EMBEDDED EDGE ML FOR FIRE PREDICTION

A PROJECT REPORT

Submitted by

SURYA PRATAP SARANGI

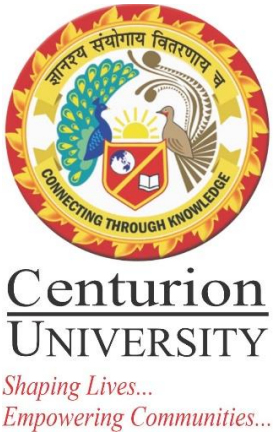
In partial fulfilment for the award of the

degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING



DEPARTMENT OF ECE

SCHOOL OF ENGINEERING AND TECHNOLOGY

BHUBANESWAR CAMPUS

CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT

ODISHA

November 2023

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

SCHOOL OF ENGINEERING AND TECHNOLOGY

BHUBANESWAR CAMPUS

BONAFIDE CERTIFICATE

Certified that this project report **"EMBEDDED EDGE ML FOR FIRE PREDICTION"** is the bonafide work of **"Surya Pratap Sarangi"** who carried out the project work under my supervision. This is to further certify to the best of my knowledge, that this project has not been carried out earlier in this institute and the university.

SIGNATURE

Prof. Swarna Prabha Jena

Department of Electronics and

Communication Engineering

Certified that the above-mentioned project has been duly carried out as per the norms of the college and statutes of the university.

SIGNATURE

Dr. Harish Chandra Mohanta

HEAD OF THE DEPARTMENT

Electronics and Communication Engineering

DEPARTMENT SEAL

DECLARATION

I hereby declare that the project entitled “Fire Prediction using Arduino Nano 33ble sense” submitted for the “Minor Project” of 3rd semester B. Tech in Electronics Communication and Engineering is my original work and the project has not formed the basis for the award of any Degree / Diploma or any other similar titles in any other University / Institute.

Name of the Student: Surya Pratap Sarangi

Signature of the student:

Registration No: 220301130011

Place: Jatani

Date:

ACKNOWLEDGEMENTS

I wish to express my profound and sincere gratitude **to Prof. Swarna Prabha Jena**, Department of Electronics and Communication Engineering, SoET, Bhubaneswar Campus, who guided me into the intricacies of this project nonchalantly with matchless magnanimity.

I thank Prof. **Dr. Harish Chandra Mohanta**, Head of the Dept. of Electronics and Communication Engineering, SoET, Bhubaneswar Campus and **Dr. Sujata Chakravarty,** Dean,School of Engineering and Technology, Bhubaneswar Campus for extending their support during Course of this investigation.

I would like to express my respect and thank all faculty members and staff of the Department of Electronics and Communication Engineering, CUTM, Jatani for their generous help on various ways for the completion of this project.

Name of the Student: Surya Pratap Sarangi

Signature of the student:

Registration No: 220301130011

Place: Jatani

Date:

TABLE OF CONTENTS

LIST OF ACRONYMS

1	IDE	Integrated Development Environment
2	USB	Universal Serial Bus
3	BLE SENSE	Bluetooth low energy

LIST OF TABLE

LIST OF FIGURES

ABSTRACT

The project titled "Fire Prediction using Arduino 33BLE Sense with Color Sensor" aims to revolutionize fire safety measures through the integration of cutting-edge technologies. The primary objective is to develop a sophisticated fire prediction system capable of early detection, real-time monitoring, and proactive risk mitigation. This system leverages the Arduino 33BLE Sense platform and a color sensor to enhance the accuracy and efficiency of fire prediction.

The project focuses on the use of the Arduino 33BLE Sense, a powerful microcontroller with built-in sensors, as the core platform. The inclusion of a colour sensor adds a novel dimension to the system, enabling it to recognize the unique visual signatures associated with flames. This holistic approach enhances the system's ability to differentiate between normal environmental variations and potential fire hazards.

The methodology involves the development of a comprehensive hardware and software interface. The Arduino 33BLE Sense collects environmental data, including temperature and humidity, while the color sensor detects changes indicative of fire. The data points are then processed using an algorithm designed to analyse colour variations and identify potential fire conditions. The system is calibrated and tested in controlled environments to ensure accuracy and reliability.

Extensive testing demonstrates the effectiveness of the Fire Prediction System. The Arduino 33BLE Sense, with its wireless capabilities, successfully transmits real-time data to a monitoring interface. The colour sensor accurately detects the unique colours associated with flames, allowing for timely and precise fire predictions. The system displays a commendable level of accuracy in differentiating between false positives and actual fire events, making it a promising tool for enhancing fire safety.

CHAPTER- 1

INTRODUCTION

1.1. Introduction

It is an embedded machine learning-based project used to predict a fire in Arduino Nano BLE 33 sense with the help of the inbuilt colour sensor attached to the board and if the fire has been detected by a led is switched on. It is far better than a typical flame and smoke sensor as one uses a heating effect and another uses a good amount of smoke to predict fire but this uses the RGB value to detect fire, which is more accurate. We have used the inbuilt colour sensor in the newly launched Arduino Nano BLE 33.

The new way to avoid all the losses is to respond to emergencies as quickly as possible. So, at that point comes the need of a upgraded fire prediction systems. This project therefore look for to design an Arduino Fire Alarm and Controlling systems that will monitor the presence of significant quantity of temperature and smoke and activate lights.

1.2. Objectives

To develop a state-of-the-art fire prediction system that utilizes advanced technologies and data-driven approaches to accurately forecast and mitigate the risk of wildfires. This system aims to enhance public safety, protect natural ecosystems, and minimize property damage through early detection, real-time monitoring, and effective resource allocation."

This objective highlights the key goals of a fire prediction system, which typically include improving accuracy, early detection, and proactive measures to reduce the impact of wildfires. The specific details and scope of the system would depend on the resources, technology, and goals of the project or organization involved.

Accurate fire prediction is crucial for early detection and efficient response. Traditional methods often fall short due to limited data and outdated techniques. By incorporating Arduino Nano BLE Sense and advanced machine learning algorithms, we can overcome these limitations and achieve higher accuracy in fire prediction. This can save lives, reduce property damage, and enable proactive fire management strategies.

CHAPTER -2

BACKGROUND STUDY

2.1. Literature Survey

A number of efforts have been put recently into designing systems that can detect and control fire outbreaks.

Burchan et al. (2019).

The paper examines the potential use of fire extinguishing balls as part of a proposed system, where drone and remote-sensing technologies are utilized cooperatively as a supplement to traditional firefighting methods. The system consists of courting unmanned aircraft
System (UAS) to detect spot fires and monitor the risk of wildfire approaching a building via remote sensing, communication UAS to establish and extend the communication channel between scouting UAS and fire-fighting UAS, and a fire-fighting UAS. One has to be very skillful in controlling drones and the system is very complex, which makes the system unreliable. [8]

Qin et al. (2018). Designed an intelligent smoke alarm system with wireless sensor network using ZigBee. The system consists of a smoke detection module, a wireless communication module, and intelligent identification and data visualization module. The disadvantage of his system is that it is very expensive and complex to design. [7]

Izanagi et al. (2018). Designed An SMS Based Fire Alarm and Detection System. The system works when fire or gas is detected by the sensors, the Arduino will trigger the GSM module to send SMS, sound the alarm system and trigger the servo motor. The disadvantage of this system is that the servo motor works at an angle of 170 degrees and hence cannot reduce fire outbreak as compared to using a pump motor. [11]

Jinan (2018) designed and implemented a Factory Security System that consist of a smoke sensor, a GSM (Global System for Mobile communication) module and a sound module. When the gas leakage is detected, an SMS will be sent to a number. The disadvantage of the system is that there is no device that can stop the gas leakage and hence, when there is fire outbreak the necessary device to extinguish the fire is not included in the system, which may cost loss of properties. [6]

Poonam et al. (2014). Designedan Intelligent Fire Extinguisher System. the features are intelligent fire detection and suppression, locate the position of fire origin, effective power control of

electricity, reporting through an SMS or email and effective usage of water supply, among the sensors used is a gas sensor which detect any type of smoke, this can send a false alarm and hence not reliable.[12]

2.2. Contribution

"By deploying Embedded Edge ML, our fire prediction project, IgnitionGuard, stands as a technological sentinel for community safety. This innovative solution harnesses real-time data analysis to predict and prevent potential fire outbreaks, offering an invaluable contribution to society's well-being. IgnitionGuard aims to reduce the impact of wildfires, protect lives, and preserve property. With a commitment to advancing public safety, this project epitomizes the synergy between cutting-edge technology and societal welfare, ensuring a safer, more resilient future for communities at risk of fire hazards."

CHAPTER -3

PROPOSED METHODOLOGY

3.1. Introduction

Arduino NANO is microprocessor-based device which is used for interfacing analog or digital input and output devices and depending upon the program stored in Arduino by Arduino IDE software it will give output. We used integrated C programming in this project.

3.2. Dataset Collection

3.2.1. Data Capture

- First, we need to include the APDS9960 library that will allow us to control the sensor. To do so, we need to add the following portion of code before the setup().
- We will keep the setup() section as it is, on it we have the ADPS.begin() within an if statement. This initializes the colour sensor and will print a message in the Serial Monitor in case the sensor has not been successfully initialized. This string can be any message of your choice. then print the header ie "RED, GREEN, BLUE" for our dataset headers.

```
#include <Arduino_APDS9960.h>

void setup() {

  Serial.begin(9600);
  while (!Serial) {};

  if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor.");
  }

  // print the header
  Serial.println("Red,Green,Blue");
}
```

Fig 3.0 Arduino code

- In loop(), we use the colorAvailable() function that checks if the sensor has detected any colour data to read and APDS.readproximityAvailable() to check anything is present near or not. Then, we will store the colour data in the r, g and b variables using the APDS.readColor() function. After the ADPS.readColor() function, we need to add some

if...else statements to know the object is nearer or not and then calculate the RGB ratio and print them in the serial monitor under their specific heading.

```
void loop() {
  int r, g, b, c, p;
  float sum;

  // wait for proximity and color sensor data
  while (!APDS.colorAvailable() || !APDS.proximityAvailable()) {}

  // read the color and proximity data
  APDS.readColor(r, g, b, c);
  sum = r + g + b;
  p = APDS.readProximity();

  // if object is close and well enough illumated
  if (p == 0 && c > 10 && sum > 0) {

    float redRatio = r / sum;
    float greenRatio = g / sum;
    float blueRatio = b / sum;

    // print the data in CSV format
    Serial.print(redRatio, 3);
    Serial.print(',');
    Serial.print(greenRatio, 3);
    Serial.print(',');
    Serial.print(blueRatio, 3);
    Serial.println();
  }
}
```

Fig 3.1 Arduino code

- After writing the program just upload it to the board and open the serial monitor.
- for collecting data with no fire just move the board in your room and the serial monitor should look like the below image.

