Summer Fellowship Report

On

Practical Machine Learning: Jupyter Notebook and ESP32

Submitted by

Harsh Raj

Shri Mata Vaishno Devi University, Katra, Jammu & Kashmir

Under the guidance of

Rajesh Kushalkar
Sr. Project Manager Open-Source Hardware
IIT Bombay

Prof. Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

August 5, 2024

Acknowledgment

First and foremost, I would like to express my sincere thanks to my project head, Prof. Kannan M. Moudgalya, who gave me this valuable opportunity to work in such a learning environment at FOSSEE.

I would like to extend my heartfelt gratitude to my project mentors, Mr. Rajesh Kushlkar, Sr. Project Manager, IIT Bombay and Mr. Pratik Nemane, project research assistant, IIT Bombay. Your guidance, constant supervision, and the wealth of knowledge and expertise you have imparted have been invaluable throughout this journey. Thank you for your unwavering support and encouragement.

Finally, I am deeply thankful to my parents and teachers who helped and inspired me throughout this fellowship.

Sincerely

Harsh Raj,
Shri Mata Vaishno Devi University, Katra, J&K

Contents

Chapter 1
Introduction

FOSSEE (Free/Libre and Open-Source Software for Education) is an initiative by the Indian Institute of Technology Bombay aimed at promoting the use of open source software in education. FOSSEE supports various open-source software and provides resources, workshops, and training programs to help students and professionals adopt and use these tools effectively. The initiative focuses on reducing the dependency on proprietary software and fostering a collaborative and inclusive learning environment.

1.1 Introduction to Machine Learning with Jupyter Notebooks

Welcome to the world of Machine Learning. These instructional materials will guide you to learn machine learning using Jupyter Notebook.

1.2 Getting Started

1.2.1 Setting Up Your Environment

Before we go deep into machine learning, you'll need to set up your environment. follow these steps to set up your environment.

Install the classic Jupyter Notebook by writing the below command in your terminal: pip install notebook

1.2.2. Launching Jupyter Notebook

To run the notebook: Type the below command on terminal jupyter notebook or if this doesn't work, write this instead. python -m notebook

1.2.3. Why choose Jupyter Notebook for ML?

Jupyter Notebook is a popular tool for machine learning due to its interactive environment that facilitates exploratory data analysis and visualization. It allows users to write and execute code in a step-by-step manner, making it easier to test and iterate on machine learning models. The ability to integrate code, visualizations, and narrative text in a single document enhances reproducibility and collaboration. Additionally, Jupyter's support for various programming languages and its extensive ecosystem of libraries and extensions make it a versatile choice for both beginners and experienced practitioners in machine learning.

1.3 Basics of Machine Learning (ML)

1.3.1 What is Machine Learning?

It is a subset of Artificial Intelligence (AI). It focuses on making a model capable of learning from data to make predictions or decisions.

1.3.2 Types of Machine Learning:

1. Supervised Learning: Learning from labelled data (e.g., classification, regression).
2. Unsupervised Learning: Finding patterns in unlabelled data (e.g., clustering, dimensionality reduction).
3. Reinforcement Learning: Learning through rewards and punishments.

Chapter 2
Task 1: Weather Prediction Model

2.1.1 Objective

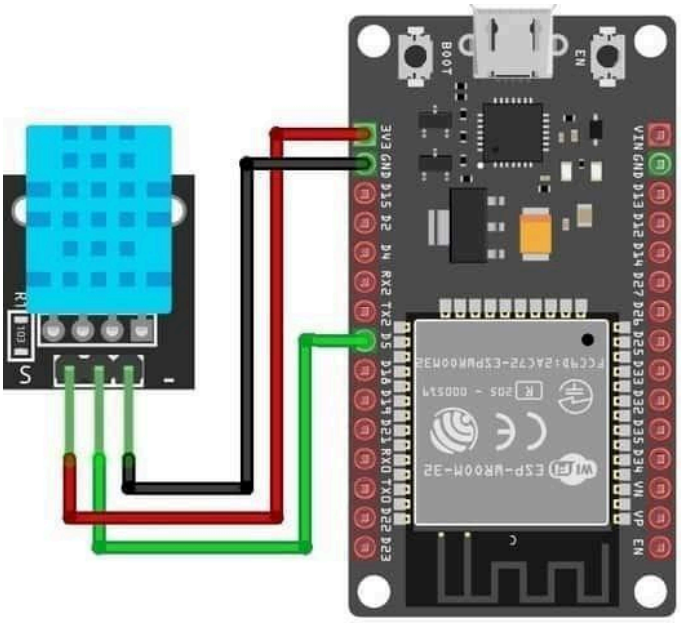
Develop a weather forecasting model capable of predicting weather conditions such as sunny, cloudy, rainy, and foggy.

2.1.2 Tools and Technologies

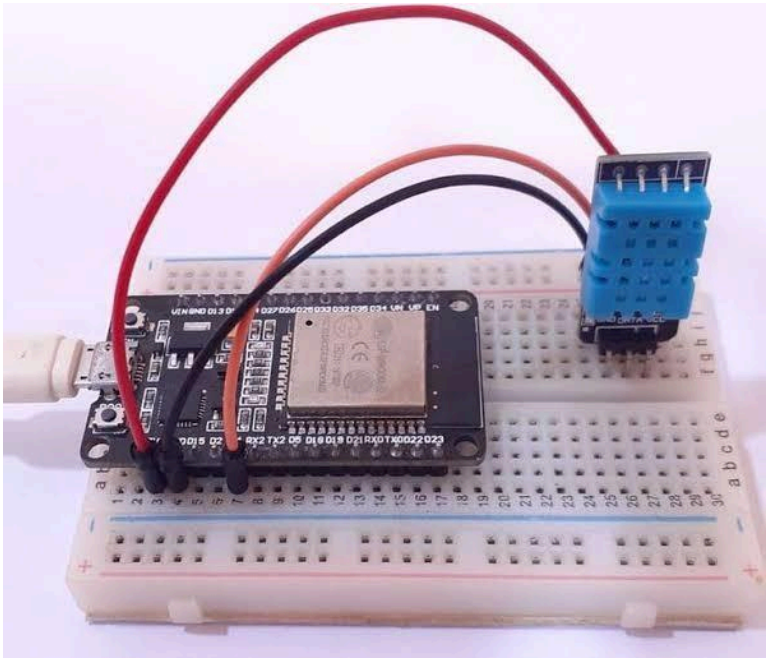
- Jupyter Notebook: For data preprocessing, model development, and training.
- Programming Language: Python.
- TensorFlow: For building and training the weather prediction model.
- NumPy: For numerical computations and data manipulation.
- Pandas: For data manipulation and analysis.

