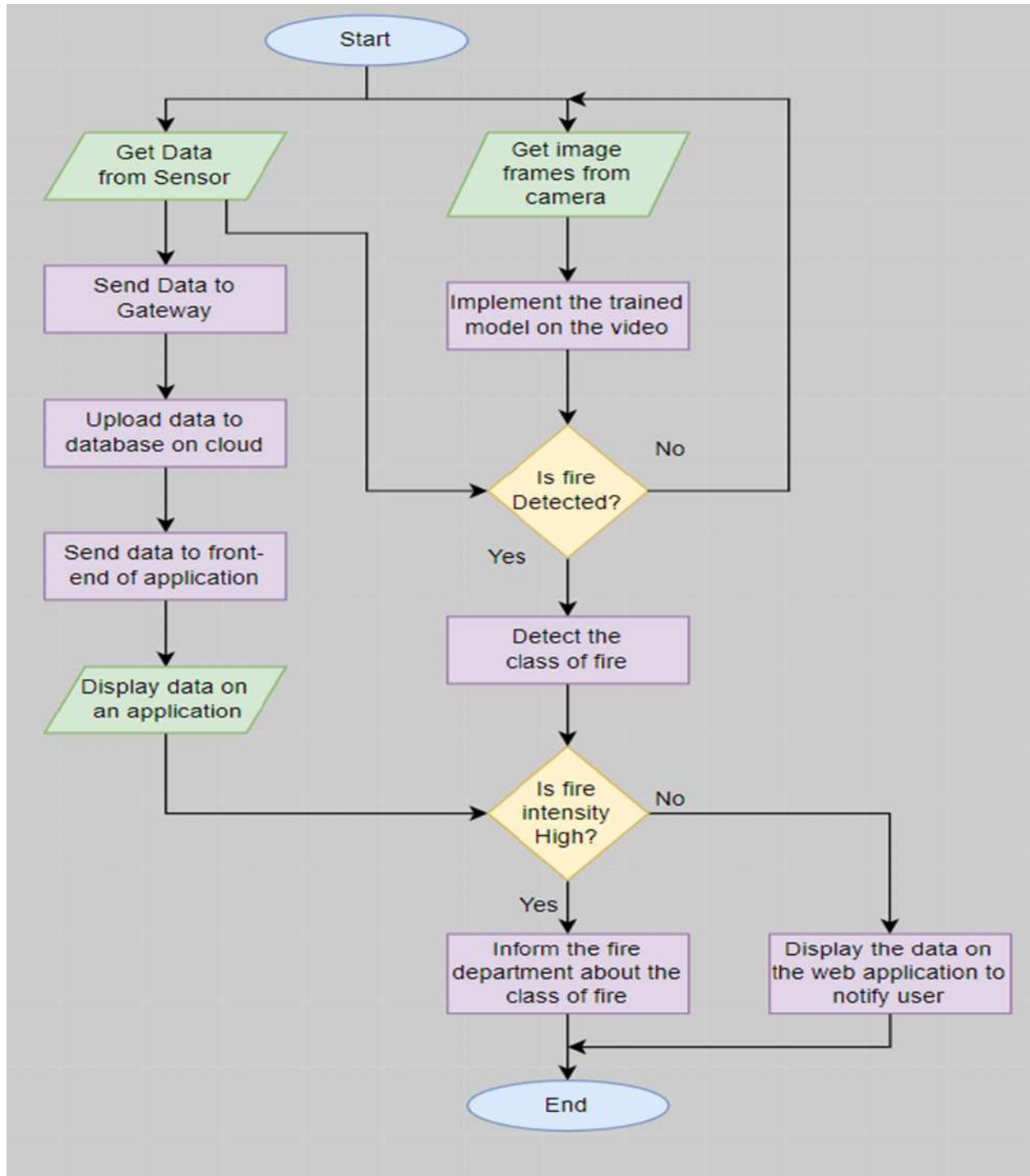


Figure 2.3: Flow Chart of the project

The details of this flow chart are provided further in this section

2.3 Software Implementation

Convolutional Neural Network (CNN) Implementation:

Dataset Preparation:

In order to implement the neural network we started with creating a dataset for the detection of fire. We gathered data belonging to two classes, one which contained images of fire and the other one containing images of the surroundings under normal conditions. Once we were done with this we moved on to the creation of the dataset that would distinguish between the types of fire. For that we gathered data belonging to 5 classes which are the actual classes of fire namely class A, class B, class C, class D and class K.

Data Augmentation:

We did data augmentation to artificially increase the size and diversity of a training dataset by applying various transformations to the existing data. Following are the transformations that we applied for our project.

```
# Rescaling the data to generate multiple examples from a single image
training_datagenerator = ImageDataGenerator(rescale = 1./255, horizontal_flip = True,
                                             vertical_flip = True, shear_range = 0.2,
                                             zoom_range = 0.2, width_shift_range = 0.2,
                                             height_shift_range = 0.2, validation_split = 0.1)
```

Splitting the data:

In order to check how well the neural network is created we divided our dataset into two portions. One of the portions containing 90% of the data was used for training the model while a smaller portion i.e. 10% of the data was used for validation of the model generated.

Neural Network creation:

We then proceeded with creating the neural network which had an input layer followed by 3 hidden layers having a **ReLU** (Rectified Linear Unit) as their activation function. After this, we used a flatten layer and then an **FC** (fully connected layer) before an output layer. A detailed summary of the model is given below.

```

Model: "sequential"

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 32)	896
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 32)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 128)	4194432
dense_1 (Dense)	(None, 1)	129

```

Total params: 4,213,953
Trainable params: 4,213,953
Non-trainable params: 0

```

Figure 2.4: CNN Layers

Fit Generation:

To generate the fit we used Adam optimizer which is an optimized form of gradient descent. It prevents the model from crawling across local minima and it also prevents zigzagging which are both responsible for poor performance. The cost function used here is **binary cross entropy** since we had two classes of data for the initial detection of fire. After this, the model fit is generated.