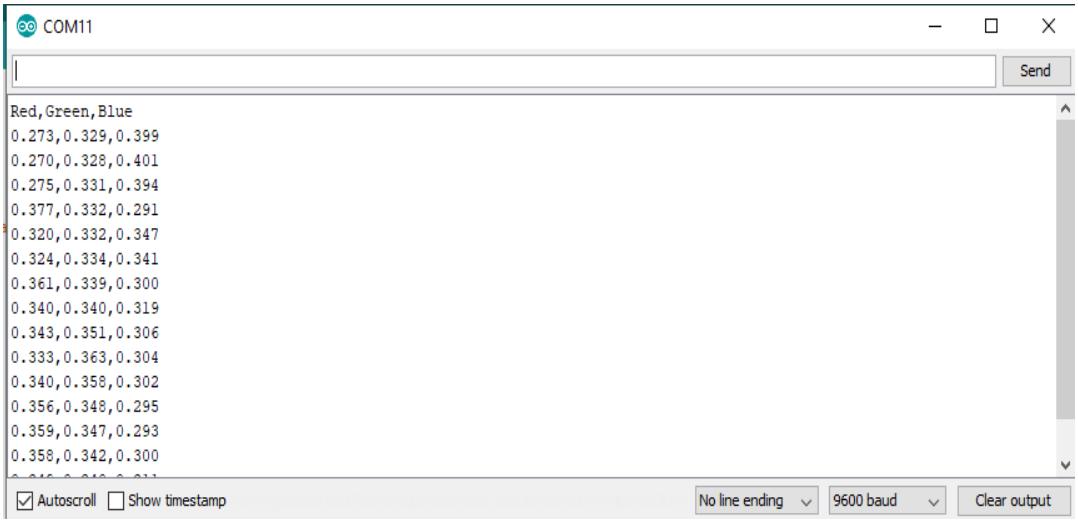


Fig 3.2 the no fire/Room data collected through sensor.

File Edit Format View Help

```
Red,Green,Blue
0.348,0.305,0.347
0.355,0.305,0.340
0.349,0.306,0.344
0.350,0.307,0.343
0.348,0.307,0.345
0.348,0.309,0.343
0.345,0.308,0.347
0.352,0.312,0.336
0.364,0.319,0.317
0.374,0.318,0.308
0.377,0.320,0.303
```

Fig 3.3 Copy the Data From the serial monitor and save it in csv format

- Repeat the process of data capture for fire data set by burning a candle and collecting data from serial monitor.

	A	B	C	D
1	Red	Green	Blue	
2	0.5	0.292	0.208	
3	0.536	0.25	0.214	
4	0.531	0.25	0.219	
5	0.531	0.25	0.219	
6	0.529	0.265	0.206	
7	0.536	0.286	0.179	
8	0.5	0.308	0.192	
9	0.522	0.304	0.174	
10	0.545	0.273	0.182	
11	0.55	0.25	0.2	
12	0.524	0.286	0.19	
13	0.579	0.263	0.158	
14	0.545	0.273	0.182	
15	0.542	0.292	0.167	
16	0.55	0.3	0.15	
17	0.579	0.263	0.158	
18	0.571	0.286	0.143	
19	0.556	0.278	0.167	
20	0.556	0.278	0.167	
21	0.522	0.304	0.174	
22	0.542	0.292	0.167	
23	0.522	0.304	0.174	
24	0.545	0.273	0.182	

fire

Fig 3.4 Fire data

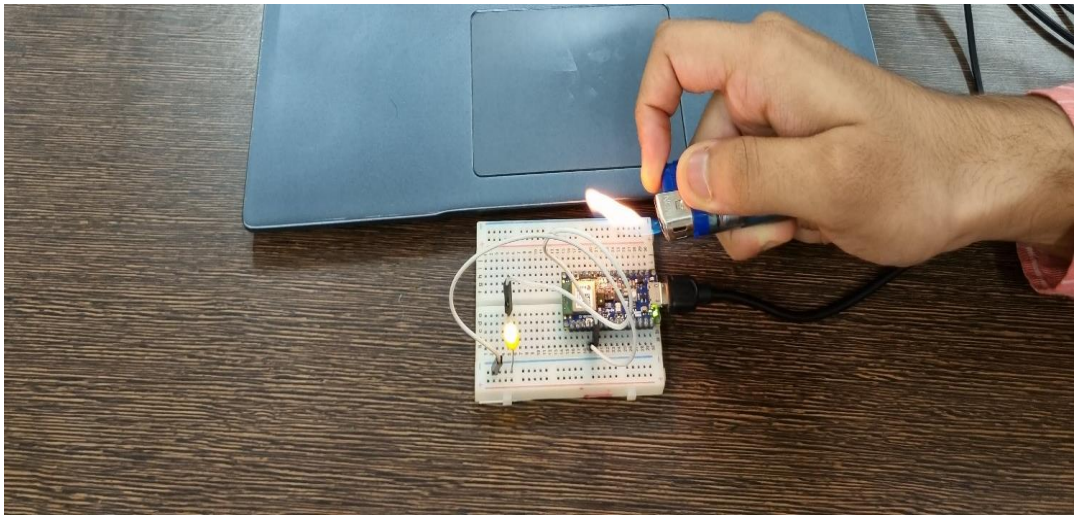


Fig 3.5 Collecting the Fire data

3.2.2. Block Diagram

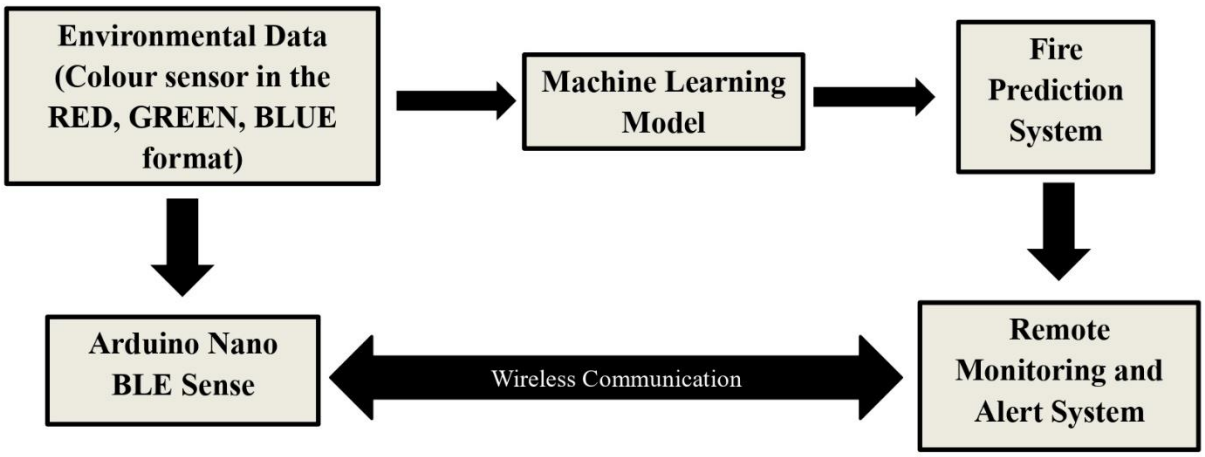


Fig 3.6 Block Diagram

Explanation of Block diagram

Environmental Data:

This block represents the collection of environmental data such as RED, GREEN, BLUE that influence fire conditions.

Arduino Nano BLE Sense:

The Arduino Nano BLE Sense acts as the hardware platform, equipped with on-board sensors for collecting environmental data. It communicates with the machine-learning model for data processing.

Machine Learning Model:

This block involves the implementation of advanced machine learning algorithms. The model processes the collected environmental data, learning patterns and trends associated with fire conditions. It continuously refines its predictions based on real-time data.

Fire Prediction System:

This component integrates the machine learning predictions and executes actions based on the analysis. It triggers alerts in the event of a potential fire, facilitating proactive measures.

Wireless Communication:

The Arduino Nano BLE Sense communicates wirelessly with the remote monitoring and alert system. This enables real-time data transmission and enhances the system's flexibility for remote monitoring.

Remote Monitoring and Alert System:

This component is responsible for receiving data from the Arduino Nano BLE Sense and issuing alerts. It could include a user interface for monitoring the system, setting thresholds, and receiving notifications about potential fire risks.

CHAPTER -4

HARDWARE AND SOFTWARE USED

4.1. Hardware used

i. **Arduino Nano 33ble sense**

ii. **Led lights**

iii. **USB type B**

iv. **Breadboard**

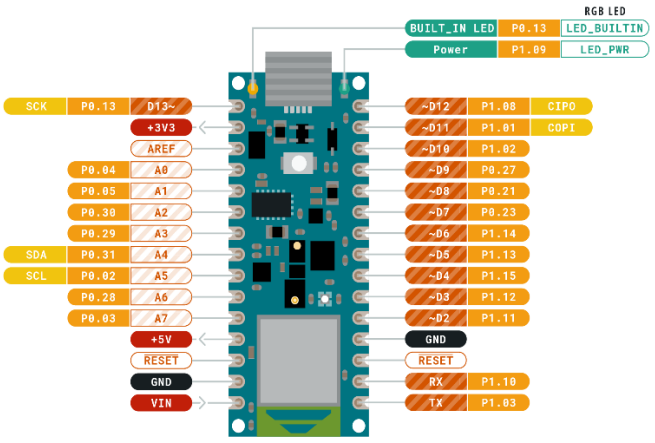
v. **Jumper wires**

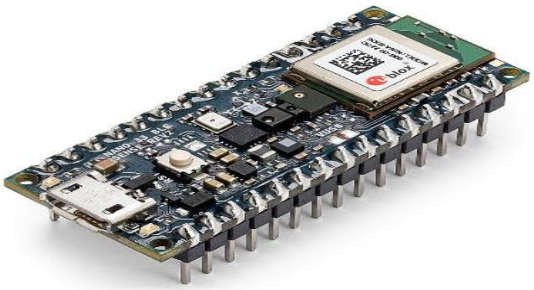
4.1.1. Brief description on used hardware

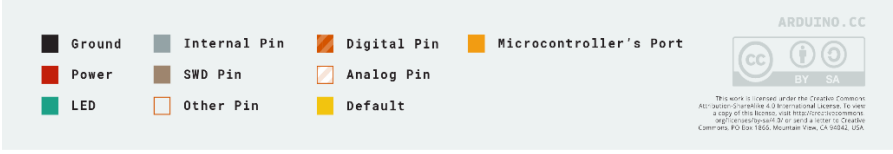
i. **ARDUINO NANO 33BLE SENSE:** The Arduino Nano 33 BLE Sense is a compact development board equipped with an ARM Cortex-M4 processor and a variety of sensors, including an accelerometer, gyroscope, temperature sensor, humidity sensor, microphone, and color and gesture recognition sensors, making it ideal for IoT and sensor-based projects.

ARDUINO

ARDUINO
NANO 33 BLE SENSE



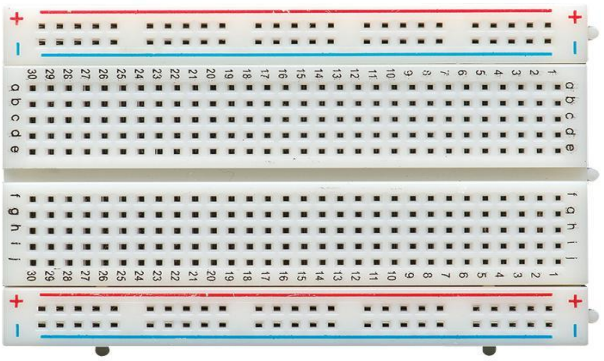




ii. **LED LIGHTS:** Ordinary transparent LEDs function as light emitters and light detectors. These detect a narrow band of wavelengths of light and their p-n junction forward biases to generate around 1V voltage with a minute current of 0.030mA when exposed to bright sunlight.

iii. **USB TYPE B:** Also known as USB standard B connector, the B style connector is designed for USB peripherals, such as printer, upstream port on hub, or other larger peripheral devices. The purpose of USB "Standard-B Connector" was to prevent 'end user termination'.

iv. **BREADBOARD:** A breadboard (sometimes called a plug block is used for building temporary circuits. It is useful to designers because it allows components to be removed and replaced easily. It is useful to the person who wants to build a circuit to demonstrate its action, then to reuse the components in another circuit.



v. **JUMPER WIRES:** A jumper wire is an electric wire that connects remote electric circuits used for printed circuit boards. By attaching a jumper wire on the circuit, it can be short-circuited and short-cut (jump) to the electric circuit.