2.1.3 Hardware Setup

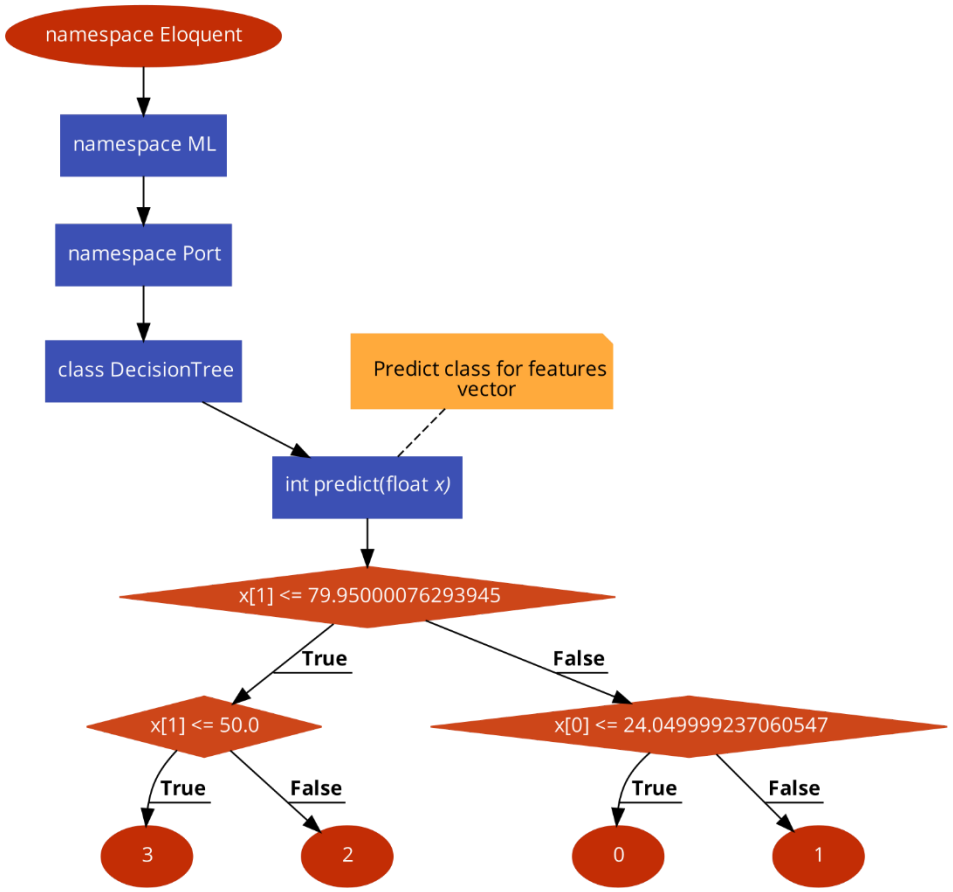
- **Components List:** ○ Esp32
  - ○ DHT 11 Sensor
  - ○ Bread Board
  - ○ Jumper wires
  - ○ USB Cable
- **Circuit Diagram:**



- **Hardware Setup:**



- **Flow-chart of Classifier Model:**



2.1.4 Task Steps

1. Dataset Acquisition:
   - Obtain a suitable weather dataset containing features such as temperature, humidity, heat index, etc., along with corresponding weather conditions.
2. Model Development:
   - Utilize TensorFlow to implement a Decision Tree Classifier for weather prediction.
   - Develop the model architecture.
   - Split the pre-processed dataset into training and testing sets.
3. Model Training:
   - Train the Decision Tree Classifier model using the training data.
   - Monitor the model's performance during training to ensure convergence and prevent overfitting.
4. Model Evaluation:
   - Evaluate the trained model's performance using metrics such as accuracy, precision, recall, and F1-score.
   - Analyse the model's predictions on the testing dataset to assess its effectiveness in predicting weather conditions.
5. Model Deployment:
   - Once satisfied with the model's performance, save the trained model for future use.
   - Provide instructions for deploying the model in real-world applications or systems where weather predictions are required.

2.1.5 Code:

```
#pragma once
#include <cstdarg>
namespace Eloquent {
    namespace ML {
        namespace Port {
            class DecisionTree {
                public:
                    /**
                    * Predict class for features vector
                    */
                    int predict(float *x) {
                        if (x[1] <= 79.95000076293945) {
                            if (x[1] <= 50.0) {
                                return 3;
                            }

                            else {
                                return 2;
                            }
                        }
```

```
            else {
              if (x[0] <= 24.049999237060547) {
                return 0;
              }

              else {
                return 1;
              }
            }
          }

        protected:
        };
      }
    }
  }
```

Now copy the above Decision Tree classifier code and save in any text file with name DecisionTree.h
Then copy the below arduino code and save it named as Weather_determining_model.ino

```
# include "DecisionTree.h"
Eloquent::ML::Port::DecisionTree weatherClassifier;
# include <math.h>
# include "DHT.h"

// Variable Declaration
float t;
float h;
float hic;
int prediction;

# define DHTPIN 4     // Digital pin connected to the DHT sensor
# define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);


void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.print("Welcome!");
  delay(10);

  dht.begin();
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(2000);
  // Read humidity
  float h = dht.readHumidity();
  // Read temperature as Celsius
  float t = dht.readTemperature();
```

```
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  // Compute heat index in Celsius (the default)
  float hic = calculate_heat_index(t,h);

  // Print values of temperature, humidity and heat index to serial monitor
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C  Heat index: "));
  Serial.print(hic);
  Serial.println(F("°C"));

  float input[3] = {t,h,hic};
  // Giving input values to predict Function
  int prediction = weatherClassifier.predict(input);

  // Checking Condition of weather
  Serial.print("Prediction: ");
  // Serial.print(prediction);
  if (prediction == 0){
    Serial.println("Foggy \U0001F32B");
  }
  else if(prediction == 1){
    Serial.println("Rainy \U0001F327");
  }
  else if(prediction == 2){
    Serial.println("Cloudy \u2601");
  }
  else if(prediction == 3){
    Serial.println("Sunny \u2600");
  }

  Serial.println();

  delay(1000);
}

double calculate_heat_index(double temperature, double humidity) {
    // Coefficients for the heat index formula
    double c1 = -42.379;
    double c2 = 2.04901523;
    double c3 = 10.14333127;
    double c4 = -0.22475541;
    double c5 = -6.83783e-3;
    double c6 = -5.481717e-2;
    double c7 = 1.22874e-3;
    double c8 = 8.5282e-4;
    double c9 = -1.99e-6;
```

```
  // Calculate the heat index
  double heat_index = (c1 + (c2 * temperature) + (c3 * humidity) + (c4 * temperature * humi
dity) +
                (c5 * pow(temperature, 2)) + (c6 * pow(humidity, 2)) +
                (c7 * pow(temperature, 2) * humidity) + (c8 * temperature * pow(humidity, 2))
 +
                (c9 * pow(temperature, 2) * pow(humidity, 2)));

  // Adjustments for specific conditions
  if (humidity < 13 && (80 <= temperature && temperature <= 112)) {
    double adjustment = ((13 - humidity) / 4.0) * sqrt((17 - fabs(temperature - 95.0)) / 17);
    heat_index -= adjustment;
  } else if (humidity > 85 && (80 <= temperature && temperature <= 87)) {
    double adjustment = ((humidity - 85) / 10.0) * ((87 - temperature) / 5.0);
    heat_index += adjustment;
  }

  return heat_index;
}
```

Now move these both files in same folder then open
the Weather_determining_model.ino file in your arduino IDE and upload the code to your Esp32 microcontroller.
Before uploading the code to the board, do verify that the DecisionTree.h file is also opened in the same arduino IDE beside the Weather_determining_model.ino file.
If any error occurs then open the DecisionTree.h file and comment the line containing #include <cstdarg>.
Now note the COM port of the esp32 and then close the Arduino IDE, so that serial connection with the Arduino IDE is disconnected.
Replace the COM10 in the below python code with the COM port noted earlier.