```
# Training the cnn model
cnn.compile(optimizer = "Adam", loss = 'binary_crossentropy', metrics = ['accuracy']) # optimizer to reduce the error

history = cnn.fit(train, validation_data = validation, epochs = 1, steps_per_epoch = train.samples//batch_size,
                 validation_steps = validation.samples//batch_size, callbacks = [checkpoint])
```

Similarly a fit was generated for the 5 classes of fire too using the same neural network architecture and the weights of both fits were saved into .h5 files which could later be loaded anywhere to make predictions using the model.

Overfitting and Underfitting Check:

Following is the plot which shows the training set accuracy and optimization as well as the validation set accuracy and optimization for every epoch when the fit is generated for 10 epochs.

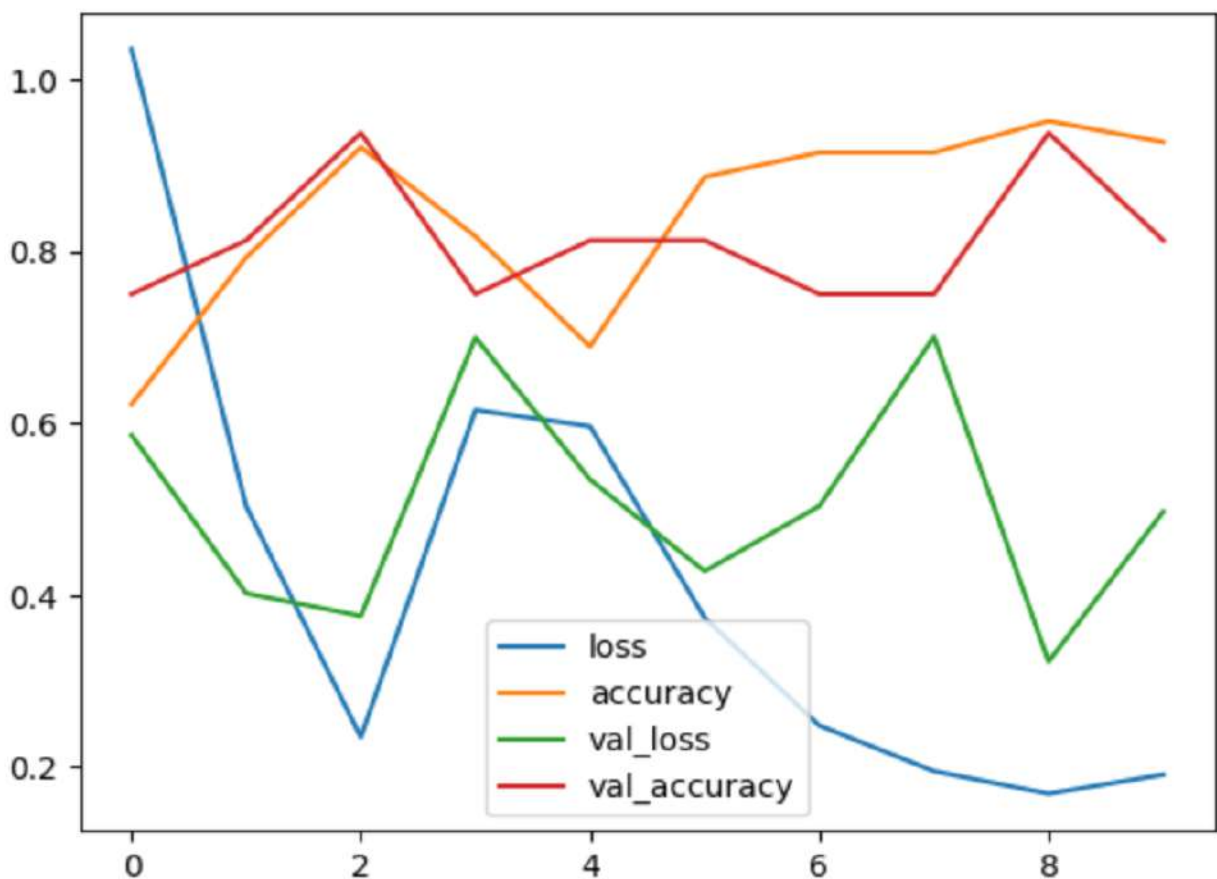


Figure 2.5: Training and validation accuracies

We can clearly see that when the accuracy is high the optimization is low and vice versa which shows that there is no overfitting or underfitting in the fit generated by the convolutional neural network because it is established already that high optimization and high accuracy result in model overfitting and low accuracy and low optimization result in an under fitted model.

Local Server Implementation

In order to get images from the ESPCAM we made it a client and then implemented a local server with the help of socket programming. We started with creating a TCP socket so that none of the images sent by the client would be lost due to the reliable connection and all the predictions made at the local server will be sent back to the client too.

Client Configuration

Client side configuration of the socket will be explained here while the camera interfacing will be explained in the hardware implementation section of this chapter.

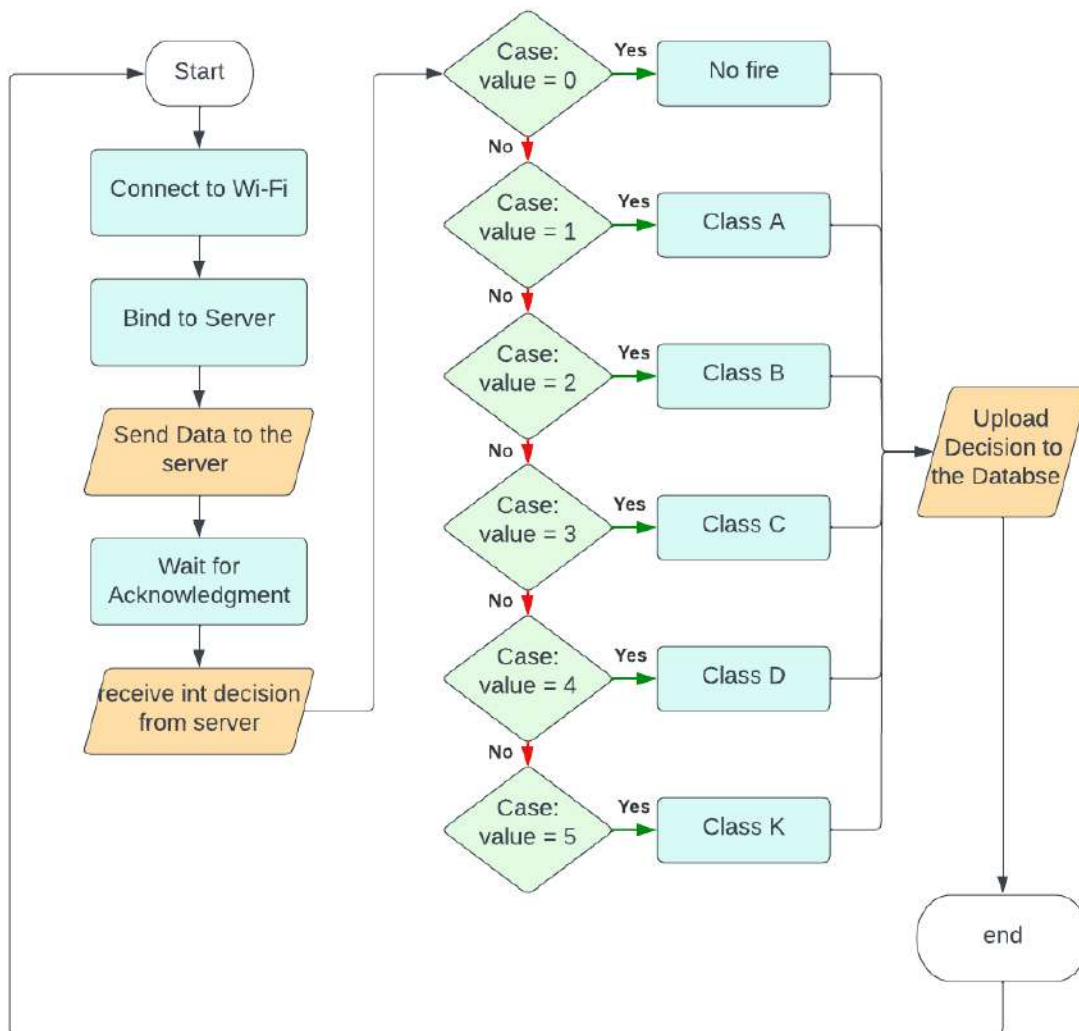


Figure 2.6 Flow chart of Client Configuration.

Code explanation

1. initially we will include the libraries

