



Menoufia University  
Faculty of Electronic Engineering



Department Of Computer Science & Engineering

Submitted by:

#	Name	Academic ID
1	Yousef Mohamed Mohamed	1900507
2	Walaa Adel Abdelaleem	1900481
3	Toka Shawky Elmaadawy	1900135
4	Ayman Ragab Abd Elwahed	1800071
5	Mostafa Ashraf Mohamed	1900425
6	Mohamed Ameen Ameen	1900319

**Graduation Project**

**Smart Agriculture System Using AI & IOT**

**Supervised By**

**Dr./ Elhossiny Ibrahim**



**DEPARTMENT OF COMPUTER SCIENCE &  
ENGINEERING**

**Graduation Project**

# **Smart Agriculture System Using AI & IOT**

**We try to help the field of agriculture using 'Smart  
Agriculture Application'**

**By**

***Yousef Mohamed Mohamed***

***Walaa Adel Abdelaleem***

***Toka Shawky Elmaadawy***

***Ayman Ragab Abd Elwahed***

***Mostafa Ashraf Mohamed***

***Mohamed Ameen Ameen***

# Table of contents

## **Chapter 1: Introduction**

1.1: Overview of IoT in Agriculture.....	6
1.2: Importance of Crop Recommendation.....	8
1.3: Objectives of the Project.....	10

## **Chapter 2: Internet of Things**

2.1: Theoretical Review.....	13
2.1.1: Existing IoT Applications in Agriculture.....	13
2.1.2: Components of Crop Recommendation.....	15
2.2: System Architecture.....	16
2.3: Hardware Components.....	23
2.3.1: Sensors.....	23
2.3.2: Controller: Arduino UNO.....	28
2.3.3: WIFI Module NRF24L01.....	29
2.4: Block Diagram.....	31
2.5: Circuit Diagram.....	32

## **Chapter 3: Machine Learning**

3.1: Introduction to Machine Learning.....	33
3.2: Types of machine learning.....	34
3.3: Machine Learning Workflow.....	35
3.4: Algorithms of Machine learning.....	36
3.5: Machine learning in our project.....	38
3.5.1: K-Nearest Neighbors (KNN) Algorithm .....	38
3.5.2: The Data.....	39
3.5.3: Model step.....	40

## **Chapter 4: Deep Learning**

4.1: Introduction.....	45
4.2: Deep Learning and Its Algorithms.....	45
4.2.1: Popular Deep Learning Algorithms.....	46
4.2.2: Convolutional Neural Networks (CNNs).....	47
4.2.2.1: Introduction to CNNs.....	47
4.2.2.2: How CNNs Work.....	47
4.2.2.3: Components of CNNs.....	48
4.3: Main Parts of CNN Model.....	50

## **Chapter 5: Backend Implementation**

5.1: Introduction.....	57
5.2: Key Components of Backend Development.....	57
5.3: Common Backend Languages.....	57
5.3.1: Backend with Python.....	58
5.3.1.1: Flask.....	58
5.4: APIs.....	60
5.5: Main Parts of Backend Code.....	61
5.6: Testing API Tools.....	64
6.7: Hosting.....	66
5.8: Connection with Mobile.....	69

## **Chapter 6: Implementation (Mobile app development)**

6.1 Introduction.....	70
6.2 What is Flutter? .....	70
6.3 Object-Oriented Programming (OOP) in Flutter.....	71
6.4 Application Structure.....	71

6.5 Image Selection.....	73
6.6 Image Upload Code.....	74
6.7 Diagnosis Response.....	75
6.8 Steps to Use the Application.....	76

<b>References.....</b>	<b>7</b>
------------------------	----------

# Chapter 1

## Introduction

### 1.1: Overview of IoT in Agriculture

- **Sensors:** Collect data on various environmental parameters.
  - **Soil Sensors:** Measure moisture, temperature, pH, and nutrient levels.
  - **Weather Stations:** Track local climate conditions like temperature, humidity, rainfall, and wind speed.
  - **Livestock Monitoring:** Sensors to track health, location, and activity of animals.
- **Connectivity:** Devices are connected via the internet, enabling data transmission.
  - **Wi-Fi, LoRa, and Cellular Networks:** Common methods for connecting IoT devices in agriculture.
- **Data Management:** Data collected from sensors is stored and managed.
  - **Cloud Platforms:** Services like Firebase, AWS, and Azure for data storage and processing.
  - **Edge Computing:** Processing data closer to where it is generated to reduce latency.
- **Analytics and Machine Learning:** Analyzing data to derive actionable insights.
  - **Predictive Analytics:** Forecasting crop yields, pest outbreaks, and optimal planting times.
  - **Prescriptive Analytics:** Providing recommendations for fertilizer application, irrigation schedules, and crop selection.
- **Automation:** Implementing automated systems based on data insights.

- **Smart Irrigation Systems:** Automatically adjust water levels based on soil moisture data.
- **Automated Machinery:** Drones and robots for planting, harvesting, and monitoring crops.

## **Benefits**

- **Increased Efficiency:** Optimizes resource usage (water, fertilizers, pesticides) leading to cost savings.
- **Higher Yields:** Data-driven decisions improve crop productivity.
- **Sustainability:** Reduces environmental impact by minimizing waste and managing resources effectively.
- **Real-Time Monitoring:** Continuous monitoring allows for timely interventions to prevent crop loss.
- **Improved Livestock Management:** Enhances animal health and productivity through constant monitoring.

## **Applications**

- **Precision Farming:** Customizing farming practices for different areas of a field based on data insights.
- **Greenhouse Automation:** Controlling climate conditions within greenhouses for optimal plant growth.
- **Livestock Tracking:** Monitoring the health and location of livestock to improve management.
- **Crop Monitoring:** Using drones and sensors to monitor crop health and detect issues early.
- **Supply Chain Management:** Tracking produce from farm to market to ensure quality and reduce waste.
- **Crop Recommendation:** Data From soil sensors used as an input to machine learning model to recommend optimal crop to plant

## **Challenges**

- **High Initial Costs:** Investing in IoT technology can be expensive.
- **Data Security:** Protecting data from breaches and ensuring privacy.
- **Connectivity Issues:** Reliable internet access is crucial, which can be challenging in remote areas.
- **Technical Expertise:** Farmers need training and support to effectively use IoT systems.

## **1.2: Importance of Crop Recommendation**

Agriculture has always been a vital sector of the economy, providing food, raw materials, and employment opportunities. However, traditional farming practices often rely heavily on the experience and intuition of farmers, which can lead to suboptimal crop choices and inefficient resource utilization. The integration of technology, particularly IoT (Internet of Things) and machine learning, into agriculture has opened new avenues for improving crop management and productivity.

**One such application is crop recommendation, which offers several key benefits:**

### **1. Optimized Resource Utilization**

Crop recommendation systems analyze various environmental and soil parameters, such as temperature, humidity, soil moisture, pH levels, and nutrient content. By recommending crops that are best suited to the current conditions, these systems help farmers make informed decisions, ensuring optimal use of water, fertilizers, and other resources. This leads to more sustainable farming practices and reduces wastage.

### **2. Enhanced Crop Yield**

By selecting crops that are well-suited to the specific conditions of their land, farmers can achieve higher yields. Crop recommendation systems use historical data and real-time inputs to predict the most suitable crops, reducing the risk of crop failure and



maximizing productivity. This is particularly important in regions where agricultural output is critical for food security.

### **3. Increased Profitability**

Efficient resource utilization and higher crop yields directly contribute to increased profitability for farmers. By growing the right crops at the right time, farmers can reduce input costs and increase their overall income. Additionally, crop recommendation systems can help farmers identify high-value crops that are in demand, further boosting their earnings.

### **4. Adaptation to Climate Change**

Climate change poses significant challenges to agriculture, with unpredictable weather patterns and extreme conditions affecting crop growth. Crop recommendation systems can help farmers adapt to these changes by providing recommendations based on current and forecasted climatic conditions. This adaptability ensures that farmers can continue to produce food even in the face of changing environmental conditions.

### **5. Data-Driven Decision Making**

Traditional farming often relies on anecdotal knowledge and personal experience. Crop recommendation systems bring a scientific approach to farming by using data analytics and machine learning algorithms to provide evidence-based recommendations. This shift towards data-driven decision-making helps in improving the accuracy and reliability of crop choices.

### **6. Sustainability and Environmental Conservation**

By promoting the cultivation of crops that are well-suited to the local environment, crop recommendation systems contribute to sustainable farming practices. They help in maintaining soil health, conserving water resources, and reducing the use of harmful chemicals. This not only benefits the environment but also ensures the long-term viability of agricultural lands.

## **7. Empowerment of Farmers**

Access to advanced technologies such as IoT and machine learning empowers farmers by providing them with tools and knowledge that were previously inaccessible. This democratization of technology helps smallholder and marginal farmers improve their livelihoods and compete in the market. It also encourages the adoption of modern agricultural practices, leading to overall sectoral growth.

### **1.3: Objectives of the Project:**

Develop an IoT-based system that utilizes machine learning to recommend the most suitable crops for cultivation based on real-time data collected from various sensors. This project aims to improve agricultural productivity, resource efficiency.

- **To Design and Implement an IoT System for Data Collection**
  - Develop a network of sensors (including rain, DHT11, pH, and soil NPK sensors) to gather real-time environmental and soil data.
  - Integrate these sensors with an Arduino controller for data acquisition and processing.
  - Ensure reliable transmission of collected data to a cloud-based platform (Firebase) using a Wi-Fi module.
- **To Establish a Cloud-Based Infrastructure for Data Management**
  - Set up a Firebase Cloud database to store and manage the sensor data.
  - Implement data synchronization and real-time updates to ensure the availability of the latest information.
  - Develop mechanisms for data security and integrity within the cloud infrastructure.

- **To Develop and Train a Machine Learning Model for Crop Recommendation**

- Collect and preprocess historical agricultural data and real-time sensor data for model training.
- Explore various machine learning algorithms and select the most appropriate one for crop recommendation.
- Train the machine learning model using the prepared dataset and evaluate its performance in terms of accuracy and reliability.

- **To Integrate the Machine Learning Model with the IoT System**

- Develop an interface to connect the trained machine learning model with the Firebase database.
- Implement functionality for the model to analyse real-time data and provide crop recommendations.
- Ensure seamless interaction between the IoT components, the cloud platform, and the machine learning model.

- **To Develop a User-Friendly Interface for Farmers**

- Design a frontend application (web or mobile) that allows farmers to access crop recommendations easily.
- Implement data visualization techniques to present sensor data and crop recommendations in an intuitive manner.
- Provide additional features such as alerts, notifications, and historical data analysis to enhance user experience.

- **To Conduct System Testing and Validation**

- Perform comprehensive testing of the hardware components to ensure accurate data collection.

- Validate the accuracy and reliability of the machine learning model's crop recommendations.
- Conduct field trials to evaluate the practical effectiveness of the system in real-world agricultural settings.
- **To Assess the Impact of the System on Agricultural Practices**
  - Analyse the potential benefits of the crop recommendation system in terms of yield improvement, resource efficiency, and economic gains for farmers.
  - Gather feedback from farmers and stakeholders to identify areas for improvement and further development.
  - Document case studies and success stories to highlight the system's impact on agricultural practices.
- **To Promote Sustainability and Scalability**
  - Ensure that the system promotes sustainable farming practices by optimizing resource use and minimizing environmental impact.
  - Explore possibilities for scaling the system to cover larger agricultural areas and a wider range of crops.
  - Investigate potential integrations with other agricultural technologies and systems to enhance overall effectiveness.

**By** achieving these objectives, the project aims to provide a comprehensive solution that leverages IoT and machine learning to support farmers in making informed crop choices, ultimately leading to improved agricultural outcomes and sustainability.

## **Chapter 2**

### **Internet of Things**

#### **2.1: Theoretical Review**

##### **2.1.1: Existing IoT Applications in Agriculture :**

The integration of IoT (Internet of Things) technology in agriculture has revolutionized the way farming is conducted, leading to increased efficiency, productivity, and sustainability. Below are some of the key existing IoT applications in agriculture:

#### **1. Precision Farming**

- **Soil Monitoring:** IoT sensors are used to monitor soil conditions, including moisture levels, temperature, pH, and nutrient content. This information helps farmers make data-driven decisions about irrigation, fertilization, and crop selection.
- **Variable Rate Technology (VRT):** VRT systems use IoT sensors to apply varying amounts of inputs (seeds, fertilizers, pesticides) based on the specific needs of different areas within a field, optimizing resource use and improving yield.

#### **2. Smart Irrigation Systems**

- **Automated Irrigation:** IoT-based smart irrigation systems use soil moisture sensors and weather data to automate the watering process. This ensures crops receive the right amount of water at the right time, conserving water and improving crop health.
- **Remote Monitoring and Control:** Farmers can monitor and control irrigation systems remotely using smartphones or computers, allowing for real-time adjustments based on weather conditions and soil moisture levels.

### **3. Crop Monitoring**

- **Growth Monitoring:** IoT devices, such as cameras and drones equipped with sensors, capture images and data on crop growth and health. This information helps farmers identify issues like pest infestations, diseases, and nutrient deficiencies early.
- **Environmental Monitoring:** IoT sensors monitor environmental factors such as temperature, humidity, light intensity, and CO2 levels, providing insights into optimal growing conditions and potential stressors.

### **4. Livestock Management**

- **Health Monitoring:** Wearable IoT devices for livestock track vital signs, activity levels, and behavior. This data helps farmers monitor animal health, detect illnesses early, and manage breeding cycles more effectively.
- **Location Tracking:** GPS-enabled IoT devices help farmers track the location and movement of livestock, preventing loss and theft and ensuring efficient grazing management.

### **5. Supply Chain Management**

- **Cold Chain Monitoring:** IoT sensors monitor the temperature and humidity of perishable agricultural products during transportation and storage, ensuring they remain within safe limits to maintain quality and reduce spoilage.
- **Inventory Management:** IoT systems track inventory levels of agricultural inputs (seeds, fertilizers, pesticides) and outputs (harvested crops), helping farmers manage stock and optimize supply chain operations.

## **6. Greenhouse Automation**

- **Climate Control:** IoT sensors in greenhouses monitor and regulate temperature, humidity, light, and CO2 levels automatically, creating optimal growing conditions for crops.
- **Hydroponics and Aquaponics:** IoT systems manage nutrient delivery and water quality in hydroponic and aquaponic setups, ensuring precise control over the growing environment and improving crop yield and quality.