2.1.6 Jupyter Output:

```
Humidity: 95.00% Temperature: 27.70°C Heat index: 175.56
Prediction: Rainy 🌧️
Humidity: 95.00% Temperature: 27.70°C Heat index: 175.56
Prediction: Rainy 🌧️
Humidity: 95.00% Temperature: 27.70°C Heat index: 175.56
Prediction: Rainy 🌧️

Serial read stopped.
```

Task 2: Health prediction Model

2.2.1 Objective

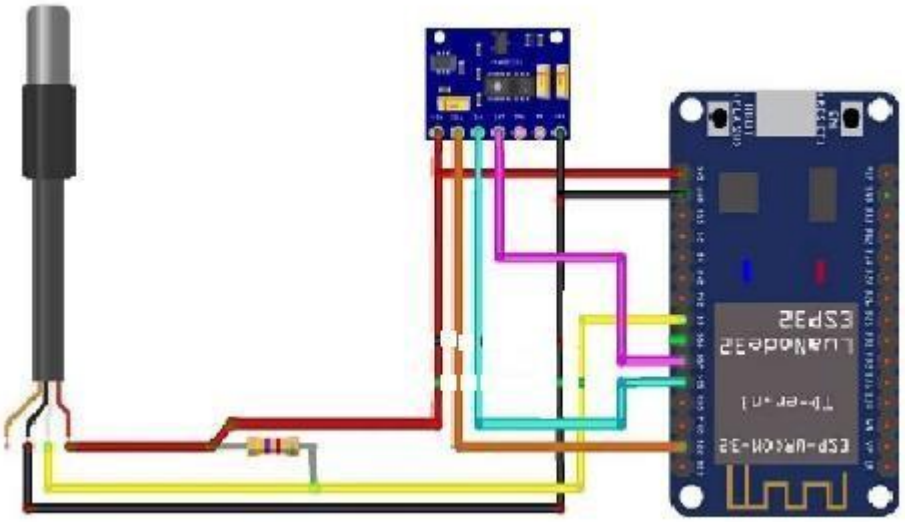
Develop instructional materials for students to learn machine learning using Jupyter Notebook and other ML tools. The aim is to accurately predict patient health conditions (normal, serious, dischargeable) based on temperature, pulse, and blood cell counts.

2.2.2 Tools and Technologies

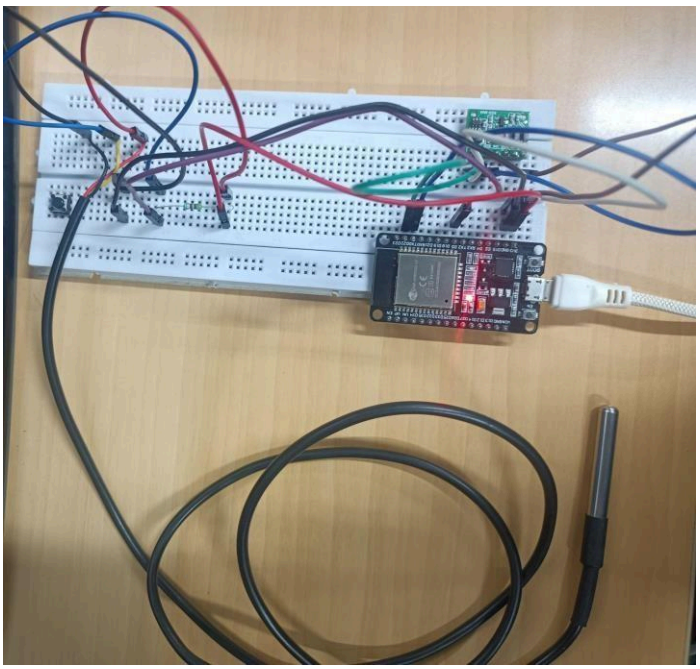
- Jupyter Notebook: For data preprocessing, model development, and training.
- Programming Language: Python.
- TensorFlow: For building and training the weather prediction model.
- NumPy: For numerical computations and data manipulation.
- Pandas: For data manipulation and analysis.

2.2.3 Hardware Setup

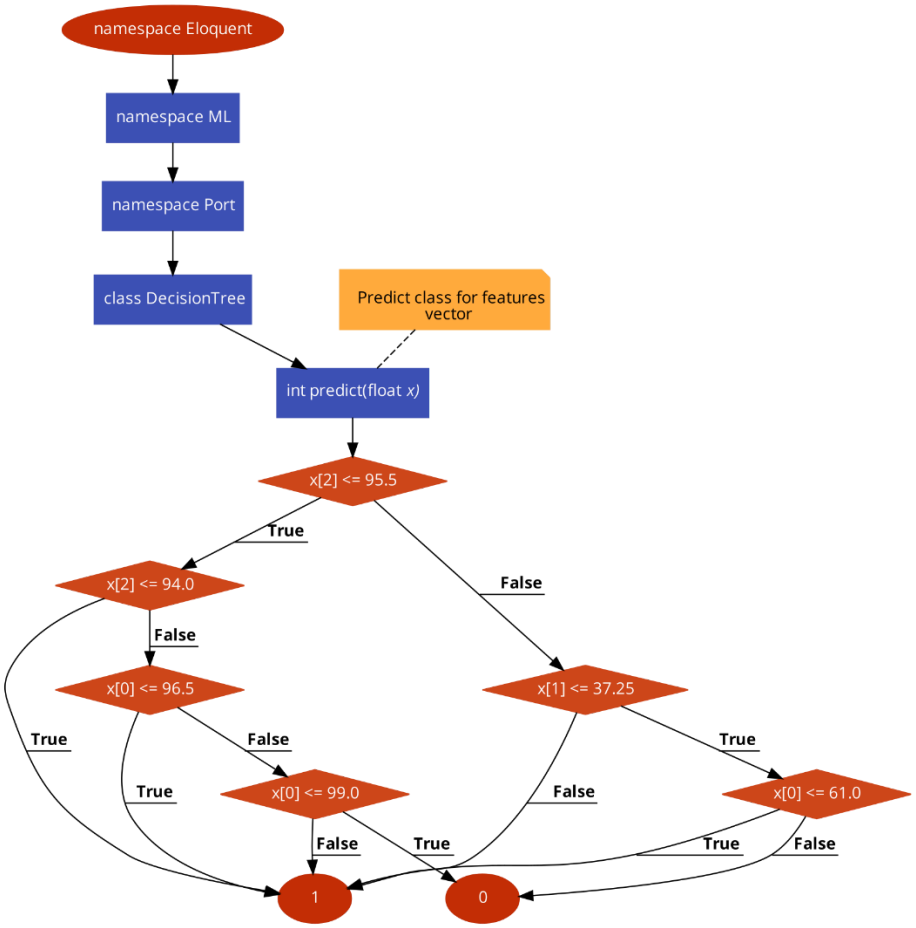
- **Components List:**
  - Esp32
  - Bread Board
  - Jumper wires
  - USB Cable
  - DS18B20 Sensor for temperature
  - Pulse Oximeter Sensor for BPM and SpO2
- **Circuit Diagram:**