```
#include "esp_camera.h"
#include <WiFi.h>
```

2. Connect the client with the required server information

```
if (!client.connected())
{
    if (!client.connect(server_ip, server_port))
    {
        //Serial.println("Connection to server failed");
        return;
    }
    //Serial.println("Connected to server");

    delay(1000);
    return;
}
```

3. To send the data to the server we use the following line of code

```
client.write(data, fb->len);
```

4. To receive the data we will use the function `getResponse`. this response is the classification result of the image we sent

```
int receiveInt(WiFiClient& client) {
    String data;
    while (client.available()) {
        char c = client.read();
        if (c == '\n') {
            break;
        }
        data += c;
    }
    return data.toInt();
}
```

5. This data is then uploaded to the AWS server (Explained in the next section of the chapter)

```
void publishMessage()
{
    StaticJsonDocument<200> doc;

    doc["Dev_Name"] = Dev_Name;
    doc["class"] = classes;

    char jsonBuffer[512];
    serializeJson(doc, jsonBuffer); // print to client

    client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
}
```

One thing to be noted is that ESP-CAM and ESP32 are both created as separate IoT “Things” and the data will be stored in separate tables of the same database.

Server Configuration

1. Importing dependencies

```
import io
import socket
import threading
import time
import numpy as np
from PIL import Image
```

- a. io is used to create a stream for converting image bytes into a PIL Image object.
 - b. socket is used to create a socket, bind it to a specific address and port, listen for incoming connections, and handle client connections.
 - c. threading handles each client connection in a separate thread, allowing concurrent execution of multiple client requests.
 - d. time This library provides functions for working with time-related operations.
 - e. numpy provides support for multi-dimensional arrays and various mathematical operations on arrays
 - f. PIL (Python Imaging Library) is used for handling and manipulating images. In the code,
2. Then we create a socket for TCP communication, open it for any device in the network to connect with it, and listen to 5 devices at once

```
serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serv.bind(('0.0.0.0', port))
serv.listen(5)
```

3. As a device connects it will be sent to a separate thread for handling the client

```
while True:
    conn, addr = serv.accept()
    print("Connected by", addr)
    thread = threading.Thread(target=handle_client, args=(conn, addr))
    thread.start()
```

4. Now in the thread, it will receive image and image data. Initially, information about the image being sent is received i.e shape, height, and width. then chunks of the image are received according to the buffer size.