### **Crop Recommendation systems:**

Crop recommendation systems are an integral part of modern agriculture, leveraging advanced technologies such as IoT (Internet of Things) and machine learning to provide farmers with data-driven guidance on the most suitable crops to cultivate. These systems analyze various environmental, soil, and historical data to recommend crops that will yield the best results under specific conditions. Below are the key components, methodologies, and benefits of crop recommendation systems:

#### **2.1.2: Components of Crop Recommendation Systems**

##### **1. Data Collection:**

- **Environmental Data:** Sensors collect real-time data on weather conditions, including temperature, humidity, rainfall, and sunlight.
- **Soil Data:** Soil sensors measure moisture levels, pH, and nutrient content (NPK levels) to assess soil health and suitability for different crops.
- **Historical Data:** Historical crop yield data, past weather patterns, and agricultural practices are used to identify trends and patterns.

##### **2. Data Processing and Storage:**

- **IoT Devices:** Sensors connected to microcontrollers (e.g., Arduino) gather and transmit data to cloud platforms for processing.

- **Cloud Platforms:** Data is stored and managed in cloud databases (e.g., Firebase) for easy access and scalability.

### 3. Machine Learning Models:

- **Data Preprocessing:** Collected data is cleaned and preprocessed to remove noise and inconsistencies.
- **Feature Engineering:** Relevant features (e.g., soil moisture, temperature, and historical yield) are extracted to train the machine learning model.
- **Model Training:** Machine learning algorithms (e.g., decision trees, random forests, neural networks) are trained on the preprocessed data to learn patterns and make predictions.
- **Model Evaluation:** The model's performance is evaluated using metrics like accuracy, precision, recall, and F1-score to ensure reliability.

### 4. User Interface:

- **Web/Mobile Application:** A user-friendly interface allows farmers to input their location and view crop recommendations based on real-time data.
- **Data Visualization:** Interactive charts and graphs display sensor data and crop recommendations in an easily understandable format.

## **2.2: System Architecture**

### **2.2.1 High level system design:**

The high-level system design of an IoT-based crop recommendation system integrates various components to collect, process, analyze, and present data for making informed crop choices. Below is an outline of the key components and their interactions in the system:



## 1.Sensor Network

- **Rain Sensor:** Measures rainfall levels to assess water availability.



Figure (2.1) Rain sensor

- **DHT11 Sensor:** Collects data on temperature and humidity.



Figure (2.2) DHT11 sensor

- **PH Sensor:** Monitors soil pH levels to determine soil acidity or alkalinity.

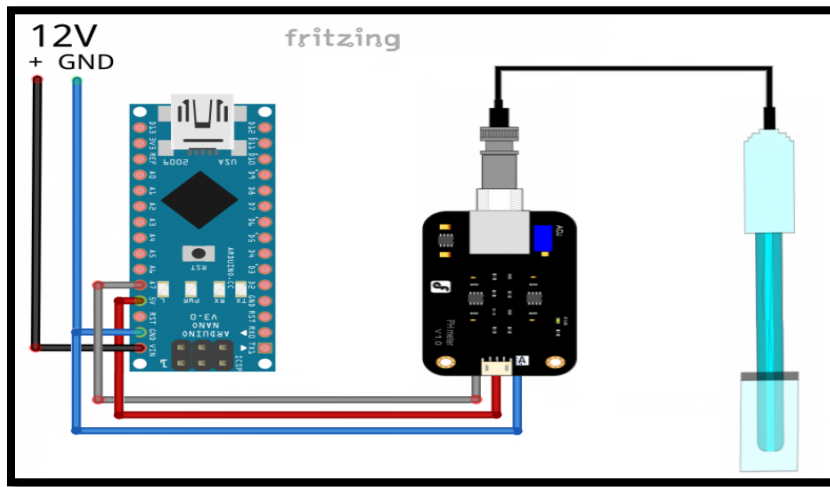


Figure (2.3) PH sensor

- **Soil NPK Sensor:** Measures nitrogen, phosphorus, and potassium levels in the soil to evaluate nutrient content.



Figure (2.4) NPK sensor

## 2. Data Acquisition

### Arduino Controller:

- **Sensor Interfacing:** Connects and communicates with various sensors to gather real-time data.
- **Data Processing:** Processes raw sensor data for transmission.
- **Data Transmission:** Sends processed data to the cloud using a Wi-Fi module.



Figure (2.5) ARDUINO UNO

### 3. Communication Module

#### Wi-Fi Module (e.g., ESP8266 or ESP32 or NRF24L01):

- **Wireless Connectivity:** Provides internet connectivity for the Arduino controller to transmit data to the cloud.
- **Data Transmission Protocol:** Utilizes protocols like HTTP or MQTT for data communication.

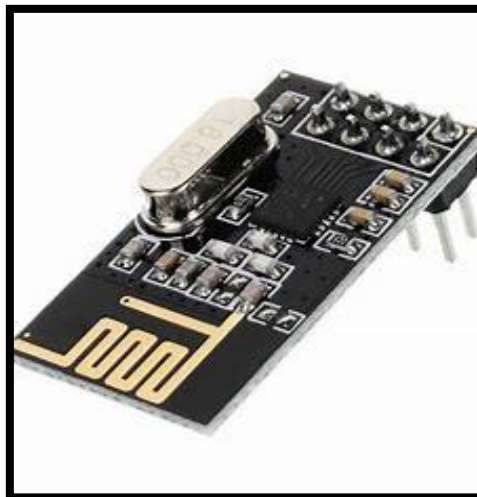
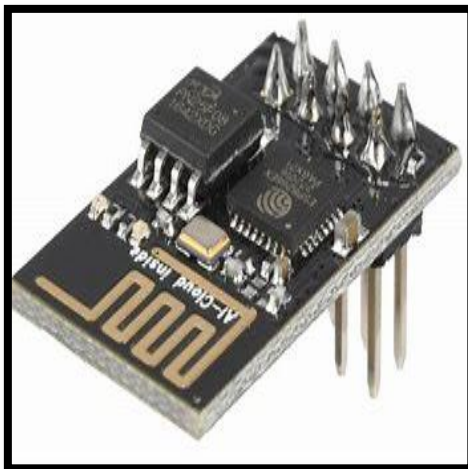


Figure (2.6) WIFI Module

## **4. Cloud Infrastructure**

- **Firestore Cloud:**

- **Real-time Database:** Stores sensor data in real-time, providing a scalable and accessible platform for data management.
- **Data Synchronization:** Ensures real-time updates and synchronization
- **Data Security:** Implements security measures to protect data integrity and privacy.



Figure (2.7) Firebase Cloud symbol

## **5. Machine Learning Model**

### **1. Data Collection:**

- **Historical Data:** Includes historical crop yield data, past weather patterns, and agricultural practices.
- **Real-time Data:** Utilizes real-time sensor data for current environmental and soil conditions.

### **2. Model Training:**

- **Data Preprocessing:** Cleans and preprocesses data for training.
- **Feature Engineering:** Extracts relevant features for model input.
- **Algorithm Selection:** Selects appropriate machine learning algorithms (e.g., decision trees, random forests, neural networks) for crop recommendation.

- **Model Evaluation:** Evaluates model performance using metrics such as accuracy, precision, recall, and F1-score.

### 3. Model Deployment:

- **Integration with Firebase:** Connects the trained model to the Firebase database for real-time analysis.
- **Prediction API:** Provides an API for making crop recommendations based on current data.

## 6. User Interface

### 1. Frontend Application:

- **Web/Mobile Interface:** Develops a user-friendly application for farmers to interact with the system.
- **Input and Output:** Allows farmers to input location and receive crop recommendations.
- **Data Visualization:** Presents data and recommendations through interactive charts, graphs, and maps.

### 2. Notifications and Alerts:

- **Real-time Alerts:** Sends notifications about critical conditions (e.g., low soil moisture, adverse weather) to the farmers' devices.

## 7. System Workflow

1. **Data Collection:** Sensors gather real-time environmental and soil data.
2. **Data Transmission:** Arduino processes and transmits data to Firebase via the Wi-Fi module.
3. **Data Storage:** Firebase stores and synchronizes data in real-time.
4. **Model Analysis:** The machine learning model processes real-time and historical data to generate crop recommendations.

5. **User Interaction:** Farmers access the frontend application to view recommendations and data visualizations.
6. **Decision Making:** Farmers make informed decisions based on the provided recommendations and alerts.

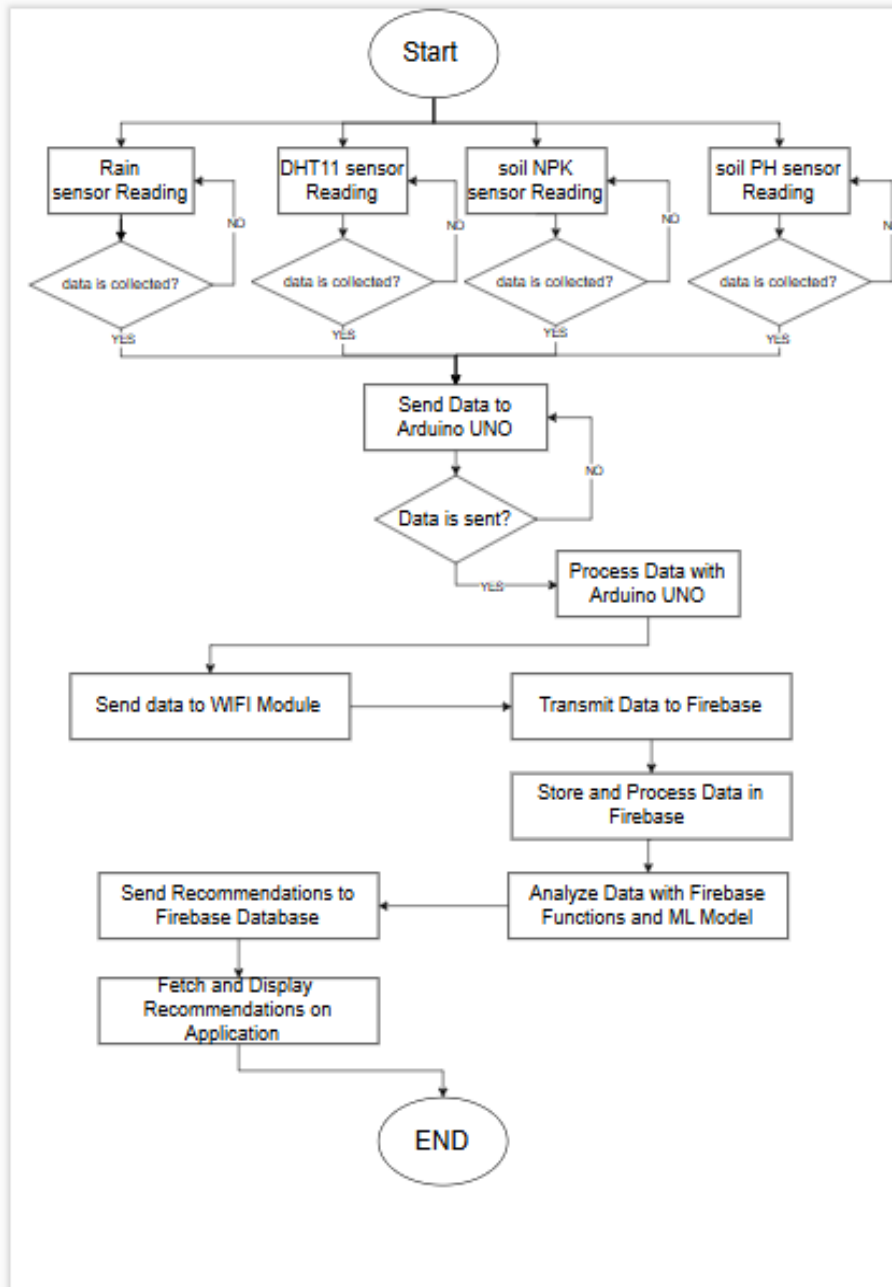


Figure (2.8) System Workflow

## **2.3: Hardware Components**

### **2.3.1: Sensors:**

- **Rain Sensor**

A rain sensor typically detects rainfall and can be used in various applications, including automated irrigation systems, weather monitoring, and other environmental sensing tasks. Here's an overview of a common rain sensor setup and its integration with Arduino:

#### **How It Works:**

- The rain sensor pad detects the presence of water through its conductive tracks.
- When water drops fall on the sensor pad, the resistance changes.
- This change in resistance is read by the control board, which outputs an analog signal.
- The Arduino reads this signal to determine whether it's raining and possibly the intensity of the rainfall.

#### **Connecting the Rain Sensor to Arduino:**

- **VCC:** Connect to 5V on the Arduino.
- **GND:** Connect to the Ground (GND) on the Arduino.
- **AO (Analog Output):** Connect to an Analog pin on the Arduino (e.g., A0).

#### **Code example:**

```
const int pHSensorPin = A1; // Analog pin connected to the pH sensor
float voltage = 0;          // Variable to store the voltage value
float pHValue = 0;          // Variable to store the pH value
void setup() {
  Serial.begin(9600);        // Start the serial communication
}
void loop() {
  int sensorValue = analogRead(pHSensorPin);
```

```

voltage = sensorValue * (5.0 / 1023.0);
pHValue = 3.5 * voltage;
Serial.print("Voltage: ");
Serial.print(voltage);
Serial.print(" V, pH Value: ");
Serial.println(pHValue);
delay(1000); // Delay for 1 second before taking the next reading
}

```

- **PH Sensor**

A pH sensor measures the acidity or alkalinity of a solution, which is crucial for applications like soil monitoring, hydroponics, and water quality testing. Here's how you can integrate a pH sensor with an Arduino for your IoT project:

**Components:**

- ✓ **pH Sensor:** Typically consists of a pH probe and a signal conditioning circuit.
- ✓ **Arduino:** The microcontroller to read and process the pH sensor data.
- ✓ **Connecting Wires:** For connecting the sensor to the Arduino.

**Connecting the pH Sensor to Arduino:**

- ✓ **VCC:** Connect to 5V on the Arduino.
- ✓ **GND:** Connect to the Ground (GND) on the Arduino.
- ✓ **Analog Output:** Connect to an Analog pin on the Arduino (e.g., A1).