- **Hardware Setup:**



- **Flow-chart:**



2.2.4 Task Steps

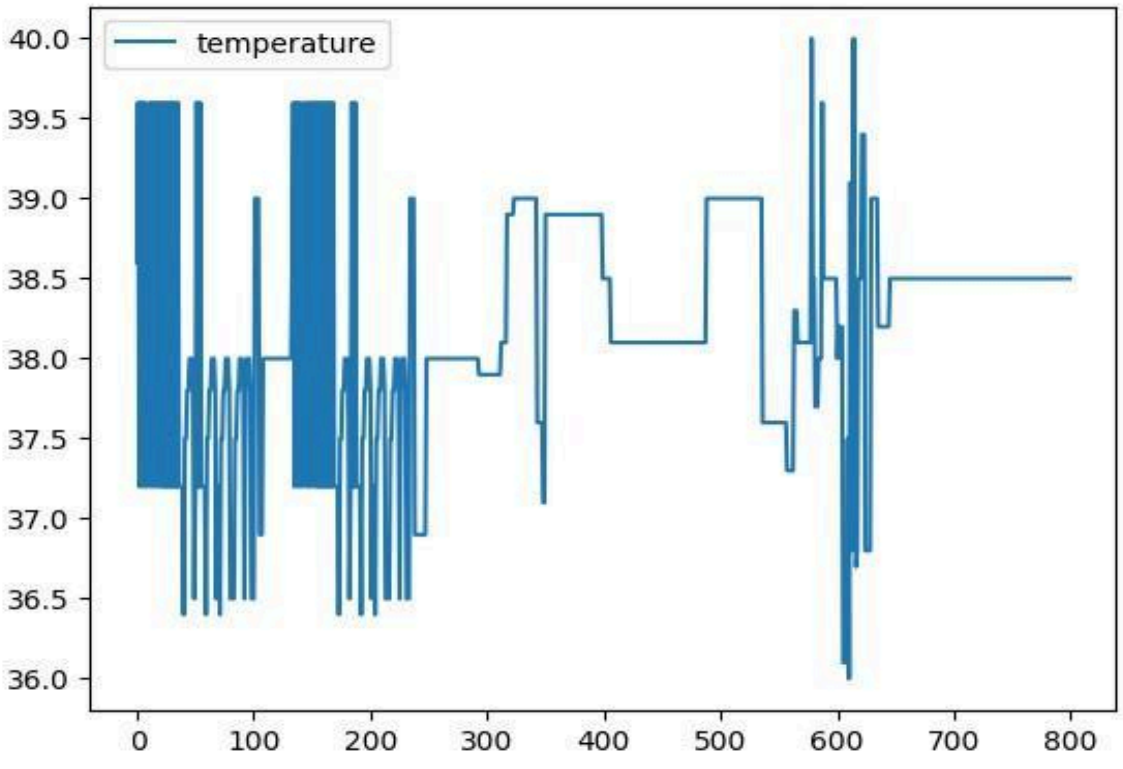
1. Import libraries:

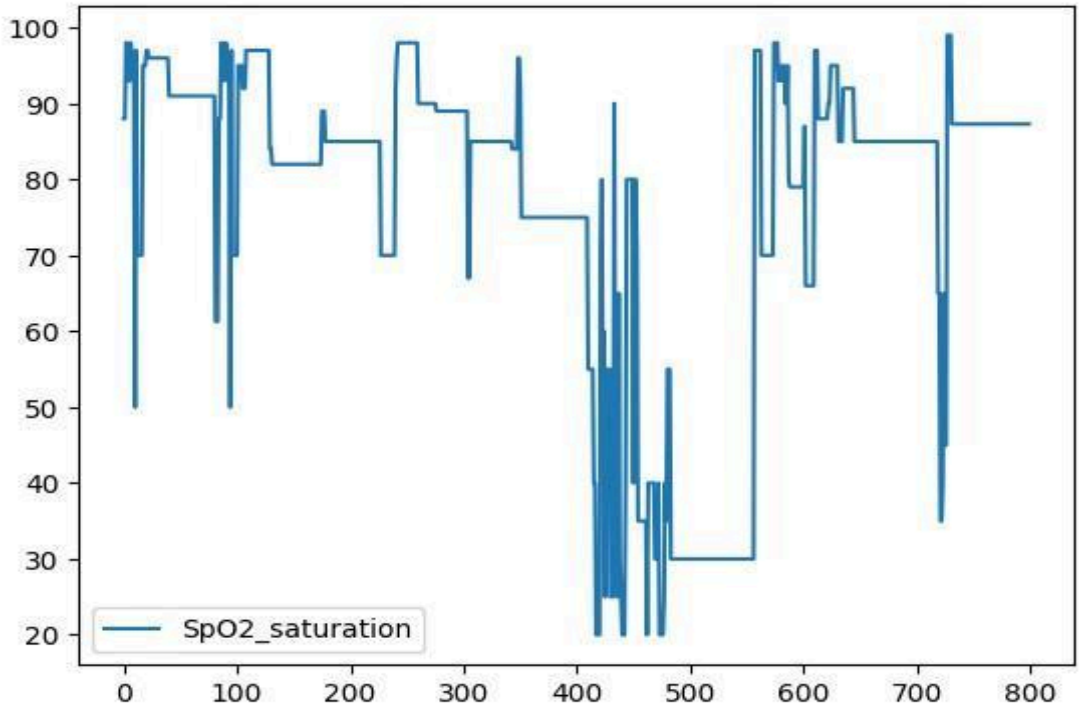
- TensorFlow
- micromlgen
- scikit-learn
- Seaborn

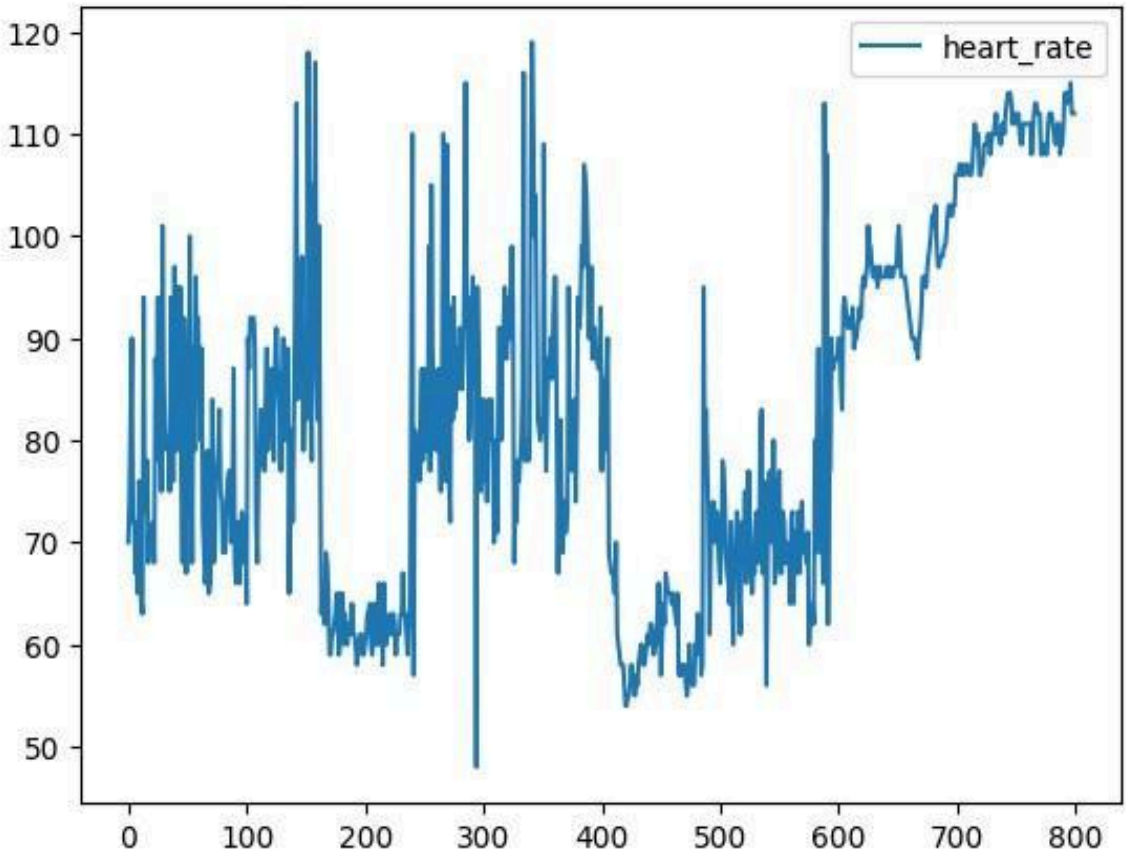
2. Dataset Acquisition
3. Determine Medical Condition:

```
def determine_medical_condition(row):
heart_rate = row['heart_rate']     temperature
= row['temperature']
   spo2 = row['SpO2_saturation']

   if 60 <= heart_rate <= 100 and 36.1 <= temperature <= 37.2 and 95 <= spo2 <=
100:
       return 0 #"Normal"
   elif heart_rate > 100 or heart_rate < 60 or temperature > 37.2 or temperature <
36.1 or spo2 < 95:
       return 1 #"Serious"
else:
       return 2 #"Dischargeable"
# refer data website
```