```

def handle_client(conn, addr):
    array_from_client = bytearray()
    shape = None
    chunks_received = 0
    start = time.time()
    shape_string = ''
    while True:
        data = conn.recv(BUFFER_SIZE)
        if not data or data == b'tx_complete':
            break
        elif shape is None:
            shape_string += data.decode("utf-8")
            if shape_string.find('\r\n') != -1:
                width_index = shape_string.find('width:')
                height_index = shape_string.find('height:')
                width = int(shape_string[width_index + len('width:'): height_index])
                height = int(shape_string[height_index + len('height:'): ])
                shape = (width, height)
                print("shape is {}".format(shape))
        else:
            chunks_received += 1
            array_from_client.extend(data)
            conn.sendall(b'ack')
    print("chunks_received {}. Number of bytes {}".format(chunks_received, len(array_from_client)))
    img: Image.Image = create_image_from_bytes(array_from_client)
    img.show()

```

5. Now we will perform our deep-learning algorithm on “img” (Explained in the next section)
6. The decision is then sent back to the client.

Model Predictions on local server

Both the models were loaded at the local server and then used to make predictions for the images that were received at the runtime by the client. An example of this is given below.



Figure 2.7 Sample testing model

Connecting Amazon Web Services(AWS) to ESP32

AWS is a cloud computing platform that offers services to developers to build and deploy applications. AWS IoT Core is one of the services that is designed for Internet of Things applications. it provides a secure, reliable and user-friendly platform for connecting Things (IoT endpoints) to the AWS cloud.

Prerequisites:

1. AWS Account Set-up
2. Hardware:
 - a. ESP-32
 - b. MQ-2 Sensor
3. Downloading Dependencies
 - a. WiFiClientSecure.h library for secure wireless connection
 - b. PubSubClient.h for MQTT protocol
 - c. ArduinoJson.h for creating and manipulating JSON objects to send to AWS IoT Core
 - d. time.h for time-related information

Setting up AWS IoT “Thing”

1. Log in to your AWS Account then from the home page click on *Services->Internet of Things-> IOT Core*

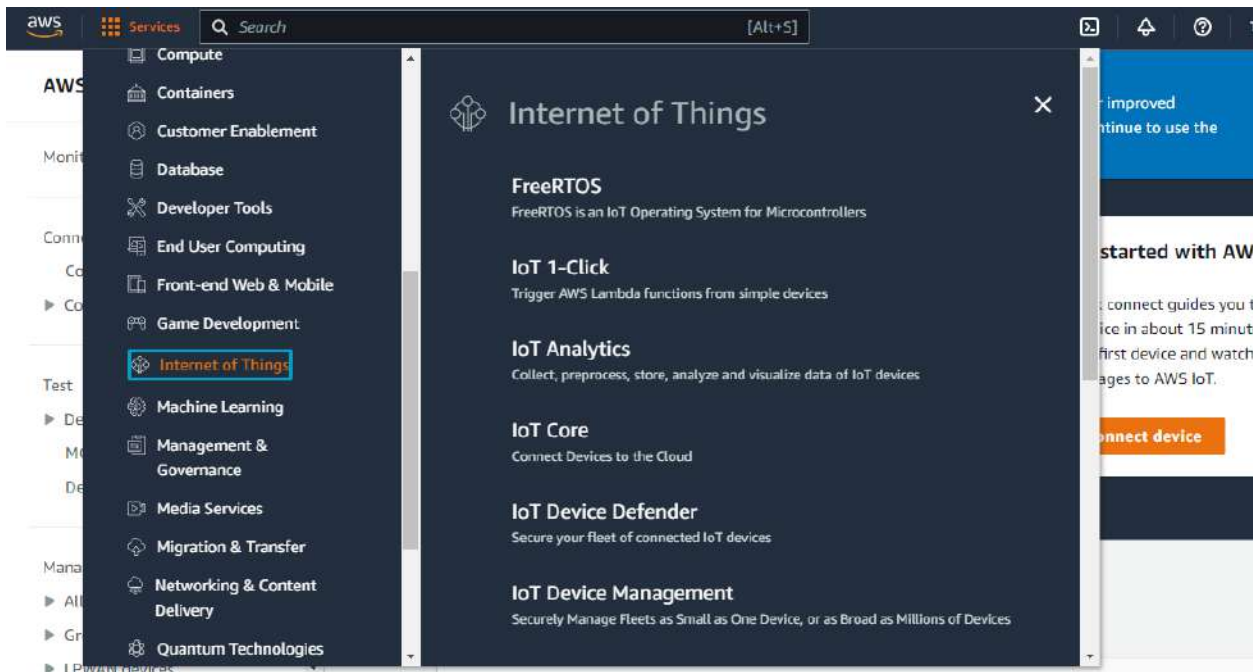


Figure 2.8a: Setting up IoT on AWS

2. Creating a thing Go to *Manage-> All devices-> Thing*.

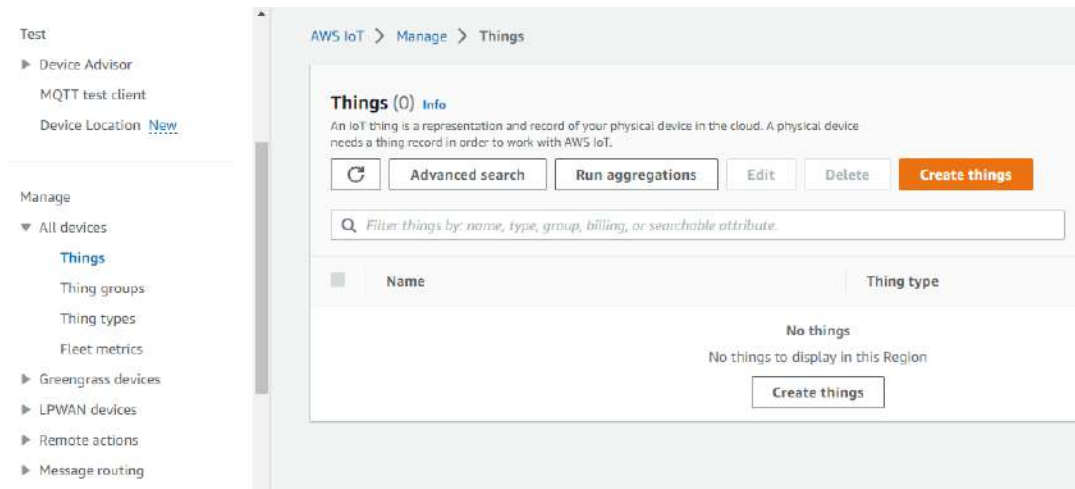


Figure 2.8b: Creating a Thing

3. Then click on create a thing. Set up a unique name for that thing. select no shadow, autogenerate new certificates, create a policy, and then download the certificates

Setting up ESP32 as the “Thing”

1. After downloading the libraries we will create a header file called Secrets.h

```
#define THINGNAME "FYP_ESP32_1" // name of the thing
const char WIFI_SSID[] = "*****";
const char WIFI_PASSWORD[] = "*****";
const char AWS_IOT_ENDPOINT[] = "*****";
// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
)EOF";
// Device Certificate
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
)KEY";
// Device Private Key
static const char AWS_CERT_PRIVATE[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
)KEY";
```

- Open the certificates and keys with notepad and copy paste the text
 - To get AWS_IOT_ENDPOINT we can go to settings on the AWS account and copy it from there
2. Define the topic for the MQTT protocol to send or receive data

```
#define AWS_IOT_PUBLISH_TOPIC "esp32/pub"  
#define AWS_IOT_SUBSCRIBE_TOPIC "esp32/sub"
```

3. Configure NTP to get current time

```
//=====NTP Config for time=====//  
const char* ntpServer = "pool.ntp.org";  
const long  gmtoffset_sec = 0;  
const int   daylightOffset_sec = 3600;  
  
struct tm timeinfo;  
//time_t now = time(NULL);  
char timestamps[20];  
//=====//
```

4. Now we get sensor data process it and then publish it to AWS Iot Core