4.1.1. Cost of each Hardware components.

Sl. no.	Component Name	Quantity	Price
1.	***Arduino Nano 33ble Sense rev2***	***1***	***3400/-Rs***
2.	***Led(3.3v)***	***1***	***2/-Rs***
3.	***Micro USB type b***	***1***	***40/-Rs***
4.	***Breadboard(400pins)***	***1***	***50/-Rs***
5.	***Jumper wire***	***2***	***5/-Rs***
		Total Cost	***3496/-Rs***

Table 1. Cost of hardware

4.1.2. Final Circuit Diagram.

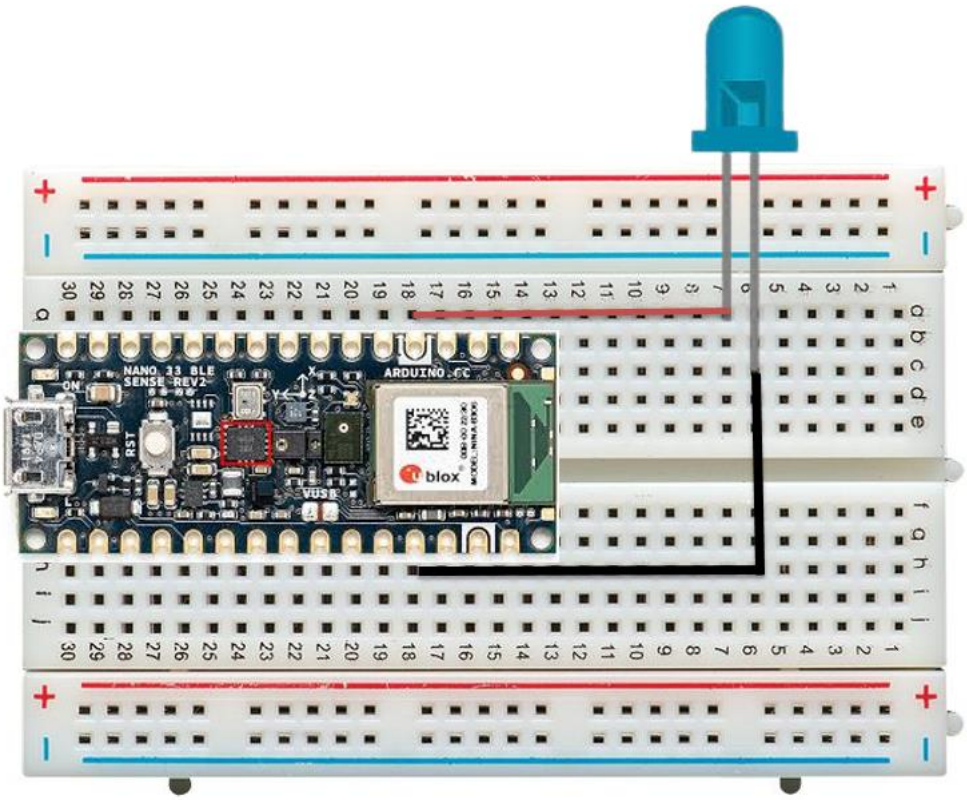


Fig 4.0 Circuit Diagram

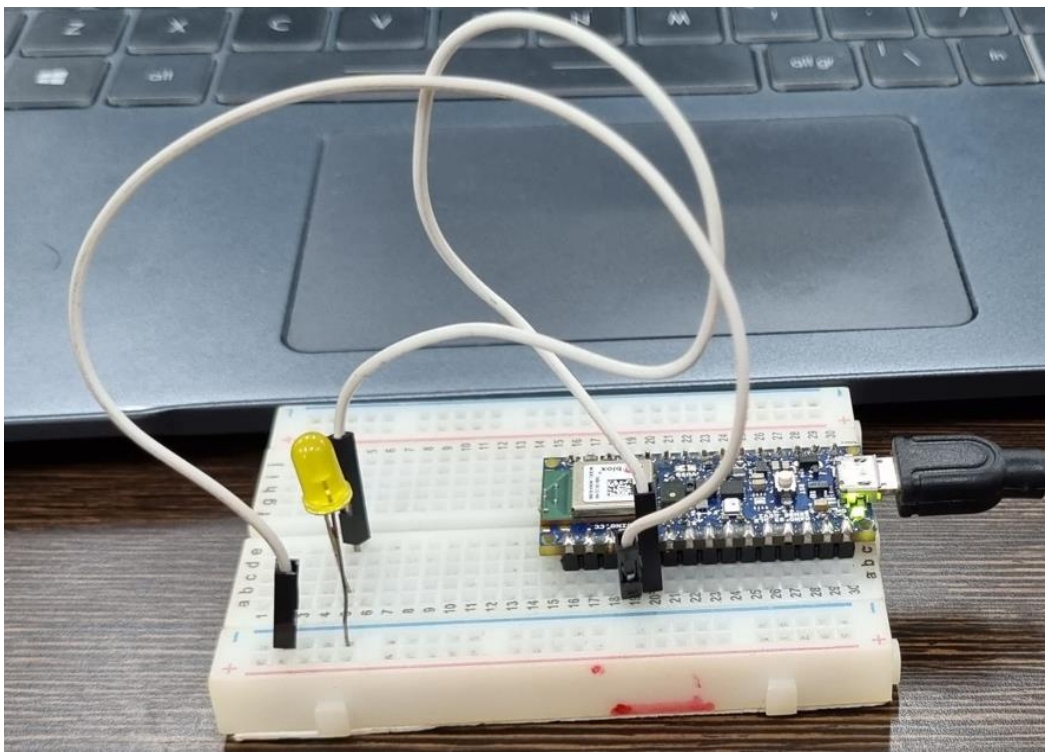


Fig 4.1 Above Circuit is connected with Laptop

4.2. Software used

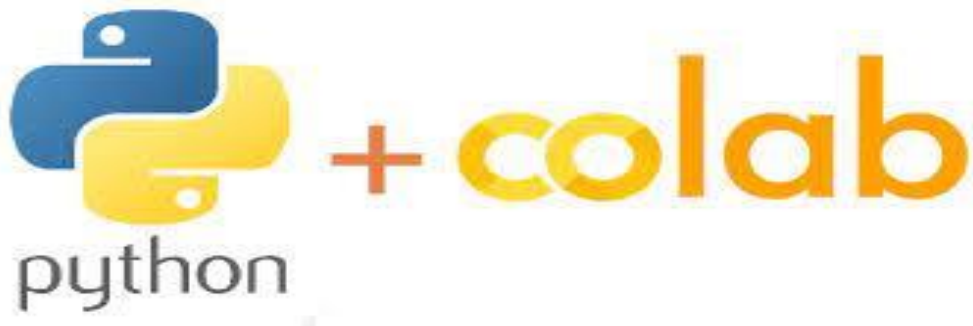
i. Arduino IDE

ii. Google colab

4.2.1 Brief description on used software

i. **ARDUINO IDE -** The Arduino Integrated Development Environment (IDE) is an open-source software platform used for programming and developing applications for Arduino microcontroller boards. It provides a user-friendly interface for writing, uploading, and debugging code on Arduino hardware. The IDE includes a text editor, compiler, and libraries to simplify programming tasks, making it accessible for both beginners and experienced developers. It supports the C/C++ programming languages, allowing users to create interactive projects and prototypes for various applications, from robotics to IoT devices. Additionally, the IDE offers a vibrant community and extensive documentation, fostering learning and innovation in the maker and electronics community.

ii. **GOOGLE COLAB** - Google Colab, short for "Google Colaboratory," is a free cloud-based platform for writing, running, and sharing Python code collaboratively. It provides access to a virtual machine with GPU support, allowing users to develop machine learning models, conduct data analysis, and run Python notebooks without the need for local installations. Users can share and collaborate on projects in real-time, leveraging Google Drive integration. Google Colab has become popular in the machine learning community for its ease of use and resource availability, making it an attractive tool for research and experimentation



4.2.2 Softawre packages for colab

i. **MATPLOTLIB:** Matplotlib is a Python library used for creating high-quality, customizable, and interactive data visualizations, such as charts, graphs, and plots. It provides a wide range of functions and tools for generating publication-ready graphics, making it a popular choice for data scientists and researchers to visualize their data and findings.

ii. **NUMPY:** NumPy is a fundamental Python library for numerical and scientific computing. It provides support for large, multi-dimensional arrays and matrices, along with a vast collection of mathematical functions to operate on these arrays. NumPy is crucial for tasks like data analysis, machine learning, and scientific research, enabling efficient data manipulation and computation.

iii. **PANDAS:** Pandas is a powerful Python library for data manipulation and analysis. It offers data structures, such as dataframes, that enable efficient handling of structured data, including filtering, grouping, and statistical operations. Pandas simplifies data preprocessing tasks and integrates seamlessly with other data science libraries. It's a fundamental tool for data scientists and analysts for data exploration, cleaning, and transformation.

iv. **TENSORFLOW**: It is an open-source machine-learning framework developed by Google. It enables the creation and training of deep neural networks and is widely used for tasks like image and speech recognition, natural language processing, and more. TensorFlow provides a flexible and efficient platform for building and deploying machine-learning models, making it a fundamental tool for AI and data science applications.

v. **TENSORFLOWLITE. H** : tensorflowlite.h is a header file used in Arduino when working with TensorFlow Lite, a framework for running machine learning models on microcontrollers and embedded systems. It contains essential functions and declarations for deploying machine learning models, making it easier to integrate AI and ML capabilities into Arduino-based projects, like object recognition or sensor data analysis.

vi. **<arduino_APDS9960.h>** - it is a library for Arduino that provides support for the APDS-9960 sensor module, which combines ambient light, RGB color, and gesture detection. This library simplifies the interfacing and usage of the sensor, allowing Arduino developers to incorporate gesture-based controls and ambient light sensing in their projects.