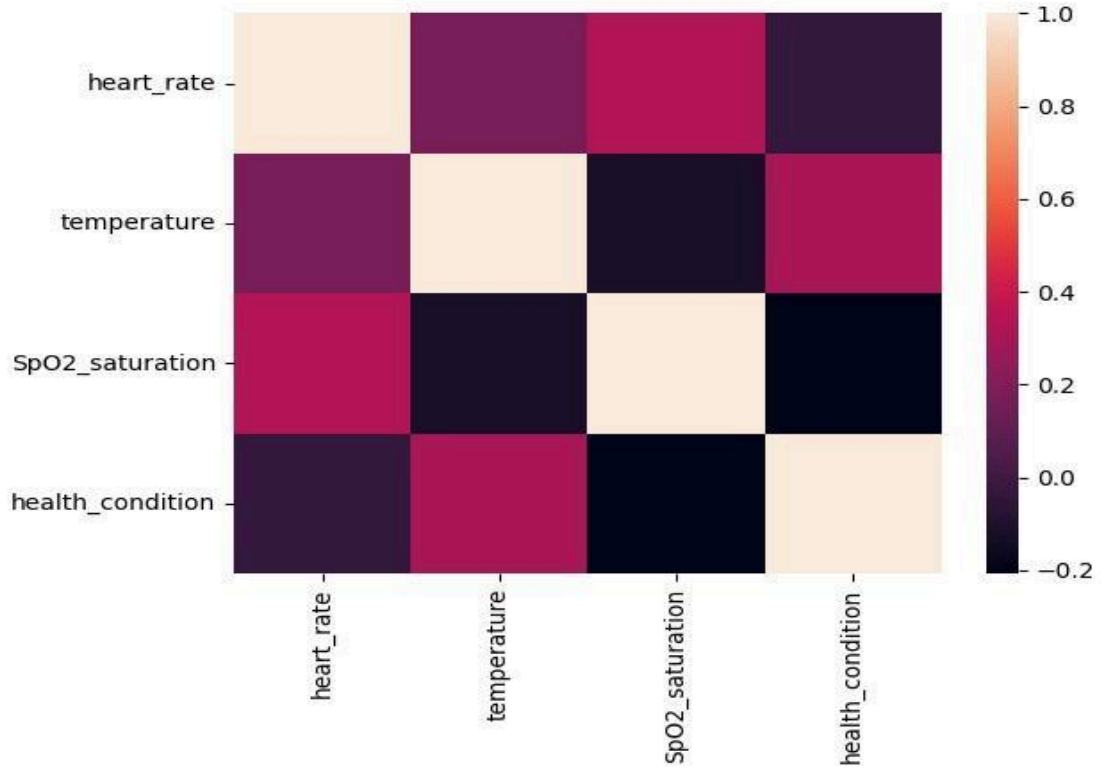
4. Data Preprocessing & Visualization




SpO2_saturation
100
90
80
70
60
50
40
30
20
0
100
200
300
400
500
600
700
800


heart_rate
120
110
100
90
80
70
60
50
0
100
200
300
400
500
600
700
800

A Seaborn heatmap is a data visualization tool that displays a matrix of values as a grid of colours, making it easy to identify patterns, correlations, and matrices, confusion matrices, and other types of matrix data. variations in the data. It is particularly useful for visualizing correlation.



5. Determine Medical Condition:

```
# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Build the Decision Tree Classifier model

model = tf.keras.Sequential([
tf.keras.layers.Input(shape=(X_train.shape[1],)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(len(label_encoder.classes_), activation='softmax')
])
```

6. Determine Medical Condition:

```
 health_classifier = DecisionTreeClassifier(random_state = 0)
health_classifier.fit(X_train, y_train)
```

7. Model Evaluation: # Measure Accuracy

```
    metrics.accuracy_score(y_test, y_pred)
```

Accuracy: 0.9810126582278481

8. Model Deployment: print(port(health_classifier))

this will print the decision tree classifier code, copy this and save it as DecisionTree.h
Now upload these files to the Esp32 board

1. DecisionTree.h
2. health_determining_model.ino

2.2.5 Code:

```
#pragma once
#include <cstdarg>
namespace Eloquent {
  namespace ML {
    namespace Port {
      class DecisionTree {
        public:
          /**
          * Predict class for features vector
          */
          int predict(float *x) {
            if (x[2] <= 95.5) {
              if (x[2] <= 94.0) {
                return 1;
              }

              else {
                if (x[0] <= 96.5) {
                  return 1;
                }

                else {
                  if (x[0] <= 99.0) {
                    return 0;
                  }

                  else {
                    return 1;
                  }
                }
              }
            }

            else {
              if (x[1] <= 37.25) {
                if (x[0] <= 61.0) {
                  return 1;
                }

                else {
                  return 0;
                }
              }
```

```
                else {
                  return 1;
                }
              }
            }

          protected:
          };
        }
      }
    }
```

Now copy the above Decision Tree classifier code and save in any text file with name DecisionTree.h