**Code Example:**

```

const int pHSensorPin = A1; // Analog pin connected to the pH sensor
float voltage = 0;          // Variable to store the voltage value
float pHValue = 0;          // Variable to store the pH value
void setup() {
  Serial.begin(9600);       // Start the serial communication
}
void loop() {
  int sensorValue = analogRead(pHSensorPin);
  voltage = sensorValue * (5.0 / 1023.0);
  pHValue = 3.5 * voltage;
  Serial.print("Voltage: ");
  Serial.print(voltage);

```



```
Serial.print(" V, pH Value: ");  
Serial.println(pHValue);  
delay(1000); // Delay for 1 second before taking the next reading  
}
```

- **DHT11 Sensor:**

The DHT11 sensor is used for measuring temperature and humidity. It is popular in environmental monitoring and control projects due to its ease of use and accuracy. Here's how you can integrate a DHT11 sensor with an Arduino for your IoT project:

**Components:**

1. **DHT11 Sensor:** Measures temperature and humidity.
2. **Arduino:** The microcontroller to read and process the sensor data.
3. **Connecting Wires:** For connecting the sensor to the Arduino.

**Connecting the DHT11 Sensor to Arduino:**

1. **VCC:** Connect to 5V on the Arduino.
2. **GND:** Connect to the Ground (GND) on the Arduino.
3. **Data:** Connect to a digital pin on the Arduino (e.g., D2).

**Installing the DHT Library:**

1. Open the Arduino IDE.
2. Go to **Sketch > Include Library > Manage Libraries**.
3. Search for "DHT" and install the **DHT sensor library** by Adafruit.

**Code Example:**

```
#include "DHT.h"  
  
#define DHTPIN 2    // Pin where the DHT11 is connected  
  
#define DHTTYPE DHT11 // DHT11 sensor type  
  
DHT dht(DHTPIN, DHTTYPE);  
  
void setup() {
```

```

Serial.begin(9600);
dht.begin();
}
void loop() {
  delay(2000); // Wait a few seconds between measurements
  float humidity = dht.readHumidity(); // Read humidity
  float temperature = dht.readTemperature(); // Read temperature as Celsius
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" *C ");
}

```

### **Usage:**

- **Data Logging:** You can log temperature and humidity data to Firebase Cloud for real-time monitoring and historical analysis.
- **Automated Systems:** Use the sensor data to control HVAC systems, humidifiers, or dehumidifiers.
- **Machine Learning Model:** Incorporate temperature and humidity data as features in your crop recommendation model to optimize growing conditions.

## **Integration with Your IoT Project:**

- **Data Logging to Firebase:** Similar to other sensors, you can send DHT11 data to Firebase Cloud.
- **Real-Time Monitoring:** Display real-time temperature and humidity readings on a web or mobile application.

### • **Soil NPK Sensor**

A Soil NPK sensor measures the levels of Nitrogen (N), Phosphorus (P), and Potassium (K) in the soil, which are essential nutrients for plant growth. Here's an overview of integrating a Soil NPK sensor with an Arduino for your IoT project:

#### **Components:**

1. **Soil NPK Sensor:** This sensor typically measures the electrical conductivity (EC) of the soil, which correlates with the nutrient levels.
2. **Arduino:** The microcontroller to read and process the sensor data.
3. **Connecting Wires:** For connecting the sensor to the Arduino.

#### **How It Works:**

- The Soil NPK sensor measures the EC of the soil, which is influenced by the presence of N, P, and K ions.
- Higher EC values generally indicate higher levels of nutrients in the soil.
- The Arduino reads the analog or digital output from the sensor and interprets it to determine nutrient levels.

#### **Connecting the Soil NPK Sensor to Arduino:**

1. **VCC:** Connect to 5V on the Arduino.
2. **GND:** Connect to the Ground (GND) on the Arduino.
3. **Data Output:** Connect to an Analog or Digital pin on the Arduino.

### **Example Code (Generic Example):**

The specific code depends on the sensor model you are using, as different sensors may have different output formats. Here's a general structure:

```
const int soilSensorPin = A2; // Analog pin connected to the soil NPK sensor
int sensorValue = 0;          // Variable to store the sensor value
void setup() {
  Serial.begin(9600); // Start the serial communication
}
void loop() {
  sensorValue = analogRead(soilSensorPin); // Read the analog value from the sensor
  // Convert sensor value to nutrient levels (specific to sensor calibration)
  // Example: Convert sensor value to NPK levels
  Serial.print("Soil NPK Sensor Value: ");
  Serial.println(sensorValue);
  delay(1000); // Delay for 1 second before taking the next reading
}
```

### **2.3.2: Controller: Arduino UNO**

Great choice with the Arduino Uno for your IoT project! The Arduino Uno is a versatile microcontroller board that's widely used for various applications due to its ease of use and extensive community support. Here's an overview to help you get started or provide further assistance:

#### **Arduino Uno Overview:**

- ✓ **Microcontroller:** ATmega328P
- ✓ **Digital I/O Pins:** 14 (of which 6 provide PWM output)
- ✓ **Analog Input Pins:** 6
- ✓ **Operating Voltage:** 5V

- ✓ **Flash Memory:** 32 KB (0.5 KB used by bootloader)
- ✓ **SRAM:** 2 KB
- ✓ **Clock Speed:** 16 MHz

### **Getting Started:**

- ✓ **Arduino IDE:** Download and install the Arduino IDE from [arduino.cc](https://www.arduino.cc).
- ✓ **Connecting:** Use a USB cable to connect your Arduino Uno to your computer.
- ✓ **Programming:** Write and upload code to your Arduino using the Arduino IDE.
- ✓ **Serial Monitor:** Use the Serial Monitor in the Arduino IDE for debugging and data output.

### **2.3.3: WIFI Module NRF24L01**

The NRF24L01 is a popular wireless communication module used for low-power applications, including IoT projects. Here's how you can integrate it with your Arduino Uno and considerations for power supply:

### **NRF24L01 WiFi Module Integration with Arduino Uno:**

#### **Components Needed:**

- ✓ **NRF24L01 Module:** Wireless communication module.
- ✓ **Arduino Uno:** The microcontroller to interface with the NRF24L01.
- ✓ **Connecting Wires:** For connecting the NRF24L01 module to the Arduino.

#### **Wiring the NRF24L01 Module to Arduino Uno:**

- ✓ **VCC:** Connect to 3.3V (not 5V) on the Arduino Uno.
- ✓ **GND:** Connect to Ground (GND) on the Arduino.
- ✓ **CE (Chip Enable):** Connect to digital pin 9 on the Arduino.
- ✓ **CSN (Chip Select Not):** Connect to digital pin 10 on the Arduino.
- ✓ **SCK (Serial Clock):** Connect to digital pin 13 (SCK) on the Arduino.

- ✓ **MOSI (Master Out Slave In):** Connect to digital pin 11 (MOSI) on the Arduino.
- ✓ **MISO (Master In Slave Out):** Connect to digital pin 12 (MISO) on the Arduino.
- ✓ **IRQ (Interrupt Request):** Optionally connect to a digital pin on the Arduino if using interrupts.

### **Arduino Library:**

Use the RF24 library for Arduino to interface with the NRF24L01 module. You can download it through the Arduino Library Manager.

### **Example Code (Basic Transmitting and Receiving):**

Here's a simple example to get you started with sending and receiving data using NRF24L01 modules:

```
#include <SPI.h>
#include <RF24.h>
RF24 radio(9, 10); // CE, CSN pins
const byte address[6] = "00001";
void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN); // Set power level
  radio.stopListening(); // Start transmission mode
}
void loop() {
  const char text[] = "Hello, world!";
  radio.write(&text, sizeof(text));
  Serial.println("Data sent.");
  delay(1000);
}
```

### **Power Supply Considerations:**

- ✓ **NRF24L01 Voltage:** Operates at 3.3V. Ensure you use a regulated 3.3V supply for the module.

- ✓ **Arduino Uno:** Provides 3.3V and 5V outputs. Use the 3.3V output for the NRF24L01 module.
- ✓ **Power Consumption:** NRF24L01 modules are low-power, but consider overall power budget if using batteries or a low-power setup.
- ✓ **Decoupling Capacitors:** Place capacitors (typically 10 $\mu$ F and 0.1 $\mu$ F) near the NRF24L01 module to stabilize power supply voltage.

### **Integration with Your IoT Project:**

- ✓ **Data Transmission:** Send sensor data wirelessly to a base station or server.
- ✓ **Mesh Networking:** Create a mesh network for extended coverage.
- ✓ **Power Efficiency:** Optimize power usage for battery-operated applications.

### **Next Steps:**

- ✓ Ensure your NRF24L01 modules are compatible with each other in terms of frequency (2.4GHz) and settings.
- ✓ Test communication between modules and integrate them into your existing Arduino Uno setup.
- ✓ Monitor power consumption to ensure efficient operation, especially in battery-powered scenarios.