```
void publishMessage()  
{  
    StaticJsonDocument<200> doc;  
  
    doc["Dev_Name"] = Dev_Name;  
    doc["timestamps"] = timestamps;  
    doc["gassensorAnalog"] = gassensorAnalog;  
    doc["Percentage"] = smoke;  
    doc["intensity"] = intensity;  
    char jsonBuffer[512];  
    serializeJson(doc, jsonBuffer); // print to client  
  
    client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);  
}
```

Linking AWS IoT Core to AWS Timestream Database

We will use Timestream as a database because Timestream is purpose-built for handling time-series data, making it an ideal choice for storing IoT sensor data that is inherently time-based. It efficiently organizes and manages large volumes of timestamped data, allowing for easy retrieval and analysis based on time intervals.

To link the AWS IoT core to Timestream we will use a feature called AWS IoT Core Rules Engine

Figure 2.9 a) IoT Rule to connect to Timestream. b) Attributes for the Database

Figure 2.9 c) SQL Query to upload the data

Website:

In order to represent all this data to the user in an interactive way we created a website that had details about the product and also highlighted options to buy the product. Furthermore, for the people who had availed the services of our product, we added a login option that redirected our user after authentication to the page where all the data collected was presented interactively. Following are some screenshots of the website front-end for users navigating through web browsers and cell phones.

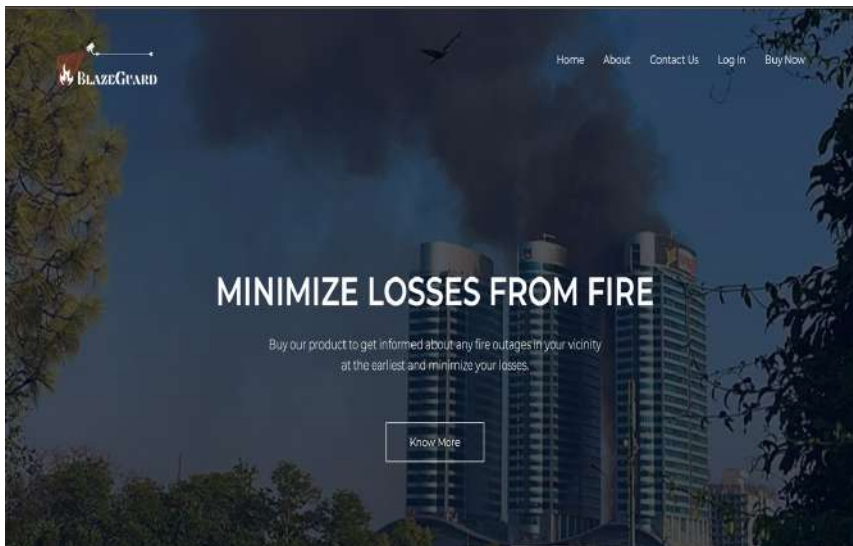


Figure 2.10a: Web interface Home page

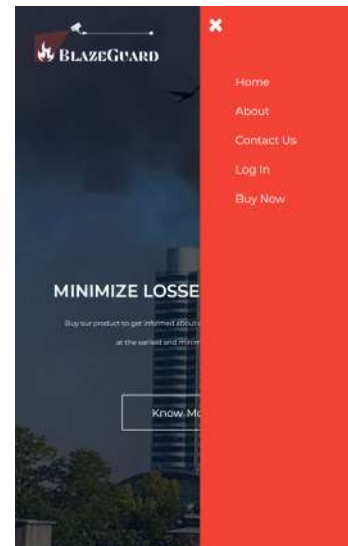
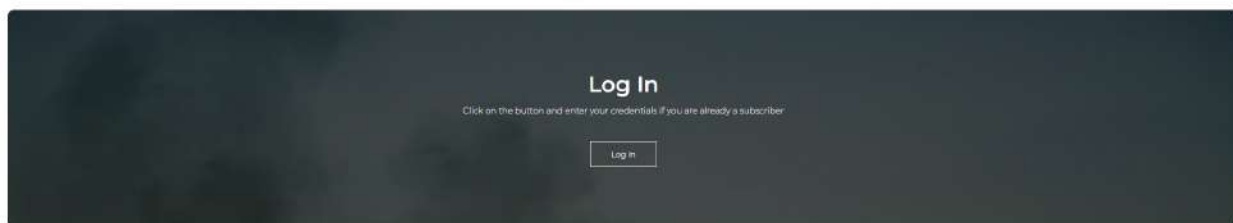


Figure 2.10b: App interface Home page



Buying Details



Figure 2.11: Login Page

Graphical Presentation of Data:



Figure 2.12: Table displaying Sensor data and Class of fire

2.4 Hardware Implementation

Code chunk to get data from MQ2 sensor:

The sensor detects the amount of smoke in air and gives a voltage reading accordingly at the A0 pin. This voltage reading is sampled by the ADC at the microcontroller between 0 and 5V and thus the sampled value is taken as the measured amount of smoke when we use the function `analogRead`.