Then copy the below arduino code and save it named as health_determining_model.ino.

```
# include <Wire.h>
# include "MAX30100_PulseOximeter.h"
# include "DecisionTree.h"

Eloquent::ML::Port::DecisionTree health_Classifier;

# define TEMPERATURE_SENSOR_PIN 32  // Analog pin for the temperature sensor (replace with the actual pin number)
# define HEART_RATE_SENSOR_PIN 35   // Analog pin for the heart rate sensor (replace with the actual pin number)

// Initialize the PulseOximeter library
PulseOximeter pox;

// Variable Declaration
float t;
int h;
int s;
int prediction;

void setup() {
  Serial.begin(9600);
  Serial.println("Welcome!");

  pinMode(TEMPERATURE_SENSOR_PIN, INPUT);
  pinMode(HEART_RATE_SENSOR_PIN, INPUT);

  // Initialize the Pulse Oximeter
  if (!pox.begin()) {
    Serial.println("Failed to initialize pulse oximeter!");
    while (1);
  } else {
    Serial.println("Pulse oximeter initialized successfully!");
  }
}
```

```
void loop() {
  delay(2000);

  // Sensor data Collection part

  // Read temperature value
  int temperatureValue = analogRead(TEMPERATURE_SENSOR_PIN);
  t = convertToCelsius(temperatureValue);
  Serial.print("Temperature (°C): ");
  Serial.println(t);

  // Read heart rate value
  h = analogRead(HEART_RATE_SENSOR_PIN);
  Serial.print("Heart Rate (bpm): ");
  Serial.println(h);

  // Read SpO2 value
  int s = pox.getSpO2();
  Serial.print("SpO2 (%): ");
  Serial.println(s);

  // Print values of heart_rate, temperature, and SpO2_saturation to serial monitor
  Serial.print(F("Heart_rate: "));
  Serial.print(h);
  Serial.print(F("  Temperature: "));
  Serial.print(t);
  Serial.print(F("  SpO2: "));
  Serial.println(s);

  float input[3] = {h, t, s};
  // Giving input values to predict Function
  int prediction = health_Classifier.predict(input);

  // Checking Condition
  Serial.print("Prediction: ");
  switch (prediction) {
    case 0:
      Serial.println("Normal");
      break;
    case 1:
      Serial.println("Serious");
      break;
    case 2:
      Serial.println("Dischargeable");
      break;
    default:
      Serial.println("Unknown");
      break;
  }
  Serial.println();
  delay(1000);
}

float convertToCelsius(int adcValue) {
```

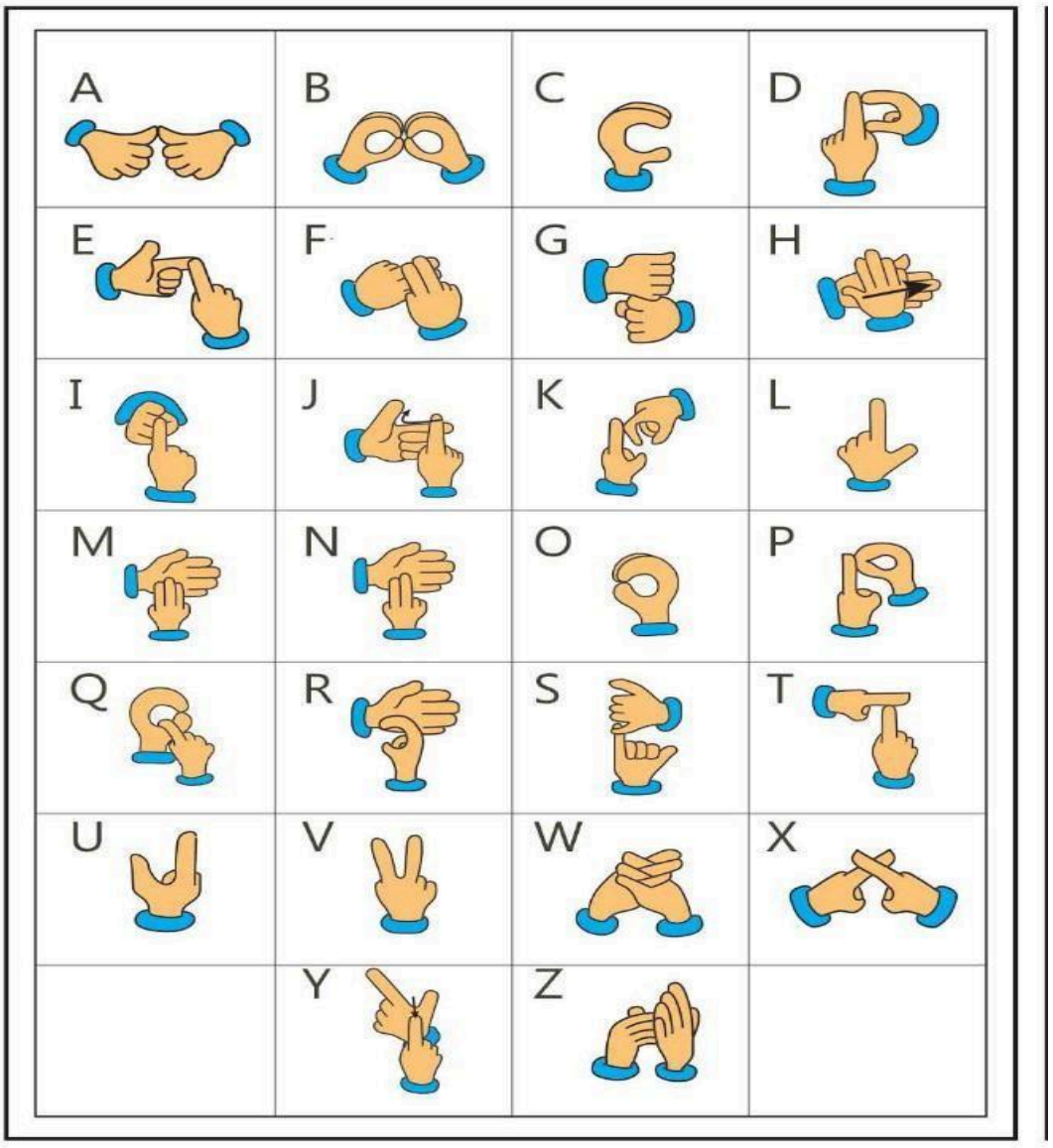
```
  float voltage = adcValue * 3.3 / 4095.0;  // Convert ADC value to voltage for ESP32
  float temperatureCelsius = (voltage - 0.5) * 100;  // Convert voltage to temperature in Celsi
us
  return temperatureCelsius;
}
```

Now move these both files in the same folder then open the "health_determining_model.ino" file in your arduino IDE and upload the code to your Esp32 microcontroller.
Before uploading the code to the board, do verify that the "DecisionTree.h" file is also opened in the same arduino IDE beside the .ino file.
If any error occurs then open the "DecisionTree.h" file and comment the line containing this code

2.2.6 Jupyter Output:

```
Temperature (°C): 37.2      Heart Rate (bpm): 82        SpO2 (%): 92
Prediction: Normal
Temperature (°C): 39.6      Heart Rate (bpm): 70        SpO2 (%): 72
Prediction: Serious
Temperature (°C): 39.6      Heart Rate (bpm): 86        SpO2 (%): 96
Prediction: Normal
Serial read stopped.
```

Task 3: Indian Sign Language Recognition (gesture recognition) and Conversion to Speech



2.3.1 Objective

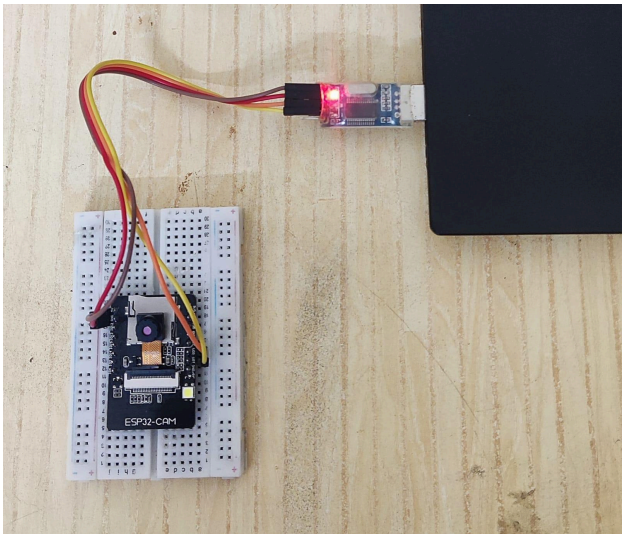
Develop instructional materials for students to learn machine learning using Jupyter Notebook and other ML tools. The aim is to accurately recognize Indian Sign Language (ISL) gestures, convert them into corresponding text, and then convert the text into speech.