## 2.4: Block Diagram

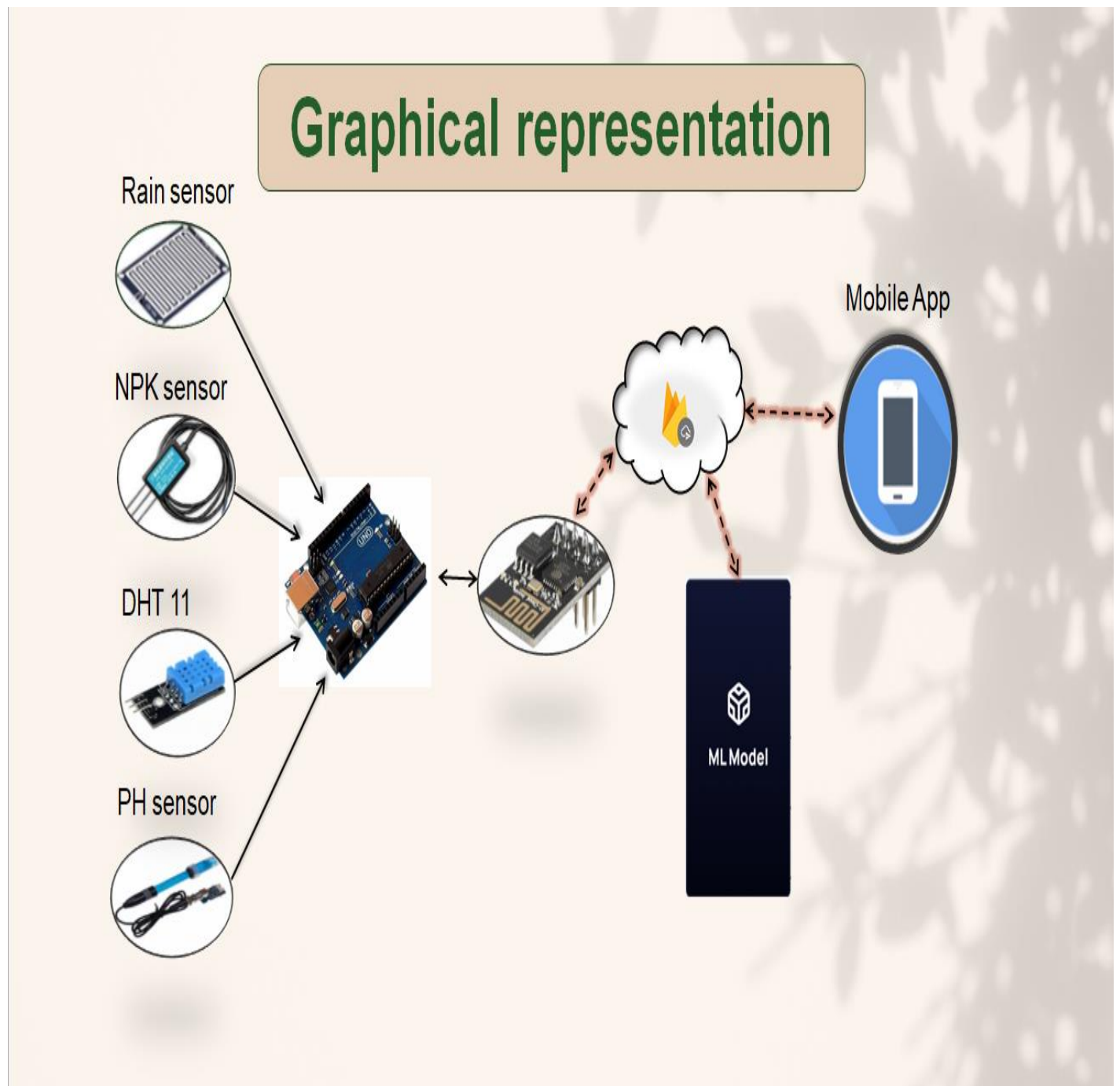


Figure (2.9) Graphical Representation



## 2.5: Circuit Diagram for Components

### Circuit Diagram

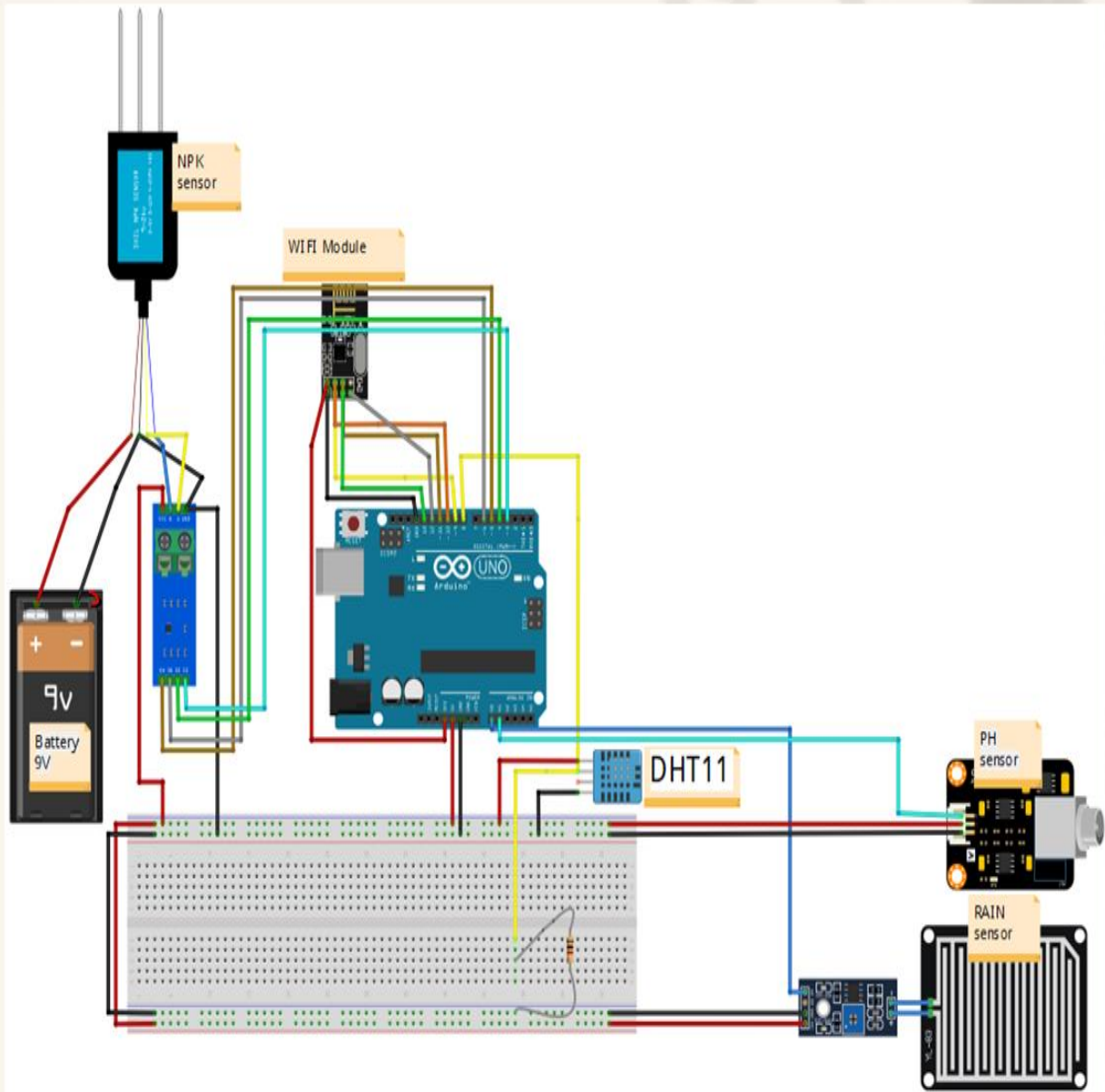


Figure (1.10) Circuit Diagram

# **Chapter 3**

## **Machine Learning**

### **3.1: Introduction to Machine Learning**

#### **What is Machine Learning?**

- Machine learning: is a subset of artificial intelligence that involves training computer systems to learn from data, without being explicitly programmed. It is a process of using algorithms and statistical models to enable computers to improve their performance on a specific task over time, based on the feedback they receive from the data they process.
- Machine learning has many applications, including image recognition, natural language processing, fraud detection, recommendation systems, and autonomous vehicles, among others.

### **3.2: Types of machine learning:**

Machine learning algorithms can be categorized into three main types:

- **Supervised Learning:** This involves training a model on labeled data, where the input-output pairs are known. The goal is to learn a mapping from inputs to outputs to make predictions on new, unseen data.
- **Unsupervised Learning:** This method deals with unlabeled data and aims to identify patterns or structures within the data. Common techniques include clustering and dimensionality reduction.
- **Reinforcement Learning:** This type focuses on training agents to make sequences of decisions by rewarding or punishing them based on their actions. The agent learns to maximize cumulative rewards through trial and error.

## **Why Machine learning?**

- Explosion of available data “big data” (especially with the advent of the Internet)
- Increasing computational power (parallel programming and distributed systems)
- Growing progress in available algorithms and theory developed by researchers
- Increasing support from industries

## **Applications of Machine Learning**

- Machine learning (ML) has a wide range of applications across various industries, revolutionizing how data is utilized to make predictions, optimize processes, and enhance decision-making.
- Below are some key real-world applications of machine learning and its importance in engineering projects.

## **Real-world Applications across Various Industries :**

- Healthcare,
- Finance,
- Retail,
- Manufacturing, etc....

## **Importance in Engineering Projects :**

- Optimization of Processes
- Enhanced Decision-Making
- Innovation and Efficiency
- Cost Reduction
- Scalability
- Risk Management

### **3.3: Machine Learning Workflow**

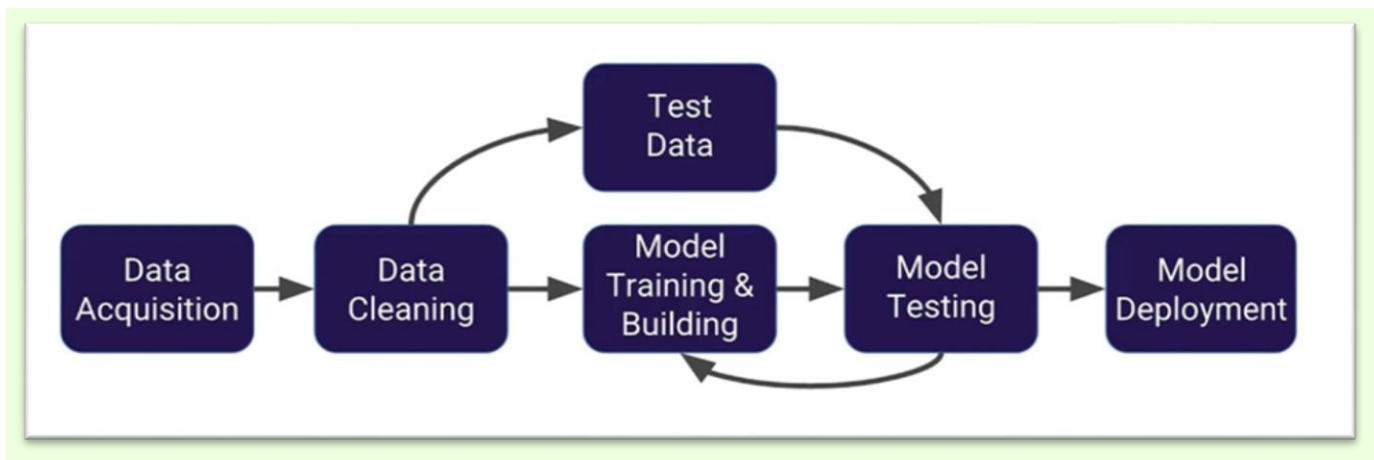


Figure (3.1) ML Workflow

#### **1. Data Collection:**

The first step is to collect and preprocess the data that will be used to train the machine learning algorithm. This may involve gathering data from various sources, cleaning and formatting the data, and splitting the data into training, validation, and testing sets.

#### **2. Data preparation:**

Once the data has been collected, it needs to be preprocessed so that it can be used for training the machine learning algorithm. This may involve tasks such as feature selection, feature scaling, and handling missing values.

#### **3. Model selection:**

The next step is to choose a machine learning model that is appropriate for the problem at hand. This may involve evaluating different models based on their accuracy, complexity, and interpretability.

#### **4. Model training:**

Once the model has been selected, it needs to be trained on the training data. During training, the model learns the relationships between the input features and the output variable.

## **5. Model evaluation:**

After the model has been trained, it needs to be evaluated on the validation set to see how well it performs on unseen data. This helps to identify any problems with the model and fine-tune its parameters.

## **6. Hyperparameter tuning:**

Machine learning models often have hyperparameters that need to be set before training. Hyperparameters are different from model parameters, as they control the behavior of the model and are not learned during training.

Hyperparameter tuning involves finding the best combination of hyperparameters those results in the best model performance.

## **7. Model testing:**

Once the model has been trained and evaluated, it needs to be tested on the testing set to see how well it generalizes to new, unseen data. This helps to ensure that the model is not over-fitting to the training data.

## **8. Deployment:**

Once the model has been tested and validated, it can be deployed in a production environment where it can be used to make predictions on new, real-world data.

➤ Note that the machine learning process is iterative, and each step may need to be repeated multiple times until a satisfactory model is obtained.

## **3.4: Algorithms of Machine learning**

### **Overview of Algorithms**

In machine learning, various algorithms are available to solve different types of problems, ranging from classification and regression to clustering and dimensionality reduction. Selecting the right algorithm is crucial for achieving the best performance on a given dataset. This section provides a brief overview of the algorithms considered for our project and justifies the choice of the specific algorithm used.

## **Brief Overview of Various Algorithms Considered**

- 1. Linear Regression:** Used for predicting a continuous target variable based on one or more predictor variables. It is simple and interpretable but can be limited in handling non-linear relationships.
- 2. Logistic Regression:** Ideal for binary classification problems. It models the probability of a binary outcome using a logistic function.
- 3. Support Vector Machine (SVM):** Suitable for classification tasks. It finds the optimal hyperplane that separates the classes in the feature space.
- 4. K-Nearest Neighbors (KNN):** A non-parametric method used for classification and regression. It classifies a sample based on the majority class among its k nearest neighbors.
- 5. Decision Trees:** A versatile algorithm that can be used for both classification and regression tasks. It splits the data into subsets based on the value of input features.
- 6. Random Forest:** An ensemble learning method that constructs multiple decision trees and merges their results to improve accuracy and control overfitting.
- 7. Neural Networks:** Composed of layers of interconnected nodes, they are particularly powerful for handling complex patterns in data, such as in image and speech recognition.
- 8. K-Means Clustering:** Used for partitioning a dataset into k distinct clusters based on feature similarity.
- 9. Gradient Boosting Machine (GBM) Model:** A model that is an ensemble of weak models and combines their outputs to create a strong model.
- 10. Clustering Models:** Models that group similar data points together based on their features, such as K-Means and Hierarchical Clustering.

➤ These are just a few examples of the many machine learning models available. The choice of model depends on the specific problem and the nature of the data being

used. It's important to experiment with different models and compare their performance to find the best one for the task at hand.

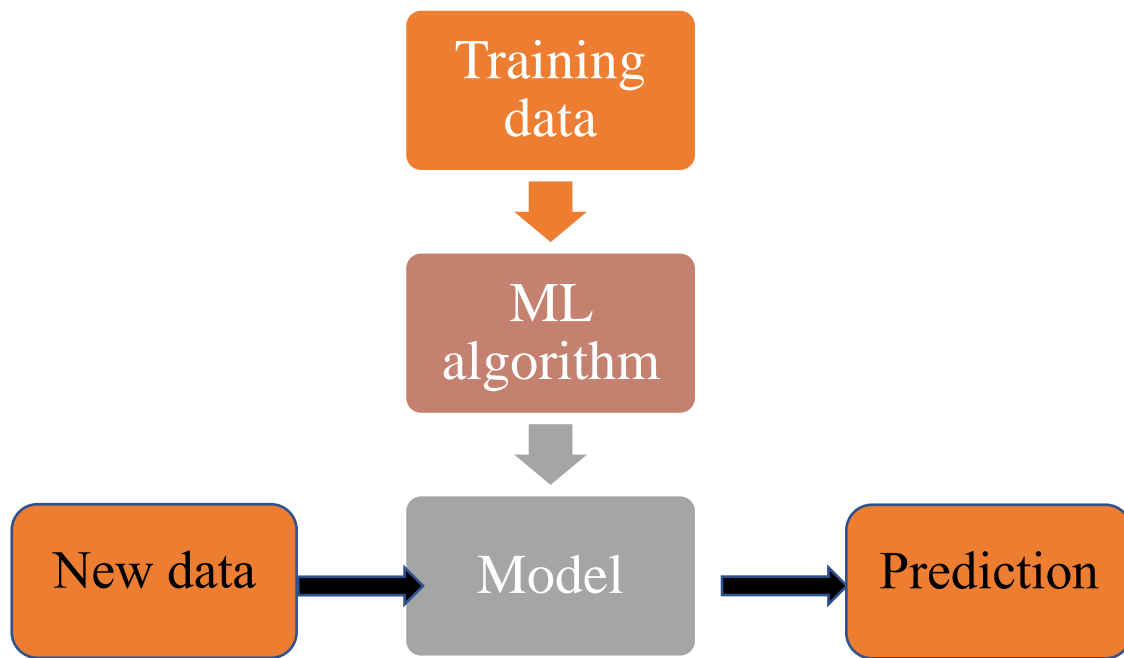


Figure (3.2) Machine learning Algorithm overview

### **3.5: Machine learning in our project**

First of all, we think about how to gain the best beneficial from the data which is collected from the sensors in our system (Smart Agriculture). We found that we can train a machine learning algorithm (K Nearest Neighbor Classifier) in a data which is look like our data to make a model with high accuracy up to 97.58% to use it in a crops recommendation system to help farming the best crop for the available conditions and the best production.

#### **3.5.1: K-Nearest Neighbors (KNN) Algorithm**

##### **Introduction to KNN**

K-Nearest Neighbors (KNN) is a simple, yet powerful algorithm used for both classification and regression tasks in machine learning. It operates on the principle that similar instances are near each other in the feature space. The KNN algorithm is a type

of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until function evaluation.

### **How KNN Works**

- **Choosing the Number of Neighbors (K):** The algorithm starts by choosing a value for K, the number of nearest neighbors to consider. The value of K is usually determined through experimentation and cross-validation.
- **Calculating Distances:** For each instance in the dataset, the algorithm calculates the distance between the new instance (query point) and all existing instances using a distance metric (e.g., Euclidean distance).
- **Finding Neighbors:** The algorithm identifies the K instances that are closest to the query point.
- **Voting for Classification:** In the case of classification, the algorithm assigns the class that is most common among the K nearest neighbors. For regression, it averages the values of the K nearest neighbors to predict the outcome.

### **Advantages of KNN**

- **Simplicity:** KNN is easy to understand and implement.
- **No Training Phase:** since KNN is a lazy learner, there is an explicit training phase. The training data is used directly during the prediction phase.
- **Versatility:** KNN can be used for both classification and regression problems.

### **Disadvantages of KNN**

- **Computationally Expensive:** KNN requires the calculation of distances for each instance in the dataset during prediction, which can be slow for large datasets.
- **Memory Intensive:** As it stores all training data, KNN can require significant memory resources.
- **Sensitive to Irrelevant Features:** The performance of KNN can be degraded by irrelevant or redundant features. Proper feature selection or dimensionality reduction techniques should be applied.