```
int Gas_analog = 4;    // used for ESP32

void setup() {
  Serial.begin(115200);
}

void loop() {
  int gassensorAnalog = analogRead(Gas_analog);
  Serial.println(gassensorAnalog);
  delay(100);
}
```

Camera Interfacing:

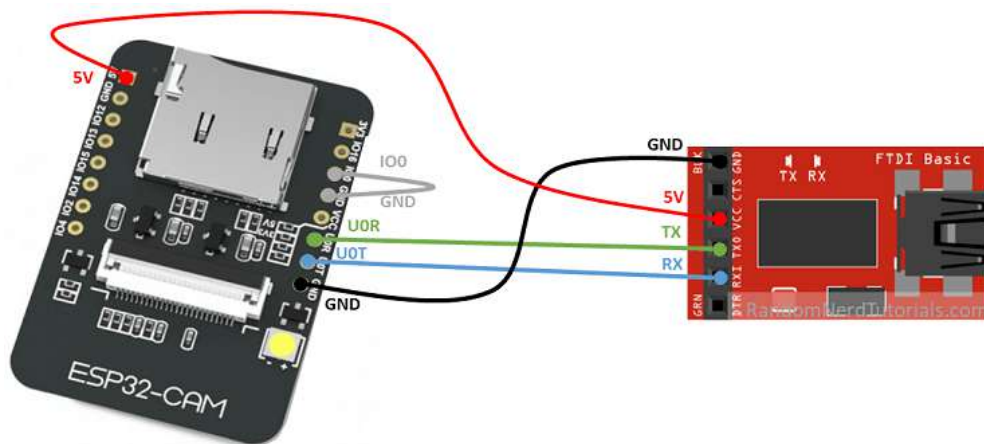
ESPCAM will be used as a camera to take live frames and stream them on an IP Address. To interface the ESPCAM module we will use an FTDI Programmer. the pin configuration is as mentioned below:

ESPCAM	FTDI
GND	GND
5V	VCC(5V)
U0R	TX
U0T	RX
GPI0 0	GND

Table 2.2: ESPCAM pin configuration

The Code will be Uploaded using Arduino IDE. Installation of ESP32 board is required. After this go to **Boards-> ESP32-> AI Thinker** . Once that board is selected we can add the code and upload.

While Uploading the Code, The IO0 pin is GND of ESPCAM while we are uploading the code. when “Connecting....” is displayed on the Serial Monitor of Arduino IDE, press the RESET button of ESPCAM repeatedly. Once the Code is Uploaded, remove the IO0 pin and press Reset button. The Serial Monitor will Display an IP Address on which Video will be streamed.



Hardware Casing:

We designed the casing for the hardware using AutoCAD 2023. The 3D diagram for the casing alongside the 3D printed casing is shown in the figure below.

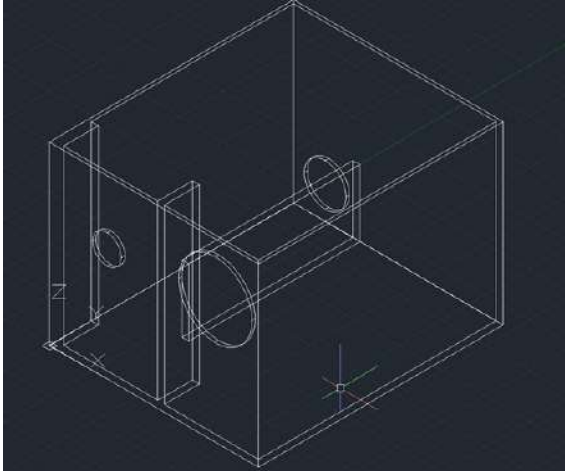


Figure 2.13a AutoCAD view of the model



Figure 2.13b Physical view of the model

Chapter 3 Result and Recommendations

The final prototype is ready for deployment, showcasing the successful implementation of a node equipped with a convolutional neural network (CNN) model. The CNN model was trained meticulously on diverse datasets and optimized to achieve optimal training accuracy and minimal validation error. The integrated smoke sensor demonstrated remarkable precision in detecting smoke levels. Rigorous testing was conducted on different materials, including wood, paper, and even magnesium (fire on metal), validating the node's effectiveness in fire detection across various scenarios.

3.1. Results:

Smoke sensor readings

The ESP32 device, integrated with the MQ-2 sensor, was subjected to small smoke generated from a matchstick to obtain the following results. The dataset comprises five key attributes: the timestamp indicating when the data was read from the sensor, the unique device name assigned to each device, the intensity derived from the analog value, the analog value ranging from 0 to 4095, representing the concentration range of the smoke detected by the sensor, and the percentage of smoke present in the environment. These results provide valuable insights into the smoke detection capabilities of the ESP32 and MQ-2 sensor combination.

```

Time Stamp : 2023-03-28 18:38:08 || Device Name: ESP32_2 || Intensity: Low || Analog Value: 1097 || Percentage: 26.79%
Time Stamp : 2023-03-28 18:38:09 || Device Name: ESP32_2 || Intensity: Low || Analog Value: 1018 || Percentage: 24.86%
Time Stamp : 2023-03-28 18:38:10 || Device Name: ESP32_2 || Intensity: Low || Analog Value: 961 || Percentage: 23.47%
Time Stamp : 2023-03-28 18:38:11 || Device Name: ESP32_2 || Intensity: Low || Analog Value: 794 || Percentage: 19.39%
Time Stamp : 2023-03-28 18:38:12 || Device Name: ESP32_2 || Intensity: Low || Analog Value: 639 || Percentage: 15.60%
Time Stamp : 2023-03-28 18:38:13 || Device Name: ESP32_2 || Intensity: Minimal || Analog Value: 0 || Percentage: 0.00%
Time Stamp : 2023-03-28 18:38:14 || Device Name: ESP32_2 || Intensity: Minimal || Analog Value: 306 || Percentage: 7.47%
Time Stamp : 2023-03-28 18:38:15 || Device Name: ESP32_2 || Intensity: Minimal || Analog Value: 112 || Percentage: 2.74%

```

Figure 2.14: Serial monitor of the ESP32

Readings displayed on AWS IoT Core



Figure 2.15: AWS IoT Client testing

Testing the CNN:

Here are some test images used for testing the model alongside the model predictions. The model gave correct predictions for fire and none corresponding to the images fed to it for testing.

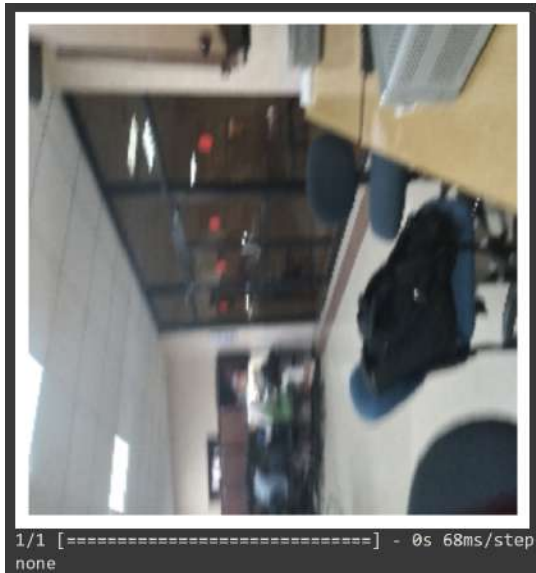


Figure 2.16a: Test Image of no fire



Figure 2.16b: Test Image of fire

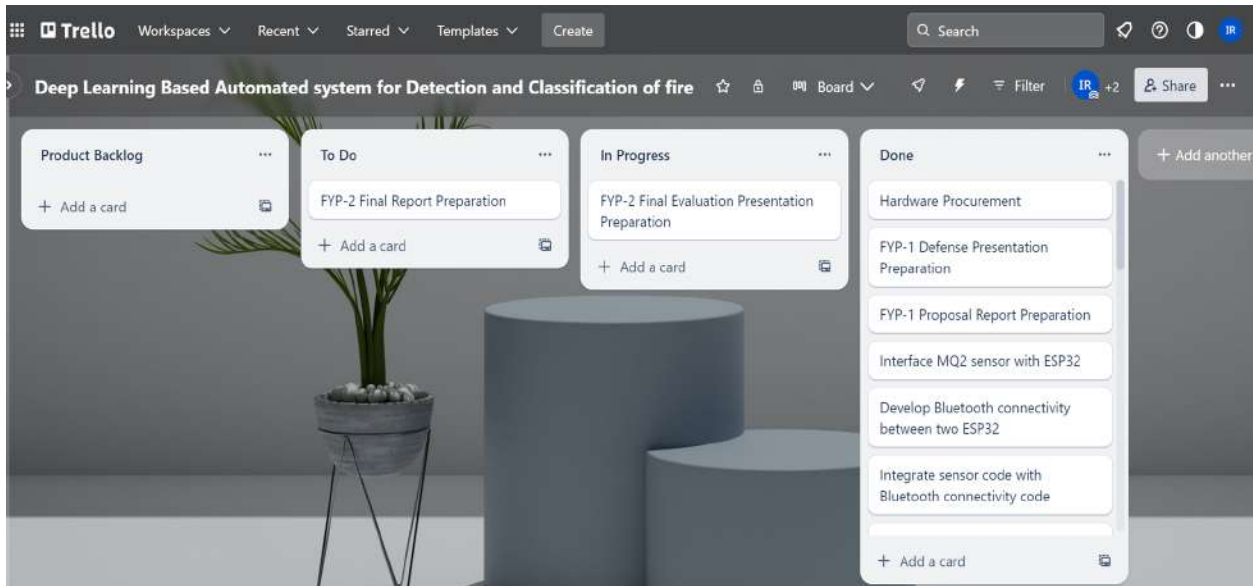
To check the accuracy of the model we went through multiple parameters including the accuracy and optimization of the model on training and validation data as well as the overfitting and underfitting check. Following figure shows the detailed values of these parameters:

```
cnn.fit_generator(train, validation_data = validation, epochs = 1, steps_per_epoch = train.samples//batch_size,  
118/118 [=====] - 556s 5s/step - loss: 0.3432 - accuracy: 0.8747 - val_loss: 0.1154 - val_accuracy: 0.9663  
<keras.callbacks.History at 0x7f1f45640580>
```

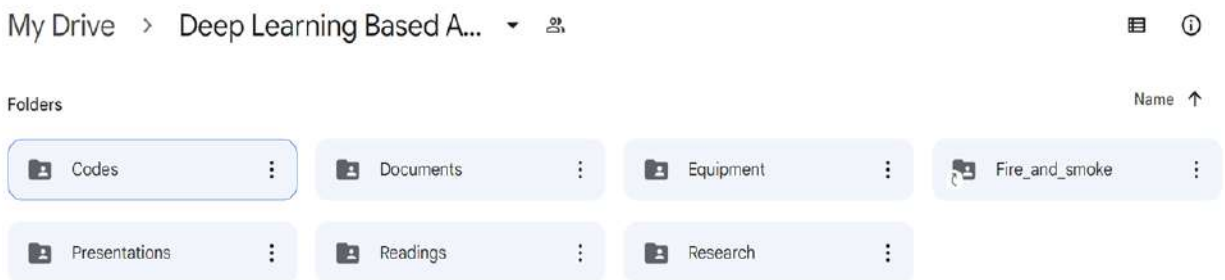
3.2. Project Management:

We used Trello to have a synchronization between the group members. All of the tasks to be done for the project were placed into the backlog at the start of the project and we kept updating their status according to the progress made regarding all of them.

Here is a screenshot of the Trello board that we created



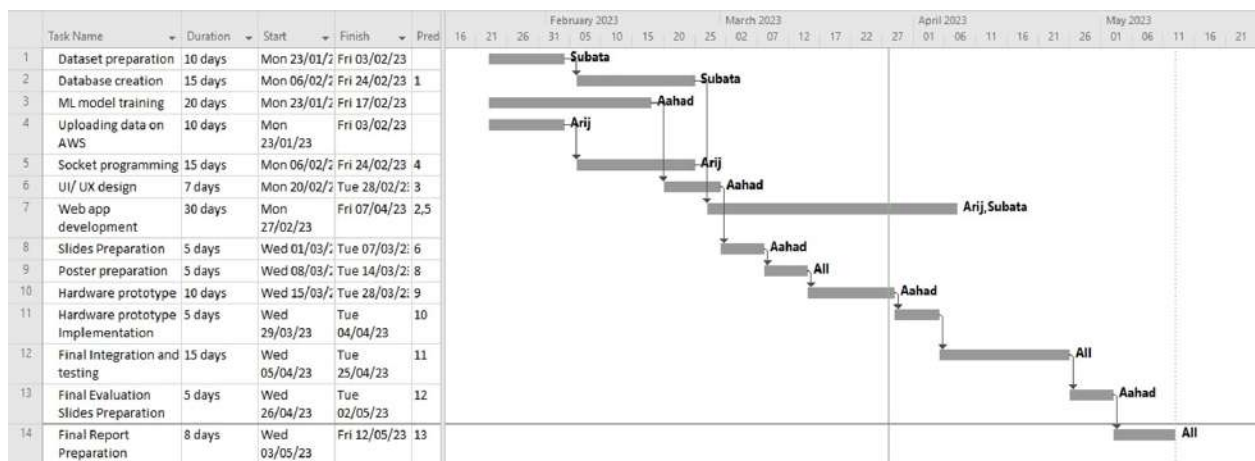
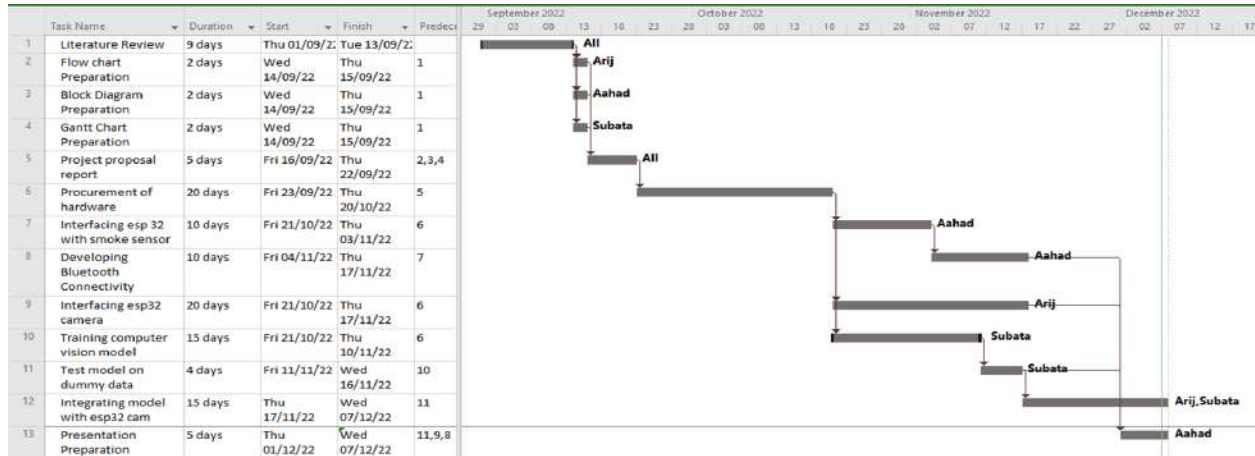
The major communication of the project with our supervisor was done using Google Drive. We kept uploading all of our data into google drive folders which were also created at the start of the project and data could be accessed from them at any moment.



We also prepared a gantt chart at the start of our project in order to estimate the time that we would require to complete this project and to know whether we were committing a doable amount of work. We remained on schedule for the tasks and any slack was utilized in the best way possible.

Following are the Gantt charts of FYP 1 and FYP 2.

Chapter3: Result and Recommendations



3.3 Budget

The budget of this project is mentioned below according to the recent prices.

Node				
Requirement	IC Name	Quantity	Price/Unit (Rs)	Cost (Rs)
Microcontroller	ESP32	2	1050	2100
Camera	OV2640	2	1400	2800
Smoke Sensor	MQ-2	2	300	600
Adaptor	-	2	500	1000
Total Cost of all nodes (Rs):				6500
Delivery Charges				1000
Cost without Overhead (Rs):				7500
Overhead (10%) (Rs):				750
Total Cost (Rs)				8250/-

3.4 Conclusions

BlazeGaurd is an innovative IoT and Deep Learning-based solution designed to address the limitations of traditional smoke detectors in detecting fires early. By leveraging advanced technologies, BlazeGaurd offers an economical and feature-rich solution for enhanced fire detection. One crucial aspect of fire management is determining the class of fire to effectively extinguish it. Our solution is capable of detecting and classifying the type of fire, providing this critical information to the relevant authorities through a user-friendly web application. With BlazeGaurd, users can benefit from improved fire detection accuracy and timely response, ultimately enhancing safety and minimizing potential damages caused by fires.

3.5 Recommendations / Future Work

While the BlazeGaurd system has reached a significant milestone in its development and showcases promising capabilities for early fire detection, there are several areas where further enhancements and expansions can be pursued.

Integration with the current extinguishing system

An important avenue to explore is the integration of the BlazeGaurd system with existing fire extinguishing systems or building management systems. By establishing seamless interoperability between the BlazeGaurd system and these systems, it becomes possible to enhance the response to fire incidents. Efforts can be focused on developing protocols and mechanisms that enable the BlazeGaurd system to trigger automatic fire suppression actions based on the detected fire class and severity. This integration would significantly improve the efficiency and effectiveness of fire control measures, enabling swift and targeted responses to mitigate potential risks.

Mobile Application for Real-Time Alerts