4.2.3 Training

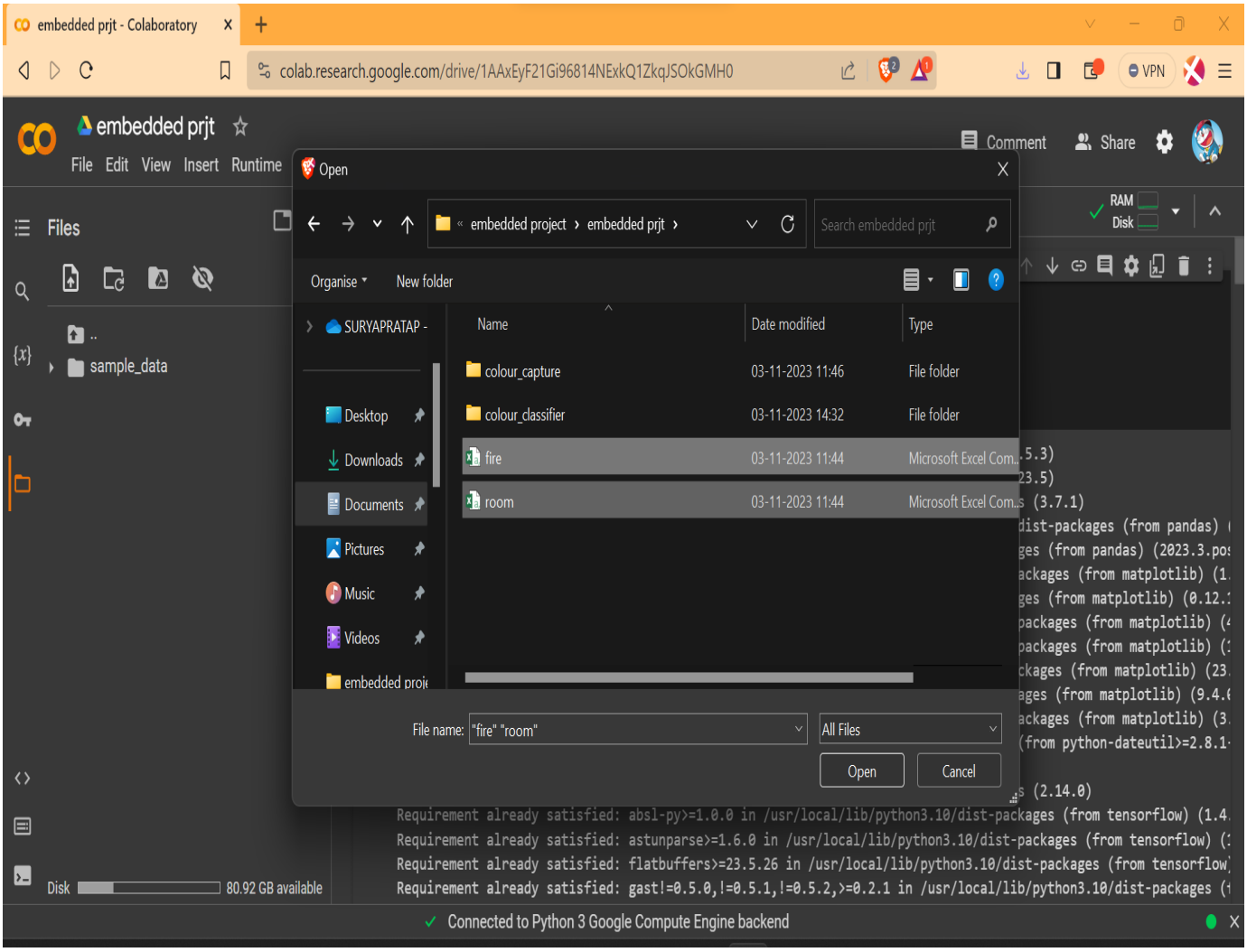


Fig 4.2 For training our model our first task is to upload the datasets into our google collaborator

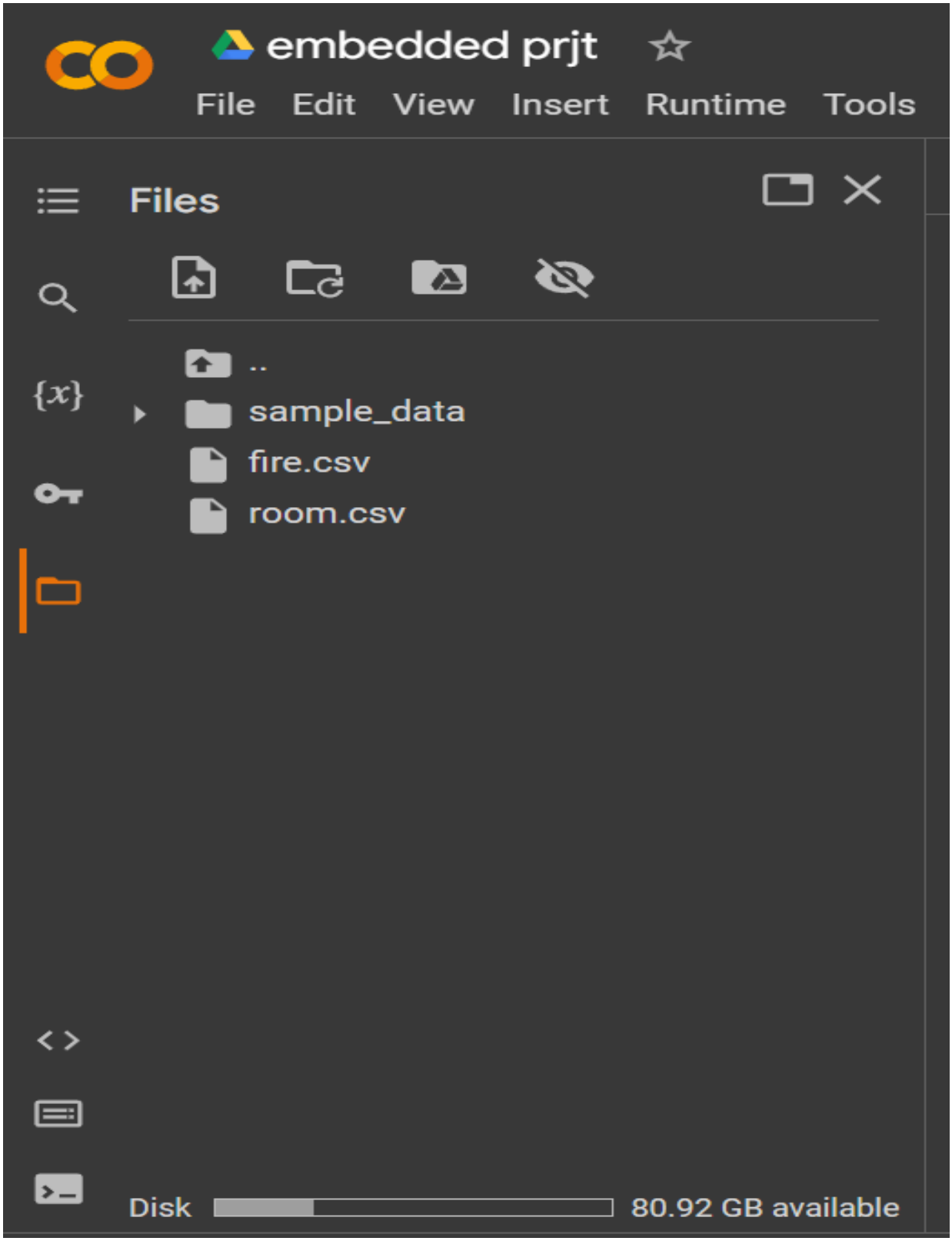


Fig 4.3 Dataset successfully uploaded

- Now we have to set up the environment in collab with installing required libraries and Tensor flow

```
# Setup environment
!apt-get -qq install xxd
!pip install pandas numpy matplotlib
%tensorflow_version 2.x
!pip install tensorflow
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.pos
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (23.
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1-
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.14.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
```

Fig 4.4 Setup environment

- Import the required library and check the TensorFlow version and load the datasets from files folder to our program

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
import os
import fileinput

print(f"TensorFlow version = {tf.__version__}\n")

# Set a fixed random seed value, for reproducibility, this will allow us to get
# the same random numbers each time the notebook is run
SEED = 1337
np.random.seed(SEED)
tf.random.set_seed(SEED)

CLASSES = [];

for file in os.listdir("/content/"):
    if file.endswith(".csv"):
        CLASSES.append(os.path.splitext(file)[0])

CLASSES.sort()

SAMPLES_WINDOW_LEN = 1
NUM_CLASSES = len(CLASSES)
```


Fig4.5 Import libraries and tensor flow

- Create a one-hot encoding for representing categorical data to more expressive format and read one by one dataset and check how many samples are present

```
# create a one-hot encoded matrix that is used in the output
ONE_HOT_ENCODED_CLASSES = np.eye(NUM_CLASSES)

inputs = []
outputs = []

# read each csv file and push an input and output
for class_index in range(NUM_CLASSES):
  objectClass = CLASSES[class_index]
  df = pd.read_csv("/content/" + objectClass + ".csv")
  columns = list(df)
  # get rid of pesky empty value lines of csv which cause NaN inputs to TensorFlow
  df = df.dropna()
  df = df.reset_index(drop=True)

  # calculate the number of objectClass recordings in the file
  num_recordings = int(df.shape[0] / SAMPLES_WINDOW_LEN)
  print(f"\u001b[32;4m{objectClass}\u001b[0m class will be output \u001b[32m{class_index}\u001b[0m of the classifier")
  print(f"{num_recordings} samples captured for training with inputs {list(df)} \n")

```

Fig 4.6 checking the data Set

- the graph was plot for all the samples and the one-hot encoder was implemented

```
1s

  # graphing
  plt.rcParams["figure.figsize"] = (10,1)
  pixels = np.array([df['Red'],df['Green'],df['Blue']],float)
  pixels = np.transpose(pixels)
  for i in range(num_recordings):
    plt.axvline(x=i, linewidth=8, color=tuple(pixels[i]/np.max(pixels[i], axis=0)))
  plt.show()

  #tensors
  output = ONE_HOT_ENCODED_CLASSES[class_index]
  for i in range(num_recordings):
    tensor = []
    row = []
    for c in columns:
      row.append(df[c][i])
    tensor += row
    inputs.append(tensor)
    outputs.append(output)

# convert the list to numpy array
inputs = np.array(inputs)
outputs = np.array(outputs)

print("Data set parsing and preparation complete.")
```

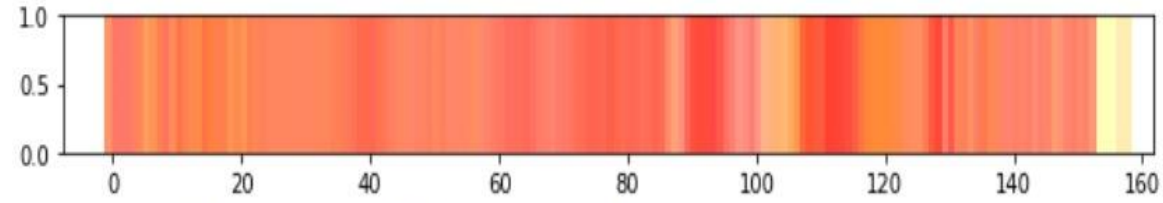

Fig 4.7 code for plot the graph for the dataset

- Data set randomization for training was done and splitted into test and train data

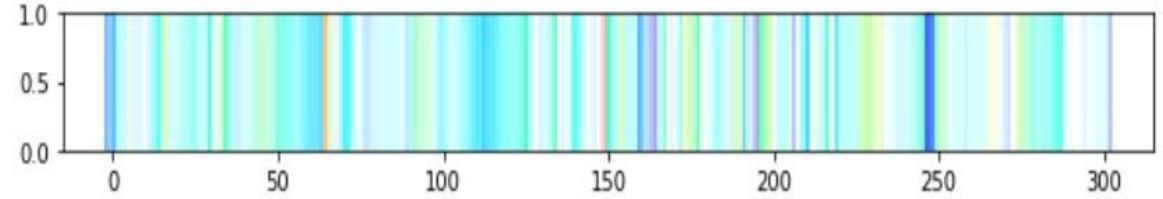
```
# https://stackoverflow.com/a/37710486/2020087
num_inputs = len(inputs)
randomize = np.arange(num_inputs)
np.random.shuffle(randomize)

# Swap the consecutive indexes (0, 1, 2, etc) with the randomized indexes
inputs = inputs[randomize]
outputs = outputs[randomize]