### **3.5.2: The Data**

The data consists of 2200 rows and 8 columns



```
df.head()
```

	N	P	K	Temperature	Humidity	ph	Rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

Figure (3.3) Data Head

- **Data fields**

N - Ratio of Nitrogen content in soil

P - Ratio of Phosphorous content in soil

K - Ratio of Potassium content in soil

Temperature - temperature in degree Celsius

Humidity - relative humidity in %

Ph. - ph. value of the soil

Rainfall - rainfall in mm

Label - Crop Name

### **3.5.3: Model steps**

#### **Importing Libraries :**

#### **Brief Explanation of Necessary Libraries**

1. NumPy: Essential for numerical computations, providing array support and mathematical functions.
2. Pandas: Used for data manipulation and analysis with DataFrames.
3. Scikit-learn: Offers machine learning tools. Specific imports include:

- a. `LogisticRegression` and `RandomForestClassifier` for modeling.
  - b. `train\_test\_split` and `StandardScaler` for data preparation.
  - c. Metrics like `accuracy\_score` for model evaluation.
4. SciPy: Adds scientific computing capabilities, including interpolation.
  5. Matplotlib: Creates static visualizations like line and scatter plots.
  6. Seaborn: Enhances Matplotlib's statistical graphics.
  7. Plotly: Generates interactive visualizations.
  8. Utility: Shuffles data arrays for unbiased datasets.

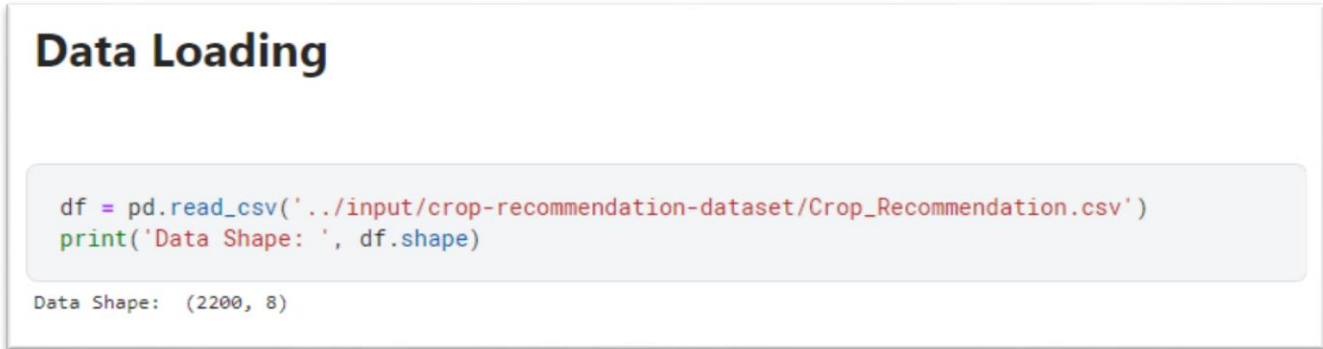
### **Example code snippet for imports**

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from scipy.interpolate import make_interp_spline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score as acc
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import plotly.graph_objects as go
from sklearn.utils import shuffle
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score
from sklearn.datasets import make_classification
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import RandomForestClassifier
```

Figure (3.4) libraries imported

## Data Loading and Preprocessing

### - Loading the dataset



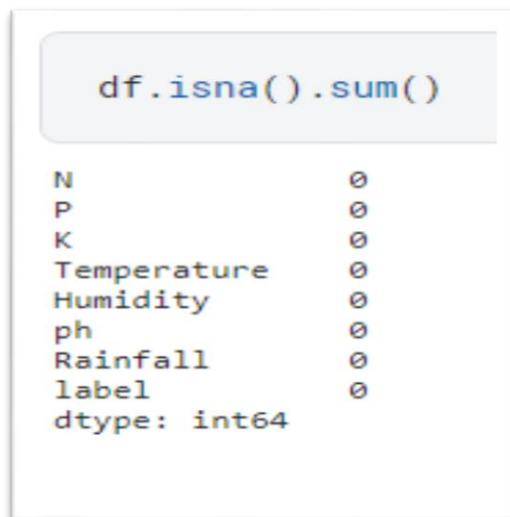
```
df = pd.read_csv('../input/crop-recommendation-dataset/Crop_Recommendation.csv')
print('Data Shape: ', df.shape)
```

Data Shape: (2200, 8)

Figure (3.5) Data Loading

### - Preprocessing steps taken in the project

- Check nulls and missing values:



```
df.isna().sum()
```

N	0
P	0
K	0
Temperature	0
Humidity	0
ph	0
Rainfall	0
label	0
dtype:	int64

Figure (3.6) Check Missing values

- Correlation matrix (corr)

Calculating the correlation matrix (corr) is considered a part of the preprocessing steps in data analysis. This step can be useful for understanding the relationships between different variables in a dataset and identifying variables that may have a significant impact on the target variable we are trying to predict.

Text(0.5, 1.0, 'Correlation Matrix')

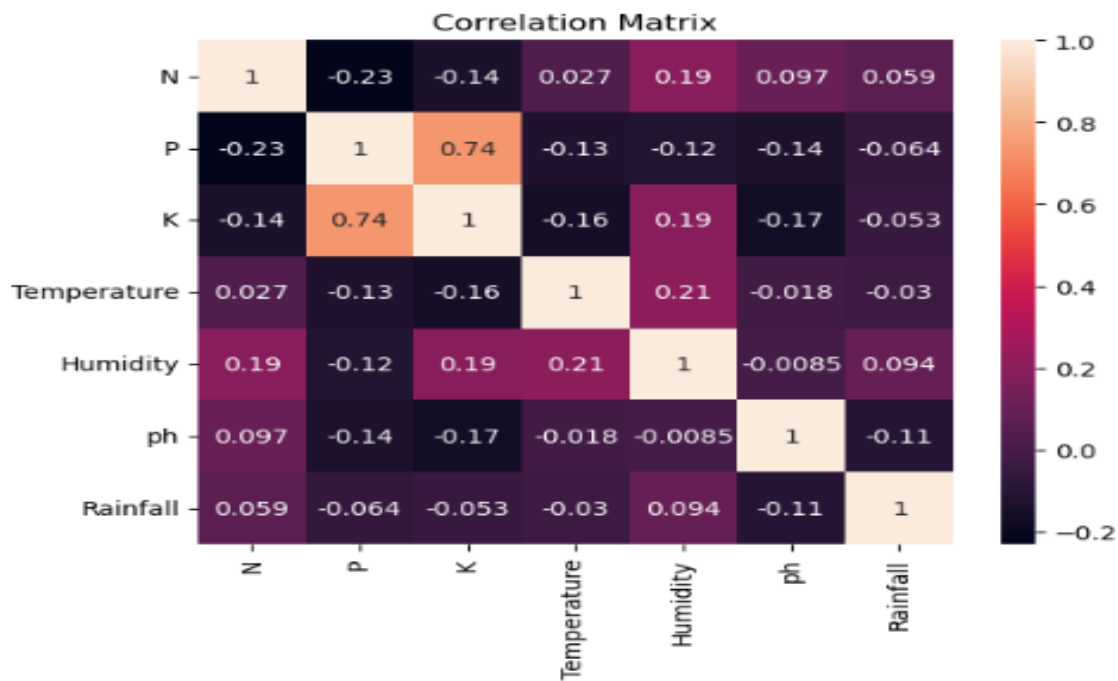


Figure (3.7) Correlation Matrix

## **Building & Training The Model:**

- Split The Data to x and y:

```
y = xdf['label_codes'] # Targeted Values Selection
X = xdf[selected_features] # Independent Values
```

Figure (3.8) Data split

- Train and Test Data Splitting:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
print(X.shape,X_train.shape,X_test.shape)
```

(2200, 7) (1540, 7) (660, 7)

Figure (3.9) Train and Test The Model

- Building The Model

```
from sklearn.neighbors import KNeighborsClassifier  
  
knn_clf=KNeighborsClassifier()  
model = MultiOutputClassifier(knn_clf, n_jobs=-1)  
model.fit(x_train, y_train)  
knn_pred = model.predict(x_test)
```

Figure (3.10) Building Model

## Model Evaluation

Recommended crop is watermelon

Figure (3.11) Model Evaluation

## Model Saving and Loading

- Saving the model to a file
- Loading the model for future use
- Example code snippets

### Crop Recommendation Application

103	17	51	25.111892	80.026213	6.209888	44.206570	Predict
-----	----	----	-----------	-----------	----------	-----------	---------

Figure (3.12) Model Saving and Loading

**#Saving the trained KNN model.**

```
import pickle

ML_pkl_filename = 'CropRecomendation.pkl'
# Open the file to save as pkl file
ML_Model_pkl = open(ML_pkl_filename, 'wb')
pickle.dump(model, ML_Model_pkl)
# Close the pickle instances
ML_Model_pkl.close()
```

Figure (3.13) saving the model

**Kaggle Link:** <https://www.kaggle.com/code/walaaadel2/crop-recommendation-model>

# Chapter 4

## Deep Learning

### 4.1: Introduction

Deep learning has emerged as a transformative technology, impacting numerous fields by enabling machines to learn from large datasets. In agriculture, one significant application is plant disease detection. Early and accurate detection of plant diseases can help farmers take timely actions to protect crops, improve yields, and reduce economic losses. This chapter focuses on using Convolutional Neural Networks (CNNs) for detecting plant diseases from images of plant leaves.

### 4.2: Deep Learning and Its Algorithms

Deep learning is a subset of machine learning that involves training neural networks with many layers to learn hierarchical representations of data. These networks, known as deep neural networks Figure (4.1), are capable of learning complex patterns and making predictions with high accuracy. It is inspired by the structure and function of the human brain, where multiple layers of interconnected neurons process information

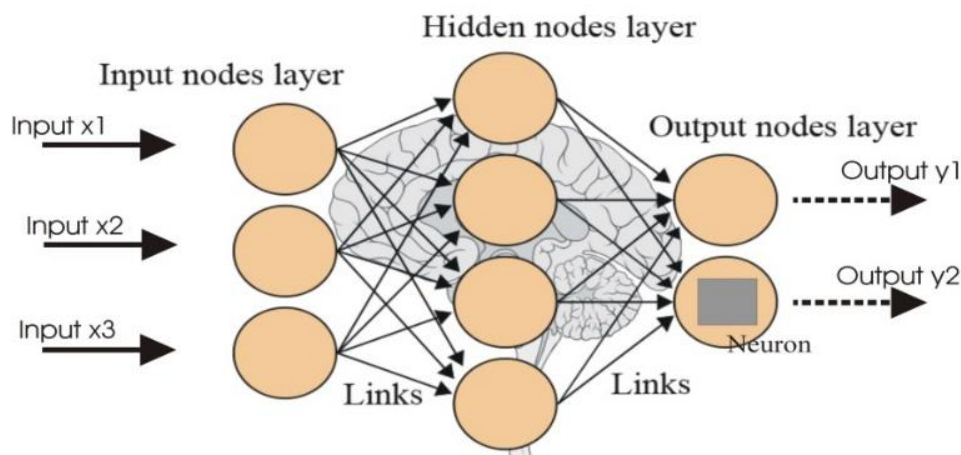


Figure (4.1) deep neural network

## **Key Concepts in Deep Learning**

- **Neurons and Layers:** The basic building blocks of neural networks. Layers are composed of neurons, each performing a weighted sum of inputs followed by an activation function.
- **Activation Functions:** Functions such as ReLU, sigmoid, and tanh that introduce non-linearity into the model, enabling it to learn complex patterns.
- **Backpropagation:** The algorithm used to train neural networks by adjusting weights based on the error gradient.
- **Optimization Algorithms:** Methods like stochastic gradient descent (SGD) and Adam used to minimize the loss function during training.

### **4.2.1: Popular Deep Learning Algorithms**

- **Convolutional Neural Networks (CNNs)**  
It specialized for processing data such as images. They are highly effective in tasks like image recognition and object detection.
- **Recurrent Neural Networks (RNNs)**  
It designed for sequential data, such as time series or text, where the order of inputs matters. Variants like LSTM and GRU address the vanishing gradient problem.
- **Generative Adversarial Networks (GANs)**  
It consists of two neural networks, a generator and a discriminator, that compete against each other to produce realistic synthetic data.
- **Autoencoders**  
It used for unsupervised learning tasks like dimensionality reduction and anomaly detection. They learn to compress data into a lower-dimensional representation and then reconstruct it.



## **4.2.2: Convolutional Neural Networks (CNNs)**

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery.

### **4.2.2.1: Introduction to CNNs**

Over the years, CNNs have become fundamental in computer vision tasks such as image classification, object detection, and segmentation. Modern CNNs are implemented using programming languages like Python and leverage advanced techniques to extract and learn features from images. Hyperparameters, optimization techniques, and regularization methods are crucial for training these models effectively.