2.3.2 Tools and Technologies

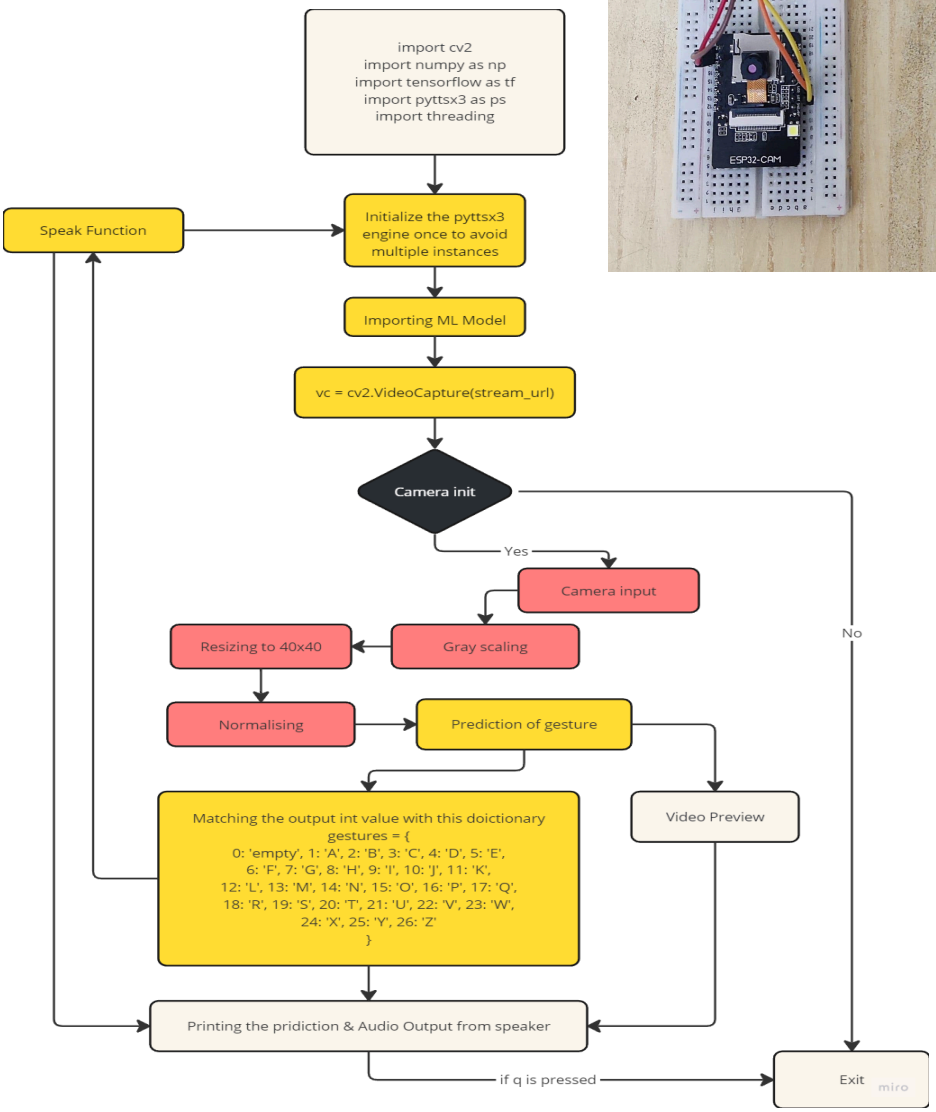
- Jupyter Notebook: For data preprocessing, model development, and training.
- Programming Language: Python.
- TensorFlow: For building and training the weather prediction model.
- NumPy: For numerical computations and data manipulation.
- Pandas: For data manipulation and analysis.

2.3.3 Hardware Setup

- **Components List:**
  - Esp32 Cam
  - TTL connecter
  - Bread Board
  - Jumper wires



- **Flowchart:**



2.3.4 Task Steps

So let us start this exciting journey with the following steps:

1. Making our system ready for this project, for that you need to download some libraries of python using pip command.
    - OpenCV ----> pip install opencv-python ----> open-source library for computer vision, machine learning, and image processing.
    - PIL ----> pip install pillow ----> Pillow provides the Image Draw module that provides simple 2D graphics for Image objects.
    - tensorflow ----> pip install tensorflow ----> used to build and train deep learning models as it facilitates the creation of computational graphs and efficient execution on various hardware platforms
    - sklearn ----> pip install scikit-learn used here for splitting data into test and train.
    - pyttsx3 ----> pip install pyttsx3 used to convert text to speech.

Do not worry, you are not supposed to copy these comments and install them one by one in your terminal because you will see these commands installed at the time of their use in this notebook file only, so chill....

2. Making Hardware ready for this project, for that we will be following some sub steps which are as followed:.
    - Getting the Arduino code
    - Interfacing the Hardware
    - Installing the code to the Hardware
    - Getting the esp32_url for this project
3. Now after successfully installing these libraries of python, you need to import these libraries for this project.
4. Data collection and processing, these very steps have some sub steps which is needed to be followed:
    - Data Collection: Capture images from esp32-cam and save them with their class name (in our project it is A to Z gestures).
    - Data storing: Now after creating your data with their class name you need to store these data into csv file, in our case we are taking the image data of any pixel, resizing it to 40x40, splitting the data into test and train and after this we will be storing these data in to csv files named as train.csv and test.csv.
    - Data Loading: We will be loading these csv files in our code.
    - Data Processing: Now it's time to process the data for this project which will be helpful for us to develop the model and test it for accuracy.
    - Debugging: Debugging the data is most essential before model development as it debugs our process and alert's us when there is any error.

- Previewing Data: After some more data processing now, we will be previewing the data in grayscale as this gives us the correct image data for developing our model.

5. Developing our ML model.
6. Deploying this ML model using esp32-cam, for this we will be following some sub-steps which are as follows:
    - some necessary system commands
    - Import some libraries
    - Load ML model
    - Connect to esp32-cam
    - Prediction
    - Converting the predicted value to text ● Converting the text to speech

and yes, if you follow these steps, you will be getting your project ready.