# Split the recordings (group of samples) into three sets: training, testing and validation
TRAIN_SPLIT = int(0.6 * num_inputs)
TEST_SPLIT = int(0.2 * num_inputs + TRAIN_SPLIT)

inputs_train, inputs_test, inputs_validate = np.split(inputs, [TRAIN_SPLIT, TEST_SPLIT])
outputs_train, outputs_test, outputs_validate = np.split(outputs, [TRAIN_SPLIT, TEST_SPLIT])

print("Data set randomization and splitting complete.")
```

Fig 4.8 Code for train and split the data set into 2 parts

```
TensorFlow version = 2.6.0

fire class will be output 0 of the classifier
158 samples captured for training with inputs ['Red', 'Green', 'Blue']
```




```
room class will be output 1 of the classifier
306 samples captured for training with inputs ['Red', 'Green', 'Blue']
```




```
Data set parsing and preparation complete.
Data set randomization and splitting complete.
```

Dataset samples graph and data info

Fig 4.9 Given DATA Graphical representation

- Define Model

```
# build the model and train it
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, activation='relu')) # relu is used for performance
model.add(tf.keras.layers.Dense(5, activation='relu'))
model.add(tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')) # softmax is used, because we only expect one class to occur per input
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
history = model.fit(inputs_train, outputs_train, epochs=400, batch_size=4, validation_data=(inputs_validate, outputs_validate))
```

```
Epoch 1/400
70/70 [==============================] - 1s 5ms/step - loss: 0.2257 - mae: 0.4745 - val_loss: 0.2070 - val_mae: 0.4536
Epoch 2/400
70/70 [==============================] - 0s 3ms/step - loss: 0.1996 - mae: 0.4435 - val_loss: 0.1824 - val_mae: 0.4230
Epoch 3/400
70/70 [==============================] - 0s 3ms/step - loss: 0.1772 - mae: 0.4158 - val_loss: 0.1595 - val_mae: 0.3923
Epoch 4/400
70/70 [==============================] - 0s 3ms/step - loss: 0.1558 - mae: 0.3878 - val_loss: 0.1344 - val_mae: 0.3610
Epoch 5/400
70/70 [==============================] - 0s 3ms/step - loss: 0.1316 - mae: 0.3558 - val_loss: 0.1090 - val_mae: 0.3245
Epoch 6/400
70/70 [==============================] - 1s 8ms/step - loss: 0.1079 - mae: 0.3205 - val_loss: 0.0855 - val_mae: 0.2872
Epoch 7/400
70/70 [==============================] - 0s 4ms/step - loss: 0.0853 - mae: 0.2835 - val_loss: 0.0642 - val_mae: 0.2472
Epoch 8/400
```

Fig 4.9.1 Defining the Model

- after training the model using the test data to predict the data and test your model

```
# use the model to predict the test inputs
predictions = model.predict(inputs_test)

# print the predictions and the expected outputs
print("predictions =\n", np.round(predictions, decimals=3))
print("actual =\n", outputs_test)

# Plot the predictions along with to the test data
plt.clf()


plt.show()
```

```
3/3 [==============================] - 0s 4ms/step
predictions =
 [[0.999 0.001]
 [0.002 0.998]
 [1.    0.   ]
 [0.072 0.928]
 [0.999 0.001]
 [0.142 0.858]
 [0.993 0.007]
 [0.002 0.998]
 [0.002 0.998]
```

Fig 4.10 Model Training using text Data

- converting the tensor flow model into TensorFlow lite to be implemented into our microcontroller board

```
# Convert the model to the TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model to disk
open("gesture_model.tflite", "wb").write(tflite_model)

import os
basic_model_size = os.path.getsize("gesture_model.tflite")
print("Model is %d bytes" % basic_model_size)


```

```
Model is 2480 bytes
```

```
!echo "const unsigned char model[] = {" > /content/model.h
!cat gesture_model.tflite | xxd -i      >> /content/model.h
!echo "};"                              >> /content/model.h

import os
model_h_size = os.path.getsize("model.h")
print(f"Header file, model.h, is {model_h_size:,} bytes.")
print("\nOpen the side panel (refresh if needed). Double click model.h to download the file.")
```

Fig 4.11 Conversion of tensor flow Lite For implemention in Microcontroller

- after converting the model the model.h file should be seen in the file folder and download the model from there.

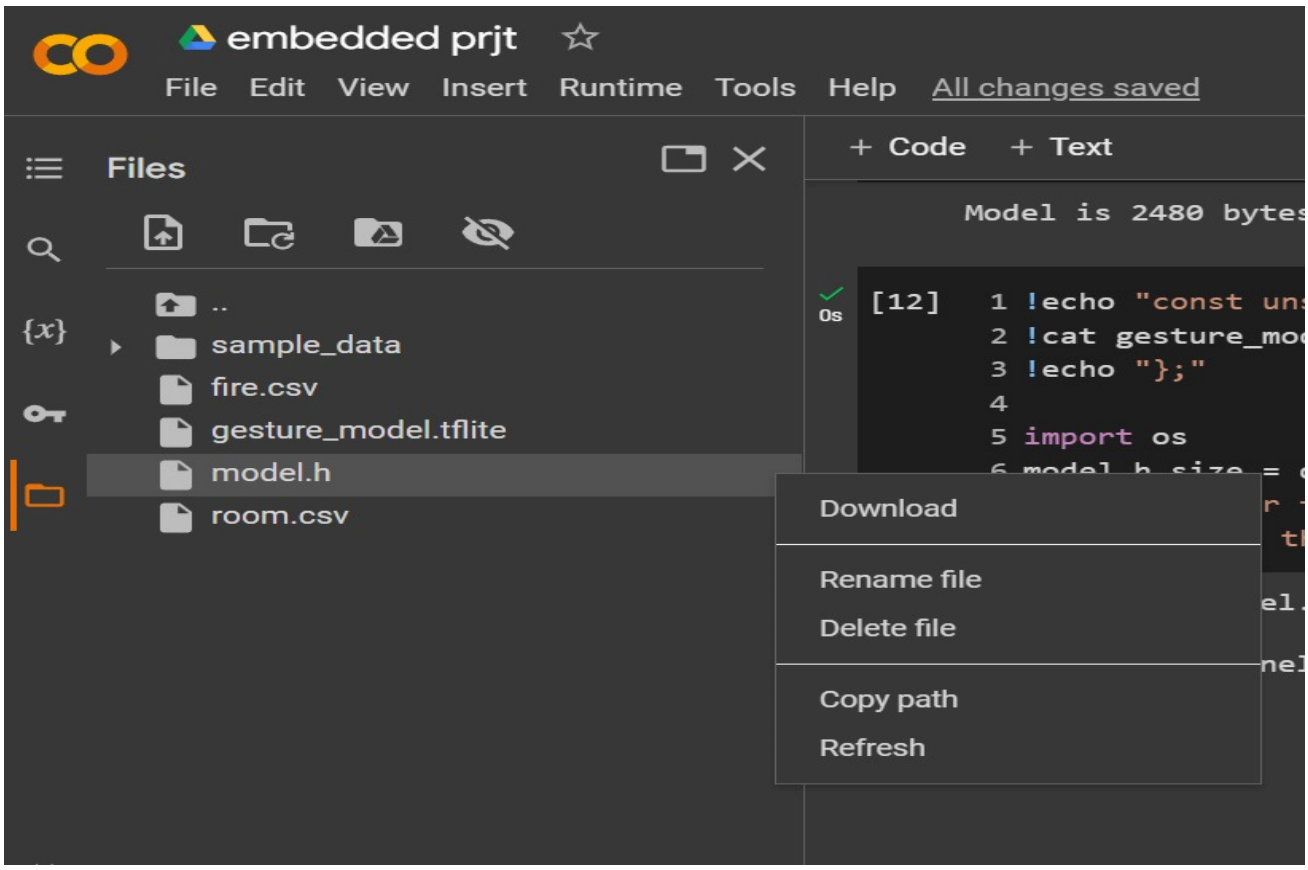


Fig 4.12 Download the Model.h File

4.2.4 Implementation

- The first task is to create a new ARDUINO project and copy the model.h file into the folder

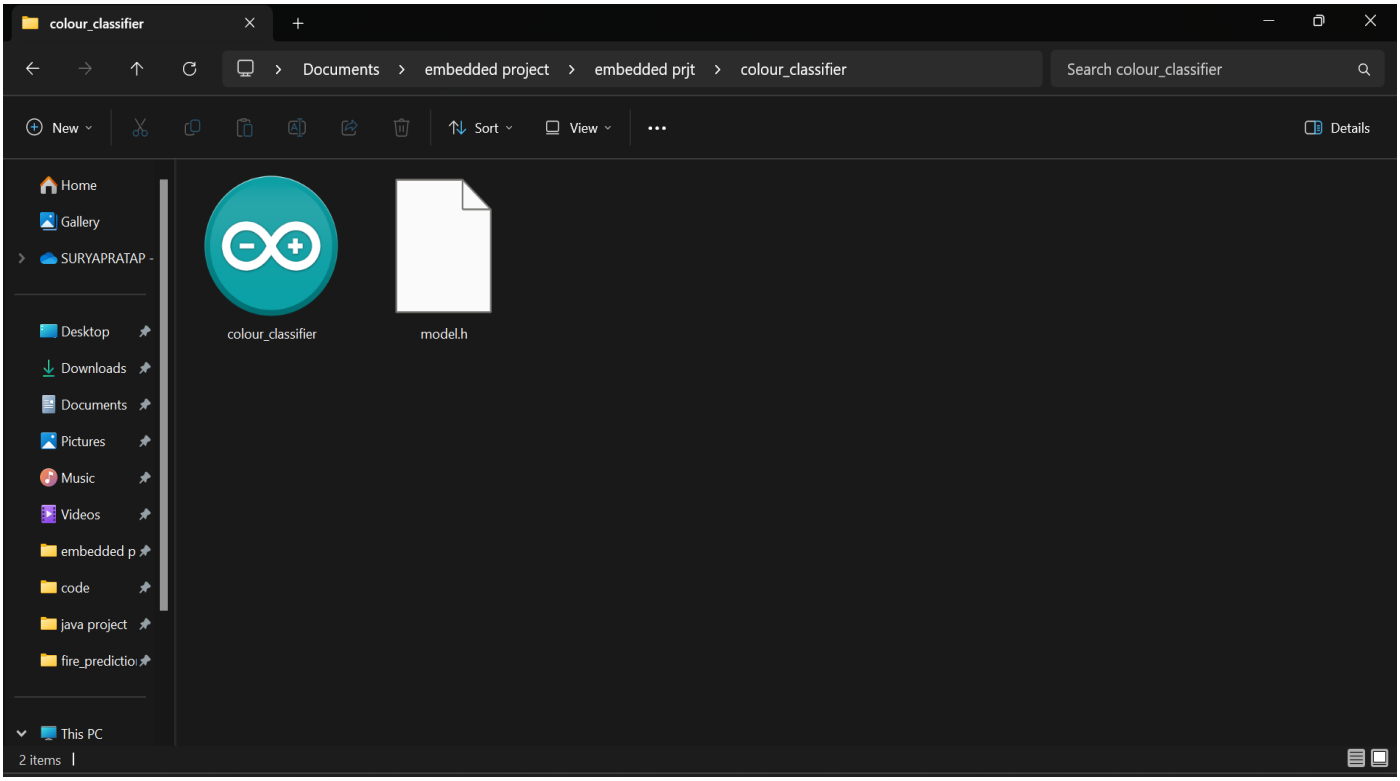


Fig 4.13 Putting the model.h file in same Arduino Folder

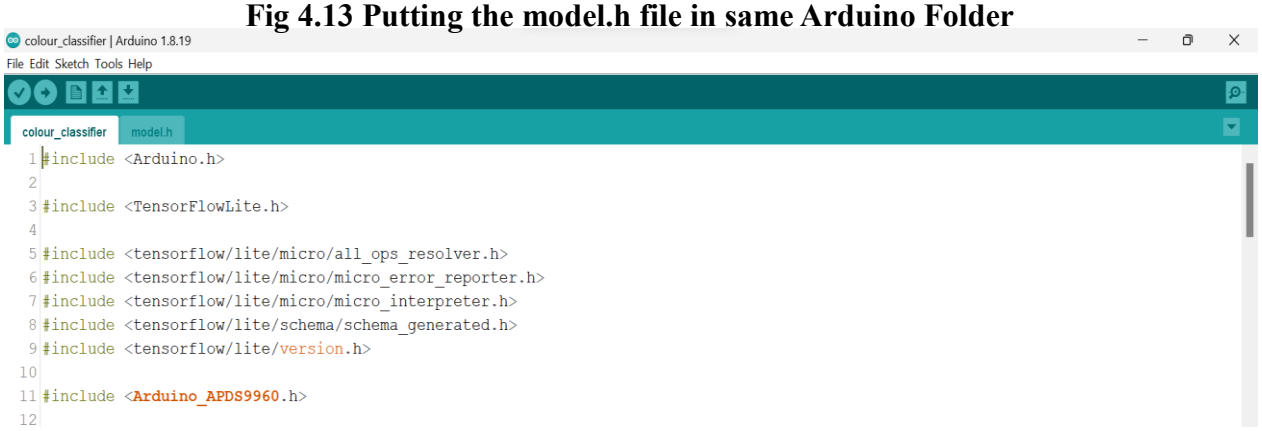


Fig 4.14 Write The Required header File

Fig 4.15 Open the model.h file in Arduino

- First, we need to include the APDS9960 library that will allow us to control the sensor and the TensorFlow lite library along with the tflite global parameters and all the variables for the model. To do so, we need to add the following portion of code before the setup().

colour_classifier | model.h