### **4.2.2.2: How CNNs Work**

CNNs Figure (4.2) work by applying a series of convolutional and pooling operations to the input image, followed by fully connected layers that classify the image based on the learned features. The convolutional layers act as feature extractors, while the fully connected

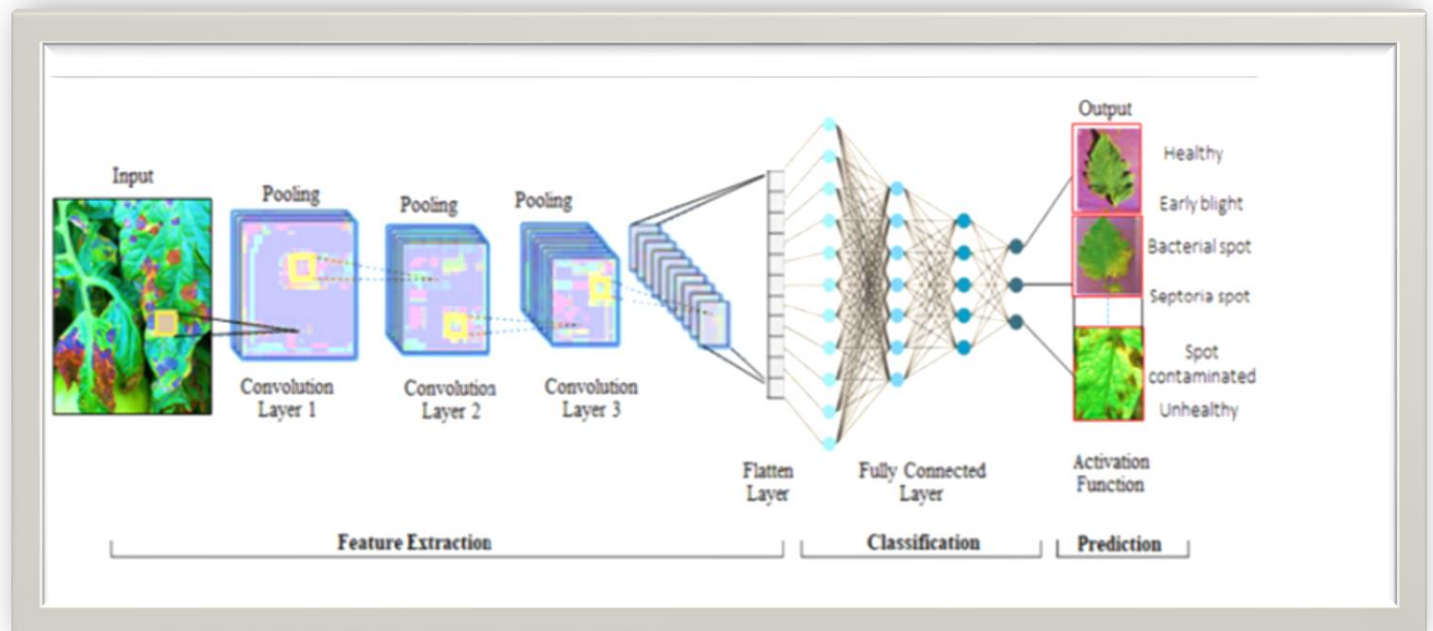


Figure (4.2) Convolutional Neural Network

### **4.2.2.3: Components of CNNs**

#### **1. Convolutional Layer**

- **Purpose:** The convolutional layer is the core building block of a CNN. It applies convolutional operations to the input, extracting features like edges, textures, and shapes.
- **Operation:** A set of filters (also known as kernels) slide over the input image, performing element-wise multiplications and summing up the results to produce feature maps. Each filter detects a specific feature in the input image.
- **Hyperparameters:**
  - Filter Size (Kernel Size)
  - Stride
  - Padding

#### **2. Activation Function**

- **Purpose:** Activation functions are essential components in neural networks as they introduce non-linearity, enabling the networks to learn and represent complex patterns and functions
- **Common Activation Functions:**
  - **ReLU (Rectified Linear Unit):** Outputs zero for negative input values and the input itself for positive values.
  - **Sigmoid:** It squashes the input to a range between 0 and 1.
  - **Tanh:** It squashes the input to a range between -1 and 1.
  - **Leaky ReLU:** Similar to ReLU but allows a small, non-zero gradient when the input is negative.
  - **Softmax:** Converts a vector of values into a probability distribution.

### **3. Pooling Layer**

- **Purpose:** Reduces the spatial dimensions (width and height) of the feature maps, decreasing the computational load and controlling overfitting.
- **Types**
  - **Max Pooling:** Takes the maximum value from each patch of the feature map.
  - **Average Pooling:** Takes the average value from each patch of the feature map.
- **Hyperparameters**
  - **Pool Size:** It defines the size of the pooling window (e.g., 2x2, 3x3).
  - **Stride:** It determines the step size by which the pooling window moves across the feature map.

### **4. Fully Connected (Dense) Layer**

- **Purpose:** Connects every neuron in one layer to every neuron in the next layer, consolidating the learned features for classification or regression tasks.
- **Operation:** Flattens the feature maps from the previous layers into a single vector and applies a series of dense layers to make predictions.

### **5. Dropout Layer**

- **Purpose:** Reduces overfitting by randomly setting a fraction of input units to zero during training.
- **Operation:** Each unit (neuron) is dropped with a probability defined by a hyperparameter (dropout rate), preventing the network from becoming too reliant on any particular unit.

### **6. Batch Normalization Layer**

- **Purpose:** Normalizes the input of each mini-batch to improve training speed and stability.

- **Operation:** Applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

## 7. Output Layer

- **Purpose:** Produces the final output of the network, typically used for classification or regression.
- **Common Activation Functions for Output Layer:**
  - **Softmax**  
Used for multi-class classification, converting logits to probabilities.
  - **Sigmoid**  
Used for binary classification, outputting a probability between 0 and 1.
  - **Linea**  
Used for regression tasks, outputting a continuous value.

### 4.3: Main Parts of CNN Model

- Data Collection
- Data Exploration and Analysis
- Model Architecture
- Model Training
- Model Evaluation
- Model Deployment

## **Data Collection**

### **Description of the Dataset**

The dataset used for this project is the “New Plant Diseases Dataset”, which consists of images of plant leaves categorized into various classes based on the type of disease or healthy condition. The dataset includes over 87k images spanning 38 different classes, covering 14 crop species and their respective diseases.

### **Dataset Sources**

The “New Plant Diseases Dataset” is publicly available and was created to support the development of plant disease diagnostics. It is widely used in research for training and testing deep learning models for plant disease detection.

## **Data Preprocessing**

### **Image Preprocessing Techniques**

Data preprocessing is a critical step to ensure the model performs well. The following techniques are applied:

- **Resizing**
  - ✓ Images are resized to a uniform size (e.g., 256x256 pixels) to ensure consistency in the input data.
- **Normalization**
  - ✓ Pixel values are normalized to a range of 0 to 1 to improve convergence during training.
- **Data Augmentation**
  - ✓ Techniques such as rotation, flipping, and zooming are applied to increase the diversity of the training data and prevent overfitting.

## Data Exploration and Analysis

Some views from code to know number of plants Figure (4.3), shape Figure (4.4) and dimension of each image and number of images in each class Figure (4.5).

```
Number of plants: 14
```

```
-----  
Unique Plants are:
```

```
['Tomato', 'Grape', 'Orange', 'Soybean', 'Squash', 'Potato', 'Corn',  
'Strawberry', 'Peach', 'Apple', 'Blueberry', 'Cherry', 'Pepper,', 'Ras  
pberry']
```

Figure (4.3) Number of Plants

```
print("Image shape:", img.shape)
```

```
Image shape: (256, 256, 3)
```

Figure (4.4) Image Shape

Validation data images count per class :

	No. of images
Tomato__Late_blight	463
Tomato__healthy	481
Grape__healthy	423
Orange__Haunglongbing_(Citrus_greening)	503
Soybean__healthy	505
Squash__Powdery_mildew	434
Potato__healthy	456
Corn_(maize)__Northern_Leaf_Blight	477
Tomato__Early_blight	480
Tomato__Septoria_leaf_spot	436
Corn_(maize)__Cercospora_leaf_spot Gray_leaf_spot	410
Strawberry__Leaf_scorch	444
Peach__healthy	432
Apple__Apple_scab	504
Tomato__Tomato_Yellow_Leaf_Curl_Virus	490
Tomato__Bacterial_spot	425
Apple__Black_rot	497
Blueberry__healthy	454
Cherry_(including_sour)__Powdery_mildew	421
Peach__Bacterial_spot	459
Apple__Cedar_apple_rust	440
Tomato__Target_Spot	457
Pepper,_bell__healthy	497
...	
Tomato__Leaf_Mold	470
Tomato__Spider_mites Two-spotted_spider_mite	435
Pepper,_bell__Bacterial_spot	478
Corn_(maize)__healthy	465

Figure (4.5) Number of Images for each Class

## Model Architecture

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 32)	896
conv2d_1 (Conv2D)	(None, 256, 256, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 85, 85, 32)	0
batch_normalization (BatchNormalization)	(None, 85, 85, 32)	128
conv2d_2 (Conv2D)	(None, 85, 85, 64)	18,496
conv2d_3 (Conv2D)	(None, 85, 85, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 28, 28, 64)	0
conv2d_4 (Conv2D)	(None, 28, 28, 128)	73,856
conv2d_5 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 9, 9, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 9, 9, 128)	512
conv2d_6 (Conv2D)	(None, 9, 9, 256)	295,168
conv2d_7 (Conv2D)	(None, 9, 9, 256)	590,880
conv2d_8 (Conv2D)	(None, 9, 9, 512)	3,277,312
conv2d_9 (Conv2D)	(None, 9, 9, 512)	6,554,112
flatten (Flatten)	(None, 41472)	0
dense (Dense)	(None, 1568)	65,029,664
dropout (Dropout)	(None, 1568)	0
batch_normalization_2 (BatchNormalization)	(None, 1568)	6,272
dense_1 (Dense)	(None, 38)	59,622

Figure (4.6) The Model Layers

The CNN model is created using the 'keras.Sequential' API, which allows for the linear stacking of layers Figure (4.6).

### Input Layer

- **Input Shape:** (256, 256, 3)
- This layer expects images of size 256x256 with 3 color channels (RGB).

### Convolutional and Pooling Layers

- **Conv2D Layers**

- **First Block**

- Two convolutional layers with 32 filters each, kernel size of (3, 3) ReLU activation, and same padding.
- Followed by a MaxPooling layer with pool size (3, 3).
- Batch Normalization for regularization.

- **Second Block**

- Two convolutional layers with 64 filters each, kernel size of (3, 3), ReLU activation, and same padding.
- Followed by a MaxPooling layer with pool size (3, 3).

- **Third Block**

- Two convolutional layers with 128 filters each, kernel size of (3, 3), ReLU activation, and same padding.
- Followed by a MaxPooling layer with pool size (3, 3).
- Batch Normalization for regularization.

- **Fourth Block**

- Two convolutional layers with 256 filters each, kernel size of (3, 3), ReLU activation, and same padding.

- **Fifth Block**

- Two convolutional layers with 512 filters each, kernel size of (5, 5), ReLU activation, and same padding.

## **Flatten Layer**

Converts the 2D output of the convolutional layers into a 1D feature vector.

## **Fully Connected Layers**

### **Dense Layers**

- First Dense layer with 1568 units and ReLU activation.
- Followed by a Dropout layer with a dropout rate of 0.5 for regularization.
- Batch Normalization for regularization.



- Final Dense layer with 38 units and softmax activation, corresponding to the number of output classes.

## **Model Training**

The fit method returns a history object, which contains information about the training process, including the loss and accuracy for both the training and validation sets for each epoch. This information can be used to visualize the training progress and evaluate the model's performance over time.

```
# Train the model on the training data and validate it on the validation data ^_^
history = model.fit(
    train_gen, # Training data generator
    validation_data=vaild_gen, # Validation data generator
    epochs=10 # Number of epochs for training
)
```

Figure (4.7) Model Training Process

## **Model Evaluation**

During evaluation, the model makes predictions for each batch of test data, and the loss function is computed by comparing the predictions to the true labels. The accuracy is calculated as the proportion of correctly predicted labels.

**Train Accuracy: 99.09 %**

**Test Accuracy: 97.61 %**

true:PotatoEarlyBlight1.JPG  
pred\_class:20  
Classname: Potato\_\_Early\_blight



true:AppleCedarRust3.JPG  
pred\_class:2  
Classname: Apple\_\_Cedar\_apple\_rust



## **Model Deployment**

After successfully training and evaluating the CNN model, the next step is to deploy it so that it can be used in a production environment. This section outlines the key steps and considerations for deploying the model.

First step is to save the model Figure (4.8) and after that we would begin in the backend part.

### **Finally..Saving our model 📁**

```
# save it as a h5 file ^_^  
from tensorflow.keras.models import load_model  
model.save('model.h5')
```

Figure (4.8) Saving the Model

**Kaggle Link:** <https://www.kaggle.com/code/tokashawky/new-plant-diseases-dataset-cnn>

# Chapter 5

## Backend Implementation

### 5.1: Introduction

Backend development refers to the server-side of a web application, responsible for managing data and ensuring that everything on the client side (frontend) runs smoothly.

It is everything that the users don't see and contains behind-the-scenes activities that occur when performing any action on a website.

It involves the creation and maintenance of the technology that powers the components users interact with on the frontend.

### 5.2: Key Components of Backend Development

1. **Server:** The server is the system that processes requests from clients. It hosts, processes, and responds to client requests.
2. **Database:** Databases store and organize data. Common databases include MySQL, PostgreSQL, MongoDB, and SQLite.
3. **Application:** This is the core logic that handles data processing, requests, and responses. It often involves using frameworks and programming languages like Node.js, Django, Flask, Ruby on Rails, and Spring Boot.

### 5.3: Common Backend Languages

- **JavaScript (Node.js):** Popular due to its non-blocking, event-driven architecture.
- **Python:** Known for its simplicity and readability, with frameworks like Django and Flask.

- **Java:** Widely used for large-scale applications, often with Spring Boot.
- **Ruby:** Known for its elegant syntax, with Ruby on Rails being a popular framework.
- **PHP:** Common in web development, especially with WordPress.

### **5.3.1: Backend with Python**

Python is an ideal choice for backend development, particularly when integrating machine learning or deep learning models. Its extensive libraries, such as TensorFlow, PyTorch, and scikit-learn, offer powerful tools for model building and deployment. Python's simplicity and readability make it easier to manage complex algorithms and data structures inherent in these models. Moreover, frameworks like Flask (5.1) and Django (5.2) facilitate seamless API creation, enabling efficient interaction between the backend and machine learning components. This synergy allows for the development of intelligent, data-driven applications that can learn and adapt over time.



Flask

Figure (5.1) Flask



Figure (5.2) django

#### **5.3.1.1: Flask**

Flask is a web framework that allows developers to build lightweight web applications quickly and easily with Flask Libraries. It was developed by Armin Ronacher, leader of the International Group of Python Enthusiasts(POCCO). It is basically based on the WSGI toolkit and Jinja2 templating engine.

It is a flexible web framework for Python, designed to make it easy to get started with web development while providing the tools needed to build robust applications.

## **Key Features of Flask**

1. **Simplicity and Flexibility:** Flask gives developers the freedom to structure their applications the way they want. It doesn't impose a specific project layout or require the use of specific libraries or tools.
2. **Extensibility:** There is a wide range of extensions available for Flask that can add functionality such as database integration, form handling, authentication, and more.
3. **Built-in Development Server:** Flask includes a built-in development server and debugger, which makes it easier to test and debug applications during development.