Due to time constraints and limited expertise in application development, the implementation of a dedicated mobile application for real-time fire alerts was not feasible within the scope of the project. However, as a future recommendation, it is highly recommended to develop a mobile application that allows users to receive timely fire alerts and notifications directly on their smartphones. By implementing push notifications, users can be promptly informed about the presence of a fire, its class, and recommended actions to take.

Emergency Response System

Integrating the BlazeGaurd system with emergency response systems, such as fire departments or emergency services, is a crucial future consideration. By establishing connections with these systems, the project can enable the automatic transmission of fire alerts, precise location details, and essential sensor data to emergency responders in real-time.

Chapter 4 Societal Impacts

4.1 Sustainable Development Goals

The system is mapped onto 3 sustainable development goals provided by the United Nations. These include goal 3: good health and well-being, goal 8: decent work and economic growth and goal 9: industry, innovation and infrastructure. Goal 3 linked to the early-detection feature of the system. The earlier the fire is detected, the lesser damage is perceived leaving no or less impact on human health due to CO and CO₂ released. Goal 8 is achieved by making our system cost-friendly. With the growing inflation in our country, this project is relatively economical and affordable for all businesses to invest in. Fire hazards and the damages caused by it are a major problem if not controlled in time. This is where we find the opportunity to work on goal 9 which ensures innovation to keep industrial infrastructures intact.

4.2 Lifelong Learning

Lifelong learning refers to knowledge building of concepts that one self-teaches themselves when there is no absolute supervision found in the area. For the same reason, these concepts usually stick with you throughout your career. The ones that will be deployed in our project include deep learning, cloud computing, app development and interfacing different modules and sensors.

4.3 Benefits to the society

Benefits to society are somewhat similar to working on reaching the sustainable development goals mentioned previously. Our project is intended to promote safer workplaces for employees, along with deriving environmental friendly methods to prevent greater losses from occurring in fire outrages. Moreover, the system can save a big chunk of the company's budget by investing in efficient but less costly fire detection systems.

Appendix-A: Project Codes

1. ESP32 CAM client code:

https://drive.google.com/file/d/1_gafJw9XIwwvcVthr3KoA-1wEDmJcXmx/view?usp=drive_link



Scan QR code to go to the link

2. CNN Implementation:

https://drive.google.com/file/d/1M1tfjsS5w8YRzBvSJZK3zBFXhGG1vAJ/view?usp=drive_link



Scan QR code to go to the link

3. Local Server code:

https://drive.google.com/file/d/15norCFikBPkYzoJIPFQLHaeFXL7rhFgS/view?usp=drive_link



Scan QR code to go to the link

4. ESP32 code:

https://drive.google.com/file/d/1MwGEruWPvw_HE5HGkzL25aLiM3t-pxGK/view?usp=drive_link



Scan QR code to go to the link

5. Website frontend code:

https://drive.google.com/drive/folders/1TH-IJyh7dtwPF5_38qFmya0igzUUJ_3m?usp=drive_link



Scan QR code to go to the link



Bibliography

- [1] Richard Campbell, “NFPA’s Warehouse Structure Fires”, July 2022
- [2] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie & Laith Farhan, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions”, *Journal of Big Data*, March 31, 2021
- [3] Umer Farooq, "Exploring the Internet of Things with AWS IoT Core — Part I: Overview, Provisioning Single and Bulk Nodes", *medium.com*, Aug 8, 2021