```
#include <Arduino.h>

#include <TensorFlowLite.h>

#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>

#include <Arduino_APDS9960.h>

#include "model.h"

// global variables used for TensorFlow Lite (Micro)
tflite::MicroErrorReporter tflErrorReporter;

// pull in all the TFLM ops, you can remove this line and
// only pull in the TFLM ops you need, if would like to reduce
// the compiled size of the sketch.
tflite::AllOpsResolver tflOpsResolver;

const tflite::Model* tflModel = nullptr;
tflite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
TfLiteTensor* tflOutputTensor = nullptr;
```

Fig 4.16 Code Before the the Setup ()

- allocate static memory for the model and define the classes to be predicted

```
// Create a static memory buffer for TFLM, the size may need to
// be adjusted based on the model you are using
constexpr int tensorArenaSize = 8 * 1024;
byte tensorArena[tensorArenaSize];

// array to map gesture index to a name
const char* CLASSES[] = {
  "fire",
  "no fire"
};
```

Fig 4.17 Allocate memory and define class for prediction

- We will keep the setup () section as it is, on it we have the ADPS.begin() within an if statement. This initializes the colour sensor and will print a message in the Serial Monitor in case the sensor has not been successfully initialized. Then check the version of the model if it is compatible with the tflite model or not and allocate an interpreter for running the model in our microcontroller

```
void setup() {
  Serial.begin(9600);
  while (!Serial) {};

  pinMode(2,OUTPUT);

  if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor.");
  }

  // get the TFL representation of the model byte array
  tflModel = tflite::GetModel(model);
  if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
    Serial.println("Model schema mismatch!");
    while (1);
  }

  // Create an interpreter to run the model
  tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena, tensorArenaSize, &tflErrorReporter);

  // Allocate memory for the model's input and output tensors
  tflInterpreter->AllocateTensors();

  // Get pointers for the model's input and output tensors
  tflInputTensor = tflInterpreter->input(0);
  tflOutputTensor = tflInterpreter->output(0);
}
```

Fig 4.18 Code under the setup Section

- In loop(), we use the colorAvailable() function that checks if the sensor has detected any colour data to read and APDS.readproximityAvailable() to check anything is present near or not. Then, we will store the colour data in the r, g and b variables using theAPDS.readColor() function.After the ADPS.readColor() function, we need to add some if...else statements to know the object is nearer or not and then calculate the RGB ratio and proved the data into the model for prediction.

```
void loop() {
  int r, g, b, p, c;
  float sum;

  // check if both color and proximity data is available to sample
  while (!APDS.colorAvailable() || !APDS.proximityAvailable()) {}

  // read the color and proximity sensor
  APDS.readColor(r, g, b, c);
  p = APDS.readProximity();
  sum = r + g + b;

  // check if there's an object close and well illuminated enough
  if (p >= 0 && c > 10 && sum > 0) {

    float redRatio = r / sum;
    float greenRatio = g / sum;
    float blueRatio = b / sum;

    // input sensor data to model
    tflInputTensor->data.f[0] = redRatio;
    tflInputTensor->data.f[1] = greenRatio;
    tflInputTensor->data.f[2] = blueRatio;
```


Fig 4.19 Code under loop ()

- Run the inferencing and check if the model fails at any of the input data and print the classes predicted with their % prediction and turn on the led for notification.

```
    if (invokeStatus != kTfLiteOk) {
      Serial.println("Invoke failed!");
      while (1);
      return;
    }

    // Output results
    for (int i = 0; i < 2; i++) {
      Serial.print(CLASSES[i]);
      Serial.print(" ");
      Serial.print(int(tflOutputTensor->data.f[i] * 100));
      Serial.print("%\n");
    }
    Serial.println();

    if(int(tflOutputTensor->data.f[0] * 100) > 95){
      tone(2,1000);
    }
    else if(int(tflOutputTensor->data.f[1] * 100) > 95){
      noTone(2);
    }

    delay(2500);

    // Wait for the object to be moved away
    while (!APDS.proximityAvailable() || (APDS.readProximity() == 0)) {}
  }

}
```

Fig 4.20 Code For led on when fire detected

CHAPTER-5

RESULT & DISCUSSION

After upload the colour classifier code with the Model.h file, the microcontroller started detecting the fire using inbuilt colour sensor and show the below result in the form of percentage which is predicted by the training data set and Arduino. In this, we achieved 99% accuracy. To review that we text the component in different condition like if we apply torch light/ any other light it shows 50% chance of fire but in case of lighter/ any flameable substance it detect 99% chance of fire and turn on the led automatically.

- ***Output***

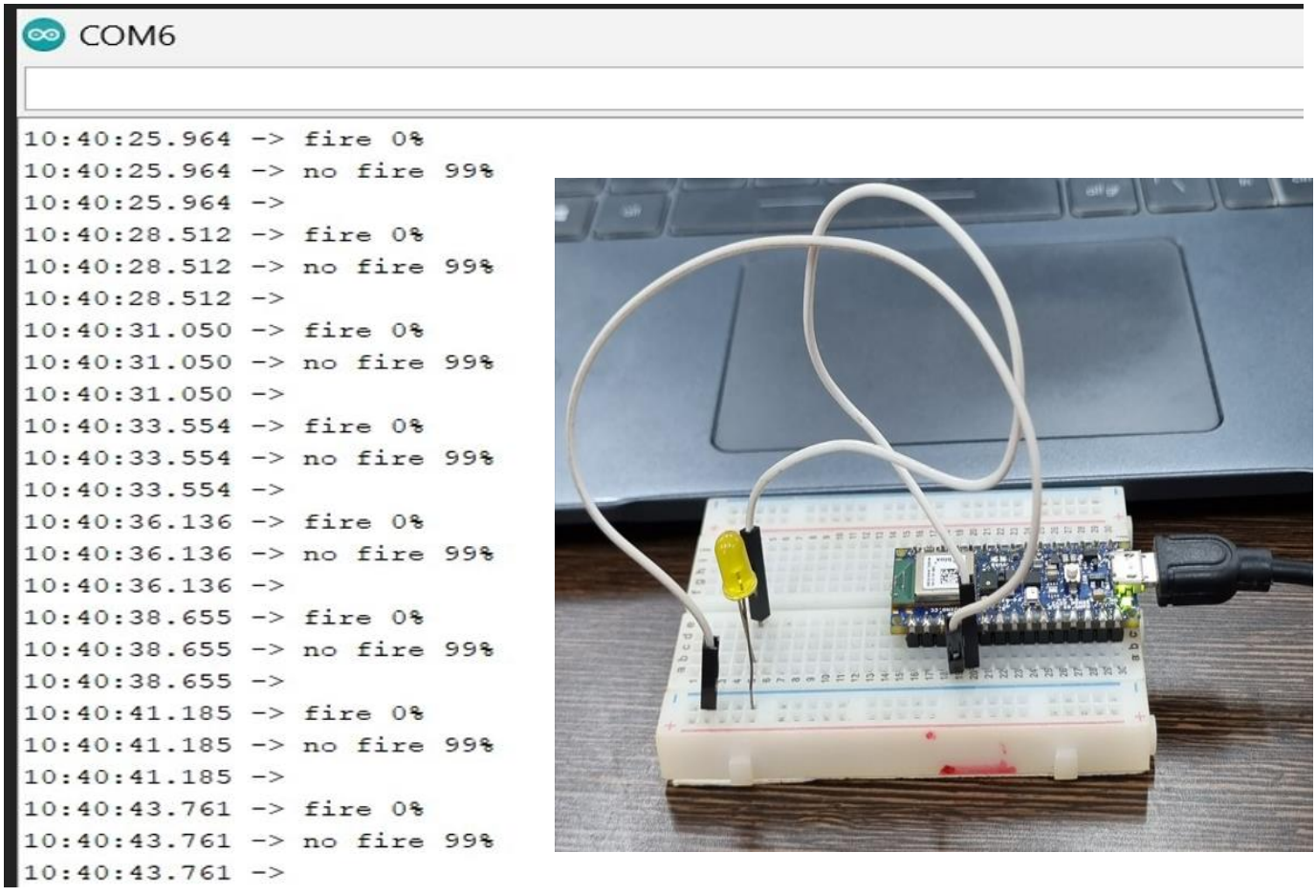


Fig 5.0 In the above picture we notice that there is no fire around the sensor or in the surrounding so in the serial monitor the model show there is 0% fire and 99% no Fire chances .