## **Getting Started with Flask**

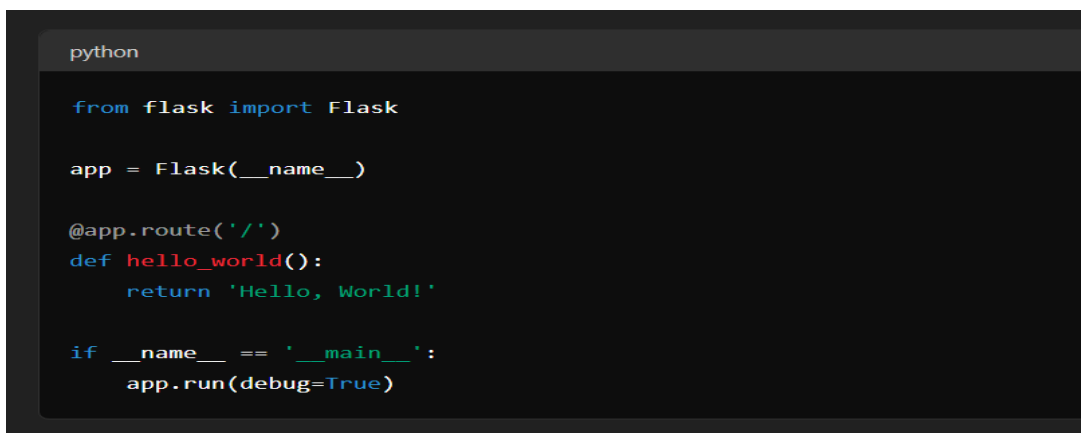
### **1. Installation**



```
bash
pip install Flask
```

Figure (5.3) Installation Flask Library

### **2. Creating a Simple Flask Application**



```
python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(debug=True)
```

Figure (5.4) Create Flask App

This will start a development server at <http://127.0.0.1:5000/>. When you visit this URL in your web browser , you will see "Hello, World!".

## **5.4: APIs**

An API (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate with each other. APIs enable the interaction between various components of software, making it possible to access data and functionality from other applications, services, or platforms.

### **Key Concepts of APIs**

- **Endpoints**

These are specific URLs defined in the API that allow clients to interact with the server. Each endpoint typically corresponds to a specific resource or functionality.

- **Requests and Responses**

Clients send requests to the API, and the API returns responses. Requests and responses usually involve HTTP methods such as GET, POST, PUT, DELETE, etc.

- **Resources**

These are the data objects or services that an API provides. For example, in a RESTful API, resources could be users, orders, or products.

- **Methods**

APIs use various methods to interact with resources. Common HTTP methods include:

- **GET:** Retrieve data from the server.
- **POST:** Send new data to the server.
- **PUT:** Update existing data on the server.
- **DELETE:** Remove data from the server.

## **Advantages of APIs**

- **Modularity**

APIs enable the separation of concerns by allowing different components of an application to interact without being tightly coupled.

- **Scalability**

APIs make it easier to scale different parts of an application independently.

- **Integration**

APIs facilitate the integration of different services and applications, making it possible to build complex systems.

- **Reusability**

APIs allow developers to reuse functionality across different applications, reducing development time and effort.

- **Interoperability**

APIs enable different systems and platforms to work together, even if they are built using different technologies.

- **Enhanced Security**

APIs can enforce authentication and authorization protocols, ensuring that only authorized users and applications can access sensitive data or functionalities.

## **5.5: Main Parts of Backend Code**

1. Imports
2. Flask Application Initialization
3. Model Loading and Class Definitions
4. Image Preprocessing Function
5. Prediction Route
6. Running the Application

## Imports

```
from flask import Flask, request, jsonify
import numpy as np
import tensorflow as tf
from PIL import Image
```

Figure (5.5) Packages used

### 1. flask

- **Flask:** It is used to create web applications, including APIs. Flask allows you to define routes, handle requests, and return responses.
- **request:** It is used to access data sent in HTTP requests, such as form data, JSON payloads, and query parameters.
- **jsonify:** It is used to convert Python dictionaries into JSON responses, which are then sent back to the client.

### 2. numpy (imported as np)

It is used for working with arrays and matrices, performing mathematical operations, and handling large datasets efficiently.

### 3. tensorflow (imported as tf)

It is used to build and train machine learning models, including deep learning models for tasks like image recognition, natural language processing, and more.

### 4. PIL (Python Imaging Library)

It is used for opening, manipulating, and saving image files.

This can include tasks like resizing images, converting between formats, and preprocessing images for machine learning models.

## Flask Application Initialization

```
app = Flask(__name__)
```



## Model Loading and Class Definitions

The pre-trained TensorFlow model is loaded from the 'model.h5' file without recompiling it and a list of classes Figure (5.6) that the model can predict is defined.

```
model = tf.keras.models.load_model('model.h5', compile=False)

classes = ['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy',
           'Blueberry___healthy', 'Cherry_(including_sour)___Powdery_mildew', 'Cherry_(including_sour)___healthy',
           'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Corn_(maize)___Common_rust_',
           'Corn_(maize)___Northern_Leaf_Blight', 'Corn_(maize)___healthy', 'Grape___Black_rot',
           'Grape___Esca_(Black_Measles)', 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy',
           'Orange___Huanglongbing_(Citrus_greening)', 'Peach___Bacterial_spot', 'Peach___healthy',
           'Pepper_bell___Bacterial_spot', 'Pepper_bell___healthy', 'Potato___Early_blight', 'Potato___Late_blight',
           'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy', 'Squash___Powdery_mildew',
           'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___Bacterial_spot', 'Tomato___Early_blight',
           'Tomato___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot',
           'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot',
           'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']
```

Figure (5.6) List of classes

## Image Preprocessing Function

```
def prepare_image(image, target):
    if image.mode != "RGB":
        image = image.convert("RGB")

    image = image.resize(target)
    image = np.array(image) / 255.0
    image = np.expand_dims(image, axis=0)

    return image
```

Figure (5.7) Preprocessing Function

## Prediction Route

A Flask route “/predict\_image” is defined to handle POST requests. It processes the uploaded image, makes a prediction using the model, and returns the predicted class and confidence as a JSON response.

```

@app.route('/predict_image', methods=['POST'])
def predict():
    data = {"success": False}

    if request.method == "POST":
        if "image" in request.files:
            try:
                image_file = request.files["image"]
                image = Image.open(image_file)

                if image:
                    processed_image = prepare_image(image, target=(256, 256))

                    preds = model.predict(processed_image)
                    predicted_class_index = np.argmax(preds)
                    predicted_class = classes[predicted_class_index]
                    confidence = preds[0][predicted_class_index]

                    data = {"label": predicted_class, "probability": round(float(confidence), 2)}

            except Exception as e:
                data["error"] = str(e)

    return jsonify(data)

```

Figure (5.8) Endpoint for Prediction

## Running the Application

The Flask application is started when the script Figure (5.9) is run directly

```

if __name__ == "__main__":
    app.run()

```

Figure (5.9) run the application

## 5.6: Testing Api Tools

1. **Postman:** Is a versatile API testing tool that allows developers to design, test, and document APIs quickly and efficiently. It provides a user-friendly interface for making various types of HTTP requests and inspecting responses.

2. **SoapUI**: Is a comprehensive API testing tool specifically designed for testing SOAP and REST web services. It provides a graphical interface for creating and executing functional, security, and load tests on APIs.
3. **cURL**: Is a command-line tool for transferring data with URLs. It supports a wide range of protocols (HTTP, HTTPS, FTP, FTPS, SCP, SFTP, etc.) and is widely used for testing APIs and web services from the command line.

## Postman

### Features

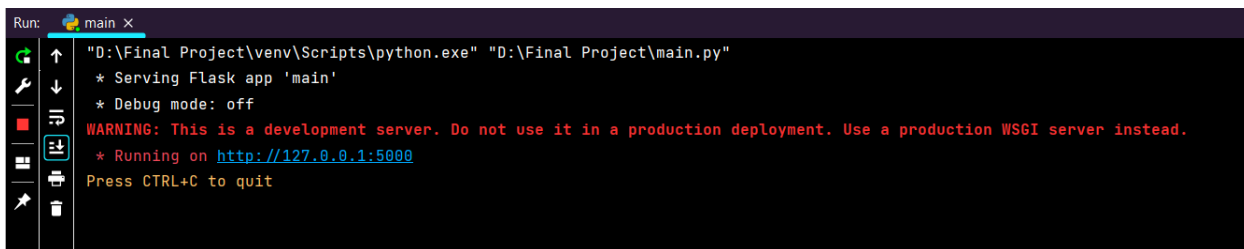
- Create and save requests to various endpoints.
- Add query parameters, headers, and body content.
- Automate tests using Postman scripts.
- Organize requests into collections for easy management.

### Usage

- Create a new request in Postman.
- Specify the HTTP method (GET, POST, PUT, DELETE, etc.).
- Enter the endpoint URL and any required parameters.
- Send the request and inspect the response.

## Postman Testing Steps

1. Run the code Figure (5.10).



```
Run: main x
"D:\Final Project\venv\Scripts\python.exe" "D:\Final Project\main.py"
* Serving Flask app 'main'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Figure (5.10) run the Flask code

2. Open postman application.



Figure (5.11) POSTMAN icon

3. Choose the method type , write the URL , write “ image “ in key filed then choose your image in value filed and press send.

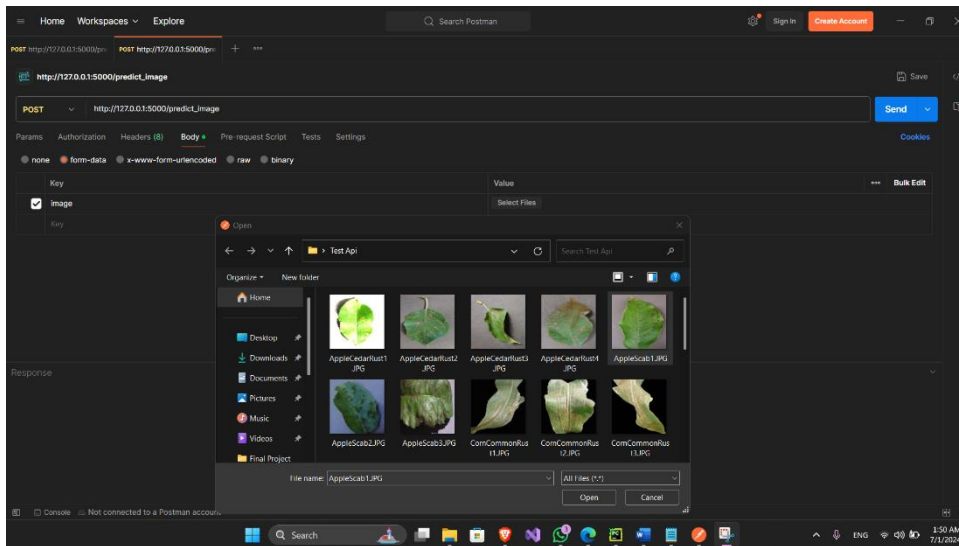


Figure (5.12) Testing with Postman

4. Get the output

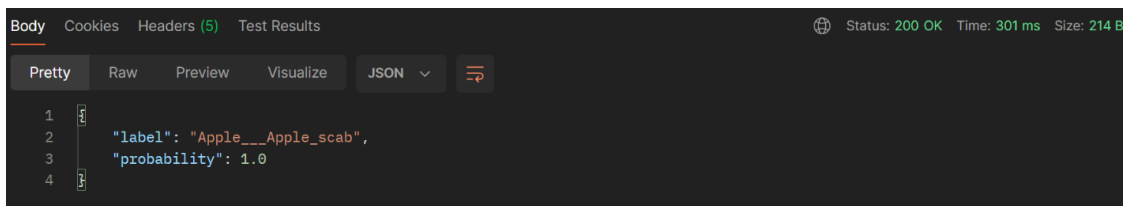


Figure (5.13) Output of “predict\_image” endpoint

## **6.7: Hosting**

refers to the service of providing storage space and access to websites, applications, or data on a server, which can be accessed over the internet. The hosting provider offers the necessary technologies and infrastructure to ensure that the content or application is available online and can be accessed by users.

## **Key Components of Hosting**

1. **Server:** A powerful computer that stores data and serves it to users over the internet.
2. **Storage:** Space on the server where website files, databases, and applications are stored.
3. **Bandwidth:** The amount of data that can be transmitted between the server and users over a period of time.
4. **Domain Name:** The address users enter in their web browser to access the hosted content (e.g., [www.example.com](http://www.example.com)).
5. **IP Address:** A unique numerical label assigned to the server for identification and location.
6. **Uptime:** The amount of time the server is operational and accessible to users.

## **Purpose of Hosting**

- **Website Hosting:** Making websites accessible to users over the internet.
- **Application Hosting:** Running web applications that can be accessed and used online.
- **Data Hosting:** Storing and managing data that needs to be accessed remotely.

## **Hosting Types**

Here are some common types of hosting Local hosting and online hosting serve different needs and have distinct advantages and disadvantages. Here's a comparison

### **local hosting**

refers to the practice of hosting websites, applications, or services on servers that are physically located within an organization's premises or within a private network. In this setup, the hardware and software resources needed to host and run the applications are maintained and managed internally by the organization's IT team.

## **Advantages**

1. **Control:** You have full control over the server environment, hardware, and software configurations.
2. **Security:** Physical access to the server can be tightly controlled.
3. **Latency:** Faster access times if the server is on a local network.

## **Disadvantages**

1. **Maintenance:** Requires in-house expertise for setup, maintenance, updates, and troubleshooting.
2. **Cost:** Higher initial costs for hardware and ongoing costs for power, cooling, and maintenance.
3. **Scalability:** Limited by local hardware capacity; upgrading can be costly and complex.
4. **Accessibility:** Remote access can be more complex to set up and manage.

## **Online Hosting**

Ideal for public-facing websites, scalable applications, and services that require high availability and remote accessibility.

## **Advantages**

1. **Accessibility:** Easily accessible from anywhere with an internet connection.
2. **Scalability:** Easy to scale resources up or down based on demand.
3. **Maintenance:** The hosting provider handles server maintenance, updates, and troubleshooting.
4. **Cost-Effective:** Lower initial costs and a variety of pricing plans to fit different needs.
5. **Reliability:** Hosting providers often offer high uptime guarantees and redundant systems.

## **Disadvantages**

1. **Control:** Less control over the server environment and configurations.
2. **Security:** Potential vulnerabilities if the hosting provider's security measures are not robust.
3. **Latency:** Higher latency compared to local hosting, especially if the server is geographically distant.

## **Decision Factors**

- **Budget:** Online hosting can be more cost-effective for smaller projects.
- **Expertise:** Local hosting requires more technical know-how.
- **Performance Needs:** Consider latency and scalability requirements.
- **Security and Control:** Determine the level of control and security necessary for your application.

## **5.8: Connection with Mobile**

This guide outlines the steps to set up a locally hosted environment for running a deep learning model accessible by a mobile team. We'll cover downloading the model, integrating it with the code, and running the application using Flask.

## **Steps**

1. Download Deep Learning Model
2. Integrate Model with Code
3. Setup Flask Application
4. Run the Flask Application
5. Ready for Mobile Team

**GitHub Link:** <https://github.com/aymanragab8/Plant-Disease-Prediction>

# **Chapter 6**

## **Mobile Application**

### **6.1: Introduction**

The advent of smartphones has revolutionized various aspects of our lives, including agriculture. This chapter discusses the development of a mobile application written in Flutter designed to detect plant diseases through images.

The application aims to assist farmers and gardeners in maintaining plant health by providing accurate and timely diagnoses.

### **Objectives of the Application:**

To provide an easy-to-use interface for detecting plant diseases.

To leverage machine learning for accurate disease detection.

To offer actionable advice based on the diagnosis.

### **6.2: What is Flutter?**

Introduction to Flutter:

Flutter is an open-source UI software development toolkit created by Google. It is used to develop applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase.

### **Why Use Flutter for Mobile Development?**

Cross-Platform Development: Write once, run anywhere.

High Performance: Fast development with hot reload and native performance.

Rich Widgets: A wide range of customizable widgets for a native look and feel.



Strong Community: Extensive documentation and community support.

## **Benefits of Flutter:**

Fast Development: Hot reload allows for quick iteration.

Expressive and Flexible UI: Extensive library of widgets.

Native Performance: Compiles to native ARM code.

## **6.3: Object-Oriented Programming (OOP) in Flutter**

Overview of OOP Concepts:

Classes and Objects: Basic building blocks of OOP.

Inheritance: Mechanism to create new classes from existing ones.

Polymorphism: Ability to present the same interface for different data types.

Encapsulation: Bundling data with methods that operate on the data.

Examples from the Application Code:

```
class PlantDisease {  
  final String name;  
  final String description;  
  
  PlantDisease(this.name, this.description);  
}  
  
class PlantApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: HomePage(),  
    );  
  }  
}
```

## **6.4: Application Structure**

- **Main Components of the Application:**

**Home Page:** The main interface with options to capture or select an image.

**Diagnosis Page:** Displays the results of the image analysis.

**Disease Info page:** Provides detailed information about detected diseases.

- **Navigating Through the Application:**

The app uses Flutter's navigation system to switch between pages.

Navigation is managed through the Navigator widget, which allows for a smooth user experience.

## Home Page

## Welcome Page Features:

**Introduction:** Brief overview of the app's functionality.

**Instructions:** How to use the app effectively.

### Camera Icon Functionality:

**Capture Image:** Opens the camera to take a real-time photo.

```
Future takePhoto() async {  
    final returnTakePhoto =  
        await ImagePicker().pickImage(source: ImageSource.camera);  
    setState(() {  
        _selectionTakeImage = File(returnTakePhoto!.path);  
        print('>>>>>>>>>>>>>>>${returnTakePhoto.path}');  
        print('>>>>>>>>>>>>>>>$_selectionTakeImage!.path');  
    });  
}
```

**Upload Image:** Allows users to upload an existing photo from the gallery.

```
Future select_Image() async {  
    final returnImage =  
        await ImagePicker().pickImage(source: ImageSource.gallery);  
  
    setState(() {  
        _selectionImage = File(returnImage!.path);  
        print('>>>>>>>>>>>>>>>${returnImage.path}');  
        print('>>>>>>>>>>>>>>>${_selectionImage!.path}');  
    });  
}
```

## **Plant Diseases**

### **List of Plant Diseases Detected by the App:**

- ❖ Powdery Mildew
- ❖ Rust
- ❖ Blight
- ❖ Mosaic Virus
- ❖ Leaf Spot

### **Detailed Descriptions of Each Disease:**

Powdery Mildew: White powdery spots on leaves and stems.

Rust: Orange or yellow spots on the undersides of leaves.

Blight: Rapid yellowing and browning of leaves.

Mosaic Virus: Mottled appearance on leaves.

Leaf Spot: Small brown or black spots on leaves.

## **6.5: Image Selection**

### **Options for Image Selection:**

**Camera:** Users can take a real-time picture of the plant.

**Gallery:** Users can choose an existing photo from their device's gallery.

### **How to Capture and Upload Images:**

The app uses the `image_picker` package to handle image selection and uploading.

1. The `image_picker` package is used to capture images from the camera or select images from the gallery.
2. After capturing or selecting an image, the image file is converted to a byte array.

3. A multipart HTTP request is created using the http package, and the image byte array is added to this request.
4. The request is sent to the server, and the server's response is handled to determine if the upload was successful.

**By following these steps, the app:**

effectively handles image selection and uploading.

allowing users to capture or select image .

send them to a server for further processing.

This functionality is critical for the plant disease detection feature, as it enables the app to receive and analyze images of plants to diagnose potential diseases.

## **6.6: Image Upload Code**

Explanation of the Code for Uploading Images:

The following code snippet demonstrates how to use the image\_picker package to upload an image:

```
import 'package:http/http.dart' as http;
import 'package:http_parser/http_parser.dart';
import 'package:flutter/services.dart' show rootBundle;
import 'dart:typed_data';

Future<void> uploadFile() async {
  var url = 'http://127.0.0.1:5000/predict_image';
  var request = http.MultipartRequest('POST', Uri.parse(url));

  ByteData assetByteData = await rootBundle.load('assets/images/testImg.jpg');
  List<int> assetBytes = assetByteData.buffer.asUint8List();
  var multipartFile = http.MultipartFile.fromBytes(
    'image',
    assetBytes,
    filename: 'testImg.jpg',
    contentType: MediaType('image', 'jpg'),
  );
  request.files.add(multipartFile);
  try {
    var response = await http.Response.fromStream(await request.send());
    if (response.statusCode == 200) {
```

```

    print('File uploaded successfully');
    print('Response: ${response.body}');
  } else {
    print('Failed to upload file. Error: ${response.statusCode}');
  }
} catch (e) {
  print('Exception during file upload: $e');
} }

```

## Handling Image Data in Flutter:

Once an image is selected, it is processed and sent to a backend server for analysis using an HTTP request.

### 6.7: Diagnosis Response

#### Displaying the Diagnosis:

The diagnosis is displayed on the Diagnosis Page with the following details:

**Label:** The name of the detected disease.

**Probability:** The confidence level of the diagnosis.

**Label:** Disease Name:

The disease name is displayed prominently to inform the user.

**Probability:** Confidence Level of Diagnosis:

A percentage value indicating the confidence of the diagnosis, helping users to know the reliability of the results.

```

import 'package:equatable/equatable.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
part 'responses.g.dart';

@JsonSerializable()
class Response extends Equatable {
  final String? label;
  final String? probability;
  Response({this.label, this.probability});
}

```

```
factory Response.fromJson(Map<String, dynamic> json) =>
    _$ResponseFromJson(json);
@override
    TODO: implement props
    List<Object?> get props => [label , probability]; }
```

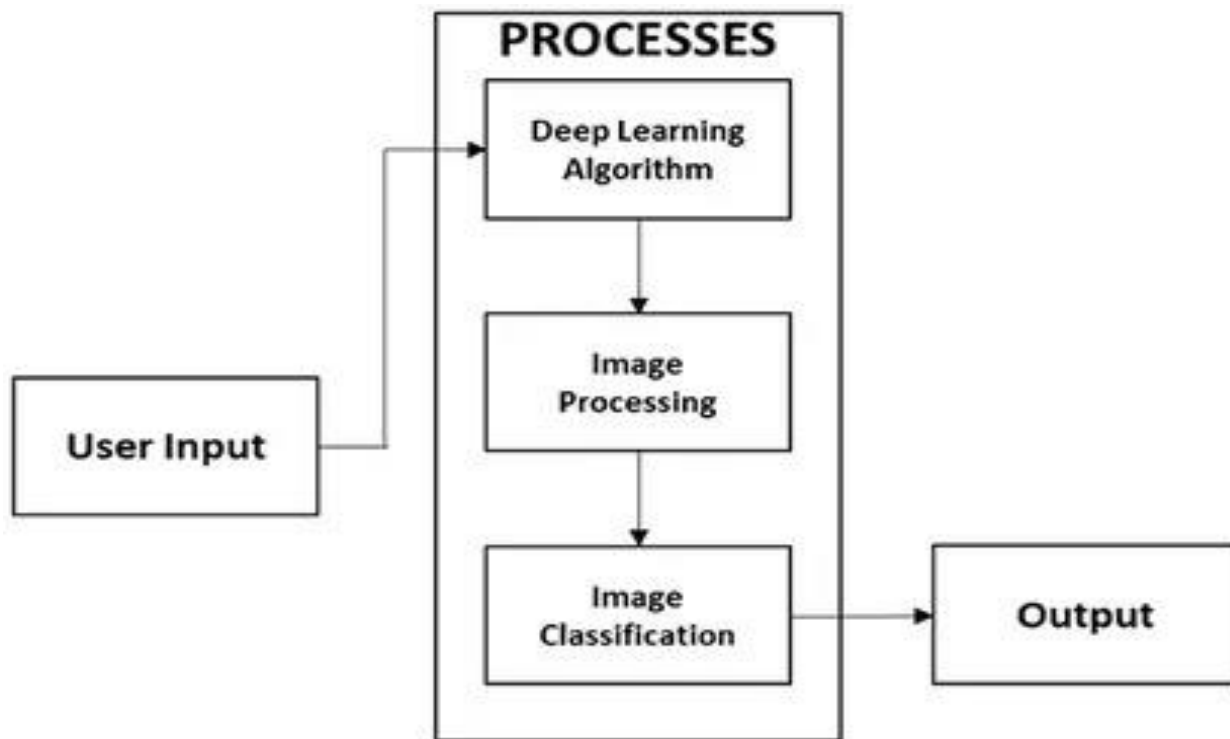
## **6.8 Steps to Use the Application**

### **Step-by-Step Guide to Using the App:**

Press the Camera Icon: Opens the camera to take a photo.

Take a Picture of the Plant: Capture the image for analysis.

Receive a Diagnosis and Treatment Plan: The app processes the image and returns a diagnosis along with treatment suggestions.



## **Challenges and Solutions**

### **Common Issues Faced During Development:**

**Image Processing:** Ensuring accurate and fast image analysis.

**User Interface:** Creating an intuitive and user-friendly interface.

**Cross-Platform Compatibility:** Ensuring the app works seamlessly on both Android and iOS.

### **Solutions and Workarounds Implemented:**

**Optimized Algorithms:** Improved the accuracy and speed of image processing algorithms.

**User Feedback:** Iterated on UI design based on user feedback.

**Testing:** Rigorous testing on multiple devices to ensure compatibility.

### **Testing and Evaluation**

#### **Testing Methodologies Used:**

**Unit Testing:** Testing individual components for functionality.

**Integration Testing:** Ensuring components work together seamlessly.

**User Testing:** Gathering feedback from real users to improve the app.

#### **Evaluation of App Performance and Accuracy:**

**Performance Metrics:** Evaluating the speed and responsiveness of the app.

**Accuracy of Diagnosis:** Measuring the success rate of disease detection.

# References

## IOT

- 1- <https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://deviceauthority.com/unpacking-iot-architecture-layers-and-components-explained/&ved=2ahUKEwiz2uKunJWHAXtUqQEHduBB34QFnoECBIQBQ&usg=AOvVaw0pqehJHEzOjgqZ6BJKtDXq>
- 2- <https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.geeksforgeeks.org/architecture-of-internet-of-things-iot/&ved=2ahUKEwiz2uKunJWHAXtUqQEHduBB34QFnoECA0QAQ&usg=AOvVaw1QuWqFGUb57MKTUP60k2gP>
- 3- <https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://how2electronics.com/measure-soil-nutrient-using-arduino-soil-npk-sensor/&ved=2ahUKEwj2wtfRnJWHAXUTdqQEHVdpAIwQFnoECC0QAQ&usg=AOvVaw1wSEoRoGdKbJOIr5tYDxEN>
- 4- <https://lastminuteengineers.com/rain-sensor-arduino-tutorial/>
- 5- <https://lastminuteengineers.com/dht11-module-arduino-tutorial/>
- 6- <https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/>
- 7- <https://www.circuito.io/blog/arduino-uno-pinout/>
- 8- <https://www.renkeer.com/product/soil-npk-sensor/>

## Machine Learning

1. <https://www.geeksforgeeks.org/introduction-machine-learning/>
2. <https://www.javatpoint.com/types-of-machine-learning>
3. <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>
4. <https://www.geeksforgeeks.org/k-nearest-neighbours/>



5. <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>

## **Deep Learning**

1. <https://www.javatpoint.com/deep-learning-algorithms>
2. <https://www.ibm.com/topics/convolutional-neural-networks>
3. <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>
4. <https://towardsdatascience.com/components-of-convolutional-neural-networks-6ff66296b456>
5. <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>

## **Backend**

1. <https://www.spaceotechnologies.com/glossary/tech-terms/what-is-back-end-development/>
2. <https://www.freecodecamp.org/news/python-back-end-development-the-beginners-guide/>
3. <https://pypi.org/project/Flask/>
4. <https://ibm.com/topics/api>
5. <https://www.codecademy.com/article/back-end-architecture>

## **Flutter**

- 1- <https://www.geeksforgeeks.org/flutter-tutorial/>
- 2- <https://flutter.dev/learn>
- 3- <https://www.geeksforgeeks.org/dart-tutorial/>
- 4- [https://dev.to/moyeen\\_haider/object-oriented-programming-oop-in-flutter-cie](https://dev.to/moyeen_haider/object-oriented-programming-oop-in-flutter-cie)