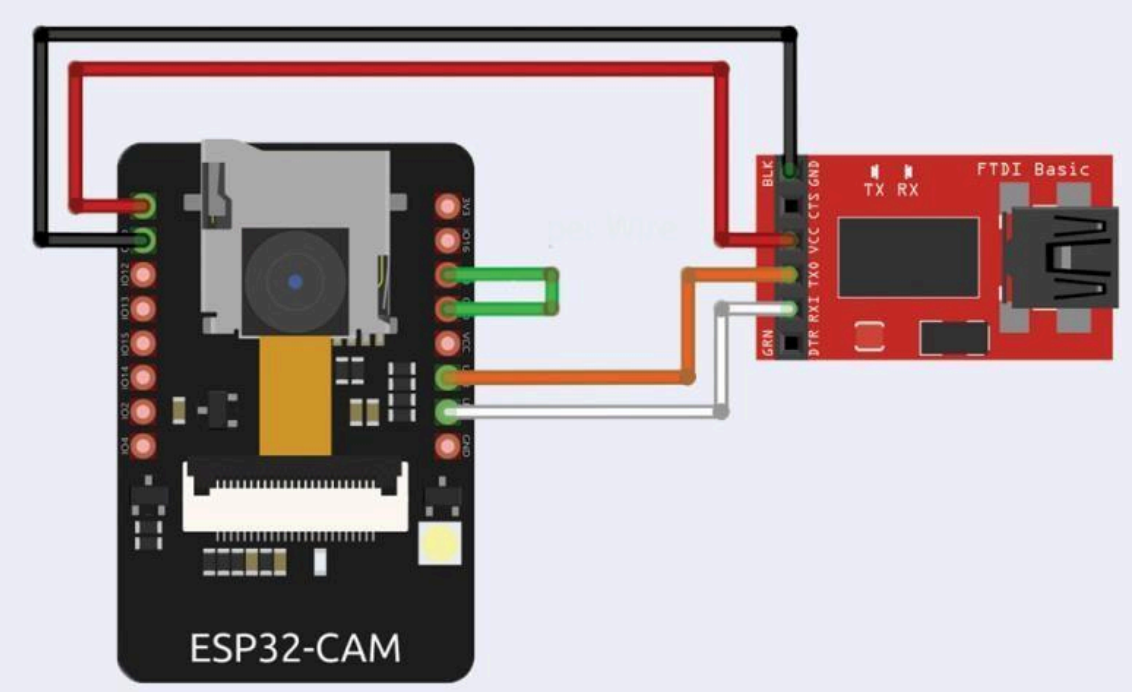
7. Interfacing the Hardware

See the below diagram and interface the esp32-cam as to is (Do not forget to connect GPIO0 pin to GND).

Now connect the FTDI to the computer through the usb cable

- 5V ---> VCC
- GND ---> GND
- U0T ---> Rx
- U0R ---> Tx
- IO0 ---> GND

pin diagram for your ease.



8. Installing code to Hardware:

Follow these steps to install code to your Hardware:

Step 1: Open the example code and do some necessary changes so that it can be used in our project, which are as follows:

- Comment the line no 17 which is #define CAMERA_MODEL_ESP_EYE // Has PSRAM
- Uncomment the line no 25 which is #define CAMERA_MODEL_AI_THINKER // Has PSRAM
- change the ssid and password const char *ssid = "**********";
- const char *password = "**********";

Step 2: Now Install the code on esp32cam by selecting the correct board and port to which the esp32-cam is connected. To do so

- Go to Tools---> Board---> esp32---> AI Thinker ESP32-CAM
- Go to Tools---> Port---> your port and hit the upload button (it will take some time to verify and upload the code to the esp32-cam).
- if it shows any error for debugging follow these steps:
- Press the reset button present on the esp32-cam module.
- Check the connection for any loose fitting.

Step 3: Your Hardware is ready with the code now, only you need to get your esp32_url to get the camera access for this project. To do so you need to follow some steps which are as follows:

- Open the serial monitor
- Set the baud rate to 115200 baud
- Remove the jumper wire which is connected to IO0 to GND of esp32-cam module
- Press the reset button present on the esp32-cam module, you will notice some URL is printed on the serial monitor.
- Copy this URL and paste on the browser of your PC (ensure that your pc is also connected to the same hotspot to which the esp32-cam module is connected).
- Now search for the start stream button red in colour.
- Click that button for starting the video stream, now video will start streaming on the web page
- Now right click on the video displayed and go to open in new tab and close the previous tab
- Copy the URL of the current page and that's it you got your esp32 URL.

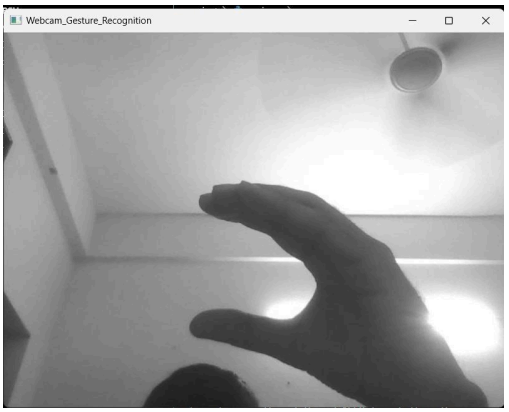
For any other details please refer to the GitHub repo link given in this file.

2.3.5 Code:

```
import cv2
import numpy as np
import tensorflow as tf
import pyttsx3 as ps
import threading

# Initialize the pyttsx3 engine once to avoid multiple instances
engine = ps.init()

def speak(text):
    def speak_thread(text):
        # Ensure the global engine is used
        engine.say(text)
        engine.runAndWait()
    thread = threading.Thread(target=speak_thread, args=(text,))
    thread.start()

# Load the TensorFlow model
model = tf.keras.models.load_model("sign_language_model.h5") # you should change these
 file path according to your file path

# Replace with your ESP32-CAM stream URL
stream_url = 'http://192.168.92.123:81/stream'

# Open a connection to the IP camera using OpenCV VideoCapture
vc = cv2.VideoCapture(stream_url)

# Check if the webcam opened successfully
if not vc.isOpened():
    print("Error: Could not open webcam.")
    exit()

# Map the predicted class to the corresponding gesture
gestures = {
        0: 'empty', 1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E',
        6: 'F', 7: 'G', 8: 'H', 9: 'I', 10: 'J', 11: 'K',
        12: 'L', 13: 'M', 14: 'N', 15: 'O', 16: 'P', 17: 'Q',
        18: 'R', 19: 'S', 20: 'T', 21: 'U', 22: 'V', 23: 'W',
        24: 'X', 25: 'Y', 26: 'Z'
}

# Loop until the user presses the 'q' key
while True:
    # Capture a frame from the webcam
    ret, frame = vc.read()
    if not ret:
        print("Failed to capture image")
        break
```

```
    # Convert the frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Resize the frame to 40x40
    resized_frame = cv2.resize(gray_frame, (40, 40))

    # Normalize the pixel values
    normalized_frame = resized_frame / 127.5 - 1.0

    # Add a batch dimension and ensure it has the right shape
    input_data = np.expand_dims(normalized_frame, axis=(0, -1)).astype(np.float32)

    # Perform inference using the model
    output_data = model.predict(input_data)

    # Get the predicted class
    predicted_class = np.argmax(output_data)

    predicted_gesture = gestures.get(predicted_class, 'unknown')

    # Print the predicted class
    print("Predicted gesture:", predicted_gesture)
    if predicted_gesture != 'empty':
        speak(predicted_gesture)

    # Display the frame
    cv2.imshow('Webcam_Gesture_Recognition', gray_frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object
vc.release()

# Close all windows
cv2.destroyAllWindows()
```

2.3.6 Jupyter Output:




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Predicted gesture: A
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Predicted gesture: A
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Predicted gesture: empty
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Predicted gesture: empty
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Predicted gesture: empty
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
Predicted gesture: empty
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Predicted gesture: A
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
Predicted gesture: A
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Predicted gesture: C
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Predicted gesture: C
```

Conclusion

This internship at FOSSEE, IIT Bombay, provided me with an invaluable opportunity to apply my knowledge of machine learning and Python programming to real-world problems. By developing models for weather forecasting, patient health prediction, and ISL gesture recognition, I gained practical experience and enhanced my skills in data preprocessing, model development, training, and evaluation. This experience has significantly contributed to my understanding of practical machine-learning applications.

Reference

- TensorFlow: TensorFlow Lite
- Dataset Task 1: Mumbai Weather

Work is Available at

- Task 1
- Task 2
- Task 3