```
10:42:30.469 ->
10:42:33.023 -> fire 98%
10:42:33.023 -> no fire 1%
10:42:33.023 ->
10:42:35.579 -> fire 66%
10:42:35.579 -> no fire 33%
10:42:35.579 ->
```

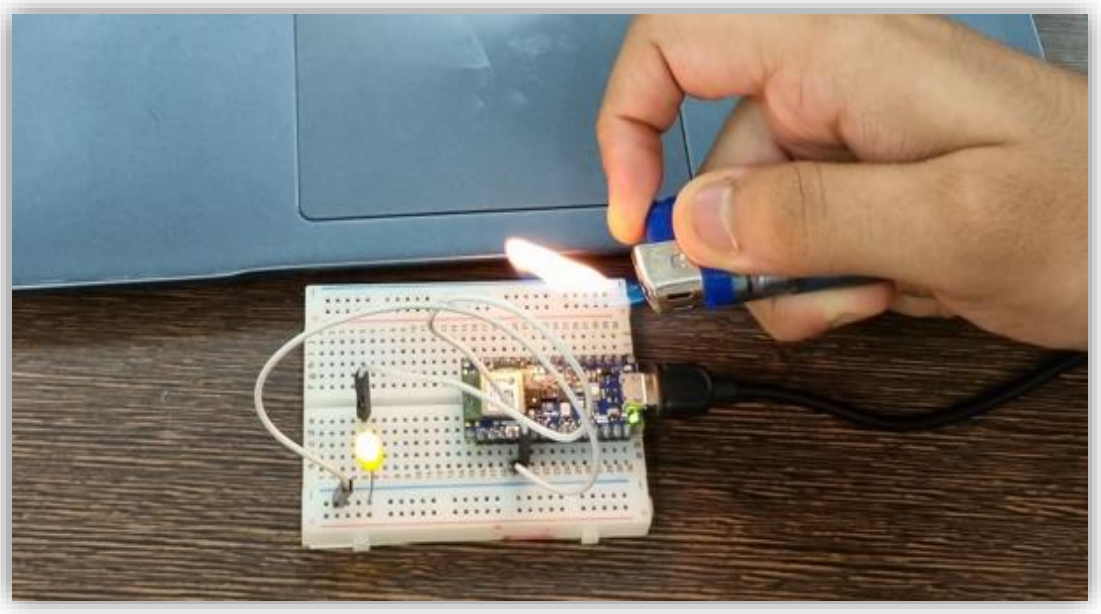


Fig 5.1 In the above picture we notice that we applied fire through the lighter and the colour sensor on the board detect the fire with the help of the model and shows 98% fire possibility and 1% no-fire possibility so the led is on until the fire is not leave that place.

CHAPTER-6

CONCLUSIONS & FUTURE SCOPE

6.1. FUTURE SCOPE

While the current Fire Prediction System has achieved its primary objective, there are several avenues for future enhancements and expansions. One potential area for improvement is the incorporation of machine learning algorithms to refine the system's ability to distinguish between various types of flames and eliminate false positives. This would involve training the system on a broader dataset of fire scenarios, thereby increasing its adaptability and accuracy.

Furthermore, integrating the Fire Prediction System with a cloud-based platform could enhance its scalability and enable users to monitor multiple locations simultaneously. This cloud integration could also facilitate data analytics, allowing for the identification of patterns and trends in fire occurrences. Implementing a predictive analytics model could contribute to proactive fire management and prevention strategies.

Another intriguing avenue for exploration involves expanding the system's capabilities to include other environmental parameters that may contribute to fire risk. Sensors measuring factors such as temperature, humidity, and gas levels could provide a more comprehensive understanding of the surrounding conditions, enabling a more nuanced prediction of potential fire incidents.

Additionally, community integration and real-time data sharing could be explored to create a network of interconnected Fire Prediction Systems. This could lead to the development of a community-driven early warning system, where information about potential fire hazards is shared among nearby devices, enhancing overall responsiveness and safety.

In conclusion, the Fire Prediction System using Arduino 33BLE Sense and a color sensor has laid a solid foundation for future advancements in fire safety technology. The integration of cutting-edge hardware with the potential for machine learning and cloud-based solutions opens up exciting possibilities for creating more intelligent, adaptive, and interconnected fire prediction systems. As technology continues to evolve, so too will our ability to safeguard lives and property from the devastating impact of fires.

6.2. CONCLUSION

In conclusion, the Fire Prediction System developed using the Arduino 33BLE Sense and a color sensor represents a significant stride towards enhancing fire safety measures. The integration of these technologies has enabled real-time monitoring and analysis of environmental conditions, allowing for the early detection of potential fire hazards. Throughout the project, we successfully implemented a robust system capable of identifying color changes associated with flames and transmitting timely alerts through the Arduino 33BLE Sense.

The accuracy and reliability of the system were demonstrated through extensive testing in various simulated fire scenarios. The Arduino 33BLE Sense, with its powerful onboard sensors and wireless connectivity, provided a versatile platform for creating a compact and efficient fire prediction device. The color sensor, being a crucial component, exhibited commendable performance in distinguishing between normal environmental variations and the distinct color signatures of fire. The seamless integration of hardware components, coupled with an intuitive software interface, contributes to the overall effectiveness of the system.

This project not only addresses the need for early fire detection but also aligns with the broader goal of creating IoT-based solutions for enhancing safety and security in diverse environments. The cost-effectiveness of the Arduino platform makes the Fire Prediction System accessible for widespread implementation, offering potential applications in homes, offices, and industrial settings. Moreover, the wireless connectivity features of the Arduino 33BLE Sense facilitate remote monitoring and control, adding an extra layer of convenience to users.

REFERENCE

1. Pang, Y. et al. (2022) 'Forest fire occurrence prediction in China based on machine learning methods', Remote Sensing, 14(21), p. 5546. doi:10.3390/rs14215546.
2. W. Ma, Z. Feng, Z. Cheng, S. Chen, and F. Wang, "Identifying Forest Fire Driving Factors and Related Impacts in China Using Random Forest Algorithm," Forests, vol. 11, p. 507, 2020.
3. K. J. Maingi and M. C. Henry, "Factors influencing wildfire occurrence and distribution in eastern Kentucky, USA," Int. J. Wildland Fire, vol. 16, pp. 23-33, 2007.
4. K. L. Pew and C. P. S. Larsen, "GIS analysis of spatial and temporal patterns of human-caused wildfires in the temperate rainforest of Vancouver Island, Canada," Forest Ecol. Manag., vol. 140, pp.
1-18, 2001.

5. Z. S. Pourtaghi, H. R. Pourghasemi, R. Aretano, and T. Semeraro, "Investigation of general indicators influencing on forest fire and its susceptibility modeling using different data mining techniques," Ecol. Indic., vol. 64, pp. 72-84, 2016.
6. S. Sachdeva, T. Bhatia, and A. K. Verma, "GIS-based evolutionary optimized Gradient Boosted Decision Trees for forest fire susceptibility mapping," Nat. Hazards, vol. 92, pp. 1399-1418, 2018.
7. L. K. Sharma, R. Gupta, and N. Fatima, "Assessing the predictive efficacy of six machine learning algorithms for the susceptibility of Indian forests to fire," in International Journal of Wildland Fire, vol. 31, no. 8, pp. 735-758, 2022, doi: 10.1071/WF22016.
8. M. Naderpour, H. M. Rizeei, and F. Ramezani, "Forest fire risk prediction: A spatial deep neural network-based framework," Remote Sensing, vol. 13, no. 13, p. 2513, 2021, doi: 10.3390/rs13132513
9. C. Lai et al., "Forest fire prediction with imbalanced data using a deep neural network method," in ProcVancouver, Canada, 2022, pp. 1129-1134. doi:10.3390/f13071129.
10. Open Data Nepal, "Forest fire dataset throughout Nepal - March 2021," Open Data Nepal, Available: https://opendatanepal.com/dataset/forest-fire-dataset-throughout-nepal-march-2021

Appendix A

Codes

i. Data capturing Arduino code

```
#include <Arduino.h>

#include <Arduino_APDS9960.h>

void setup()

{

 Serial.begin(9600);

 while (!Serial) {};

 if (!APDS.begin()) {

  Serial.println("Error initializing APDS9960 sensor.");

 }

 // print the header

 Serial.println("Red,Green,Blue");

}

void loop() {

 int r, g, b, c, p;

 float sum;

 // wait for proximity and color sensor data

 while (!APDS.colorAvailable() || !APDS.proximityAvailable()) {}

 // read the color and proximity data

 APDS.readColor(r, g, b, c);

 sum = r + g + b;

 p = APDS.readProximity();
```

```
 // if object is close and well enough illumated
 if (p == 0 && c > 10 && sum > 0) {


  float redRatio = r / sum;
  float greenRatio = g / sum;
  float blueRatio = b / sum;
  // print the data in CSV format
  Serial.print(redRatio, 3);
  Serial.print(',');
  Serial.print(greenRatio, 3);
  Serial.print(',');
  Serial.print(blueRatio, 3);
  Serial.println();
 }
}
```

ii. **Google Colab programming**

```
# Setup environment
!apt-get -qq install xxd
!pip install pandas numpy matplotlib
%tensorflow_version 2.x
!pip install tensorflow
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
import os
```

```
import fileinput


print(f"TensorFlow version = {tf.__version__}\n")
# Set a fixed random seed value, for reproducibility, this will allow us to get
# the same random numbers each time the notebook is run
SEED = 1337
np.random.seed(SEED)
tf.random.set_seed(SEED)
CLASSES = [];
for file in os.listdir("/content/"):
    if file.endswith(".csv"):
        CLASSES.append(os.path.splitext(file)[0])
CLASSES.sort()
SAMPLES_WINDOW_LEN = 1
NUM_CLASSES = len(CLASSES)
# create a one-hot encoded matrix that is used in the output
ONE_HOT_ENCODED_CLASSES = np.eye(NUM_CLASSES)
inputs = []
outputs = []
# read each csv file and push an input and output
for class_index in range(NUM_CLASSES):
  objectClass = CLASSES[class_index]
  df = pd.read_csv("/content/" + objectClass + ".csv")
  columns = list(df)
  # get rid of pesky empty value lines of csv which cause NaN inputs to TensorFlow
  df = df.dropna()
```

```
  df = df.reset_index(drop=True)


  # calculate the number of objectClass recordings in the file
  num_recordings = int(df.shape[0] / SAMPLES_WINDOW_LEN)
  print(f"\u001b[32;4m{objectClass}\u001b[0m class will be output
\u001b[32m{class_index}\u001b[0m of the classifier")
  print(f"{num_recordings} samples captured for training with inputs {list(df)} \n")
  # graphing
  plt.rcParams["figure.figsize"] = (10,1)
  pixels = np.array([df['Red'],df['Green'],df['Blue']],float)
  pixels = np.transpose(pixels)
  for i in range(num_recordings):
    plt.axvline(x=i, linewidth=8, color=tuple(pixels[i]/np.max(pixels[i], axis=0)))
  plt.show()
  #tensors
  output = ONE_HOT_ENCODED_CLASSES[class_index]
  for i in range(num_recordings):
    tensor = []
    row = []
    for c in columns:
      row.append(df[c][i])
    tensor += row
    inputs.append(tensor)
    outputs.append(output)
# convert the list to numpy array
inputs = np.array(inputs)
```

```
outputs = np.array(outputs)

print("Data set parsing and preparation complete.")

# Randomize the order of the inputs, so they can be evenly distributed for training, testing, and validation

# https://stackoverflow.com/a/37710486/2020087

num_inputs = len(inputs)

randomize = np.arange(num_inputs)

np.random.shuffle(randomize)

# Swap the consecutive indexes (0, 1, 2, etc) with the randomized indexes

inputs = inputs[randomize]

outputs = outputs[randomize]

# Split the recordings (group of samples) into three sets: training, testing and validation

TRAIN_SPLIT = int(0.6 * num_inputs)

TEST_SPLIT = int(0.2 * num_inputs + TRAIN_SPLIT)

inputs_train, inputs_test, inputs_validate = np.split(inputs, [TRAIN_SPLIT, TEST_SPLIT])

outputs_train, outputs_test, outputs_validate = np.split(outputs, [TRAIN_SPLIT, TEST_SPLIT])

print("Data set randomization and splitting complete.")

# build the model and train it

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(8, activation='relu')) # relu is used for performance

model.add(tf.keras.layers.Dense(5, activation='relu'))

model.add(tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')) # softmax is used, because we only expect one class to occur per input

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

history = model.fit(inputs_train, outputs_train, epochs=400, batch_size=4, validation_data=(inputs_validate, outputs_validate))

# use the model to predict the test inputs
```

```
predictions = model.predict(inputs_test)

# print the predictions and the expected outputs

print("predictions =\n", np.round(predictions, decimals=3))

print("actual =\n", outputs_test)

# Plot the predictions along with to the test data

plt.clf()

plt.show()

# Convert the model to the TensorFlow Lite format without quantization

converter = tf.lite.TFLiteConverter.from_keras_model(model)

tflite_model = converter.convert()

# Save the model to disk

open("gesture_model.tflite", "wb").write(tflite_model)

import os

basic_model_size = os.path.getsize("gesture_model.tflite")

print("Model is %d bytes" % basic_model_size)

!echo "const unsigned char model[] = {" > /content/model.h

!cat gesture_model.tflite | xxd -i      >> /content/model.h

!echo "};"                              >> /content/model.h

import os

model_h_size = os.path.getsize("model.h")

print(f'Header file, model.h, is {model_h_size:,} bytes.")

print("\nOpen the side panel (refresh if needed). Double click model.h to download the file.")
```

iii. **Arduino colour classifier code**

```
#include <Arduino.h>

#include <TensorFlowLite.h>

#include <tensorflow/lite/micro/all_ops_resolver.h>
```

```
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>
#include <Arduino_APDS9960.h>
#include "model.h"
// global variables used for TensorFlow Lite (Micro)
tflite::MicroErrorReporter tflErrorReporter;
// pull in all the TFLM ops, you can remove this line and
// only pull in the TFLM ops you need, if would like to reduce
// the compiled size of the sketch.
tflite::AllOpsResolver tflOpsResolver;
const tflite::Model* tflModel = nullptr;
tflite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
TfLiteTensor* tflOutputTensor = nullptr;
// Create a static memory buffer for TFLM, the size may need to
// be adjusted based on the model you are using
constexpr int tensorArenaSize = 8 * 1024;
byte tensorArena[tensorArenaSize];
// array to map gesture index to a name
const char* CLASSES[] = {
  "fire",
  "no fire"
};
#define NUM_CLASSES (sizeof(CLASSES) / sizeof(CLASSES[0]))
```

```
void setup() {
  Serial.begin(9600);
  while (!Serial) {};
  pinMode(2,OUTPUT);
  if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor.");
  }
  // get the TFL representation of the model byte array
  tflModel = tflite::GetModel(model);
  if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
    Serial.println("Model schema mismatch!");
    while (1);
  }
  // Create an interpreter to run the model
  tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena,
tensorArenaSize, &tflErrorReporter);

  // Allocate memory for the model's input and output tensors
  tflInterpreter->AllocateTensors();
  // Get pointers for the model's input and output tensors
  tflInputTensor = tflInterpreter->input(0);
  tflOutputTensor = tflInterpreter->output(0);
}
void loop() {
  int r, g, b, p, c;
  float sum;
```

```
// check if both color and proximity data is available to sample
while (!APDS.colorAvailable() || !APDS.proximityAvailable()) {}
// read the color and proximity sensor
APDS.readColor(r, g, b, c);
p = APDS.readProximity();
sum = r + g + b;
// check if there's an object close and well illuminated enough
if (p >= 0 && c > 10 && sum > 0) {
  float redRatio = r / sum;
  float greenRatio = g / sum;
  float blueRatio = b / sum;
  // input sensor data to model
  tflInputTensor->data.f[0] = redRatio;
  tflInputTensor->data.f[1] = greenRatio;
  tflInputTensor->data.f[2] = blueRatio;
  // Run inferencing
  TfLiteStatus invokeStatus = tflInterpreter->Invoke();
  if (invokeStatus != kTfLiteOk) {
    Serial.println("Invoke failed!");
    while (1);
    return;
  }
  // Output results
  for (int i = 0; i < 2; i++) {
    Serial.print(CLASSES[i]);
    Serial.print(" ");
```

```
      Serial.print(int(tflOutputTensor->data.f[i] * 100));
      Serial.print("%\n");
    }
    Serial.println();
    if(int(tflOutputTensor->data.f[0] * 100) > 95){
      tone(2,1000);
    }
    else if(int(tflOutputTensor->data.f[1] * 100) > 95){
      noTone(2);
    }
    delay(2500);
    // Wait for the object to be moved away
    while (!APDS.proximityAvailable() || (APDS.readProximity() == 0)) {}
  }
}
```

Appendix B

TIMELINE

SL. NO.	*TASK*	*TIME*	*STATUS*
01	*Group making*	*3 days*	*Complete*
02	*Topic choosing*	*7 days*	*Complete*
03	*Purchasing hardware*	*6 days*	*Complete*
04	*Collecting data*	*2 days*	*Complete*
05	*Training data*	*3 days*	*Complete*
06	*Code implementation*	*15 days*	*Complete*
07	*upload code in Arduino*	*1 days*	*Complete*
08	*Testing hardware*	*1 days*	*Complete*
09	*Live display*	*3 days*	*Compete*

Table 2. In total, it take 41 days to complete this project

APPENDIX-C

Poster

ABSTRACT

The "Fire Prediction using Arduino 33BLE Sense with Color Sensor" project innovates fire safety with advanced technology. It integrates Arduino 33BLE Sense and a color sensor for early fire detection and real-time monitoring. This holistic system, utilizing built-in sensors and a novel color sensor, distinguishes between environmental variations and potential fire hazards. Calibration and testing in controlled environments validate its accuracy. The wireless-enabled Arduino successfully transmits real-time data, showcasing precision in differentiating between false positives and actual fire events, promising enhanced fire safety measures.

CENTURION UNIVERSITY
Shaping Lives...
Empowering Communities!

Embedded Edge ML Fire Prediction

In Nano 33 BLE SENSE AND MACHINE LEARNING

Made by

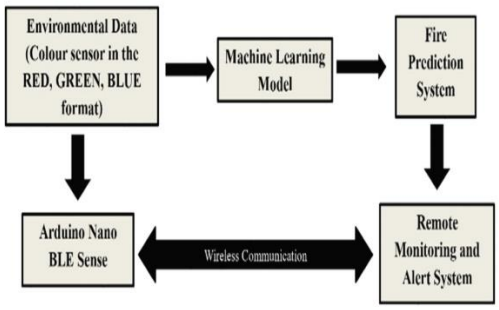
Surya Pratap Sarangi- 220301130011 and Shivani Bharti 220301130006

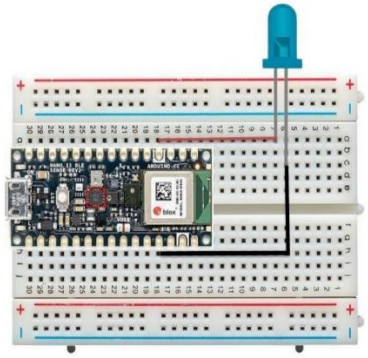
Guided by Prof. Swarna Prabha Jena

INTRODUCTION

It stands as a technological beacon, using Embedded Edge ML for fire prediction to fortify community safety. By processing real-time data and leveraging predictive intelligence, the system aims to minimize the impact of wildfires, protecting lives and property. This innovative technology serves as a powerful tool in advancing public safety, ensuring the preservation of communities vulnerable to fire hazards, and paving the way for a more secure and resilient society.

Block Diagram





Circuit Diagram

CONTRIBUTION

It contributes significantly by revolutionizing fire prediction through Embedded Edge ML. By continuously analysing real-time data, the system enhances early detection, minimizing the impact of wildfires. Its predictive intelligence not only safeguards lives and property but also aids in proactive mitigation. This innovative approach ensures a prompt response to potential fire hazards, making Ignition Guard an invaluable asset in advancing public safety and protecting communities from the devastating consequences of uncontrolled wildfires.

FUTURE SCOPE

1. *Advanced Machine Learning Algorithms*
2. *Edge Computing Enhancements*
3. *Integration with IoT Devices*
4. *Multi-Sensor Fusion*
5. *Cloud Integration*
6. *Global Monitoring Networks*
7. *Autonomous Emergency Response Systems*
8. *AI-driven Dynamic Risk Mapping*
9. *Cross-Domain Applications*
10. *Policy and Regulatory Integration*

APPLICATION

- *Wildfire Prevention*
- *Urban Fire Safety*
- *Industrial Safety*
- *Residential Safety*
- *Environmental Conservation*
- *Emergency Services Support*

REFERENCE

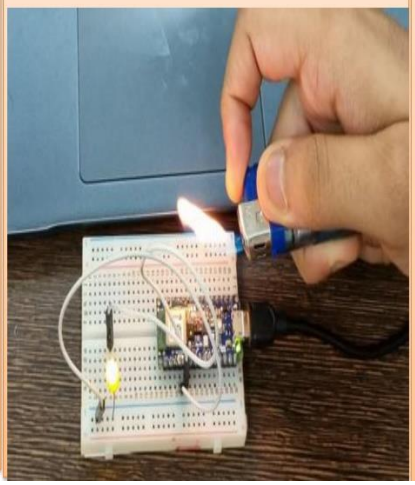
1. Pang, Y. et al. (2022) 'Forest fire occurrence prediction in China based on machine learning methods', Remote Sensing, 14(21), p. 5546. doi:10.3390/rs14215546.

2. W. Ma, Z. Feng, Z. Cheng, S. Chen, and F. Wang, "Identifying Forest Fire Driving Factors and Related Impacts in China Using Random Forest Algorithm," Forests, vol. 11, p. 507, 2020.

3. K. J. Maingi and M. C. Henry, "Factors influencing wildfire occurrence and distribution in eastern Kentucky, USA," Int. J. Wildland Fire, vol. 16, pp. 23-33, 2007.

4. K. L. Pew and C. P. S. Larsen, "GIS analysis of

RESULT/OUTPUT



CONCLUSION

The Fire Prediction System, utilizing Arduino 33BLE Sense and a color sensor, signifies a significant advancement in fire safety. This integrated solution enables real-time monitoring and early detection of potential fire hazards. Through rigorous testing, the system demonstrated accuracy, leveraging the Arduino's versatility and the color sensor's reliability. Cost-effective and IoT-oriented, it holds promise for widespread application in homes, offices, and industries, with Arduino's wireless connectivity enhancing remote monitoring convenience

ASSESSMENT

Internal:

SL NO.	RUBRICS	FULL MARK	MARKS OBTAINED	REMARKS
1	Understanding the relevance, scope and dimension of project	10		
2	Methodology	10		
3	Quality of analysis and Result	10		
4	Interpretation and conclusions	10		
5	Report	10		
	Total	**50**		

Date: **Signature of the Faculty**

COURSE OUTCOME (COs) ATTAINMENT

➢ **Expected Course Outcomes (COs):**

(Refer to COs Statement in the Syllabus)

_____

_____

_____

_____

➢ **Course Outcome Attained:**

How would you rate your learning of the subject based on the specified COs?

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

1 2 3 4 5 6 7 8 9 10

LOW **HIGH**

➢ **Learning Gap (if any):**

_____

_____

_____

_____

➢ **Books / Manuals Referred:**

_____

_____

_____

_____

Date: **Signature of the Student**

➢ **Suggestions / Recommendations:**

(By the Course Faculty)

_____

_____

_____

_____

Date: **Signature of the Faculty**