FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Master Thesis

Automatic watering of plants in a pot using plant recognition with CNN

Vadlamani Raghu Ram

(Matriculation Number- 1230934)

Under the Guidance of

MSc. Virginia Kramer and Prof. Dr. Karsten Schmidt

Dept of Computer Science and Engineering

Frankfurt University of Applied Sciences

Nibelungenplatz 1, 60318, Frankfurt am Main, Germany

Contents

List of Figures:

List of Tables

ACKNOWLEDGMENTS

I am grateful to the **Management of the Frankfurt University of Applied Sciences** for providing me a congenial atmosphere and encouragement.

I am heartily thankful to my supervisor **Dr. Karsten Schmidt and MSc. Virginia Kramer** who has given me a lot of support to carry out the project work.

Lastly, my regards to all those who supported me directly or indirectly during the completion of the thesis under title "**Automatic watering of plants in pot using plant recognition with CNN**".

List of abbreviations

ANN	Artificial Neural Network
AI	Artificial Intelligence
ADC	Analog to digital converter
ADDR	Address
APWS	Automatic plant watering system
CNN	Convolutional Neural Network
CPU	Central processing unit
Conv	Convolution
FPS	Frames Per Second
GPU	Graphics Processing Unit
GPIO	General purpose input and output pins
GSM	Global system for mobile communications
GND	Ground
HDMI	High-definition multimedia interface
I2C	Inter-Integrated Circuit
OS	Operating System
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
PGA	Programable gate array
RAM	Random access memory
ReLU	Rectified Linear Unit
R-CNN	Region based Convolutional Neural Network
SCL	Serial clock
SDA	Serial data
SSD	Single Shot Detection
SWP	Smart watering plants
T_{NP}	Threshold value of Normal plant
T_{DP}	Threshold value of Desert plant
VCC	Voltage common collector
VDD	Voltage Drain Drain

ABSTRACT

The plant's growth is dependent on the soil moisture, but the required soil moisture threshold level is not the same for all the plant species. In the modern era with the combination of advanced Embedded and Computer Vision technologies, it is possible to detect the plant species on an Embedded platform and watering automatically. In this work, we have discussed the implementation and working model which can detect the plant in a pot and water the plant automatically based on the threshold of the soil moisture and plant species. The complete model is built on Jetson Nano Nvidia GPU and a PCB which is designed according to the requirements, to control the motor pump and read sensor values. The API is implemented in a more customized way. The plant species and threshold values can be customized based on the climatic condition of any country. As the model is implemented for automatic watering of the plant in a pot, the calibration of the pot width is also done using the MobileNet SSD V2 algorithm to suggest the required soil moisture content and the water level in the tank.

Keywords—MobileNet SSD v2, Embedded Computer vision, Automatic watering model, PCB design, Jetson Nano GPU, Mars rover

1. INTRODUCTION

1.1 Problem Statement

The growth of the plants depends on the water, nutrients, and soil type. Nutrients can be absorbed from the soil through the roots of the plants, but the roots can observe the nutrients from the soil only when then the soil is having enough moisture content. The moisture content of the soil is maintained through groundwater, rains. The big trees can survive with rainwater and groundwater due to their root structure, but the small plants are not capable of observing the groundwater. So, watering is a must for small plants. Nowadays plants are growing in pots which separates the plant from the earth. So, planting in pots affects the growth of plants because plants cannot observe the groundwater. External watering of plants is compulsory mainly for plants growing in pots. How to maintain the Soil moisture automatically?

The watering of plants also depends on the type of plants. Over or less soil moisture content becomes the root cause of plant growth. Plants such as Convolvulaceae and Umbelliferae require limited soil moisture; more soil moisture can damage the starchy and fleshy tuber. Another group of plants that grow in the Desert like Aloe barbadensis miller and Cactaceae does not require more soil moisture content. Whereas the flower plants like Rose and Sunflower require more amount of soil moisture. So, the maintenance of the soil moisture content also depended on the plant species or plant family. How to identify the plant family name or flower name? What are the inputs required to identify the plant family name or flower name?

Due to the busy schedule of humans in the 21st century sometimes humans forget to water the plants. Plants are to be taken care of during the vacation periods as well. Irregular watering of plants also affects plant growth. When watering of plants should be done? This paper addresses the solution to this problem by using and integrating the techniques of machine learning, computer vision, and embedded systems.

1.2 General Idea of Model and Block Diagram:

As per the problem statement, the general idea is to implement a model which can detect the plant species or plant type and calculate the soil moisture and do the watering based on the threshold values. To implement the model, we need a microcontroller that can run the object detection algorithm and can communicate with the soil moisture sensor and start watering to the plant. Figure 1 shows the block diagram of the general model. The camera module is used to take the frames of the real world and send them to the microcontroller. The microcontroller runs the plant detection algorithm on each frame sent by the camera module. The plant detection algorithm detects the plant name by using the features like flowers, leaves, steam structure. After detecting the plant name the microcontroller collects the information of the soil moisture sensor. If the soil moisture of the detected plant is less than the threshold level then the watering to the plant starts else watering of the plant does not start. The watering unit consists of a motor pump and motor driver to control it. The microcontroller communicates with the watering unit to start and stop watering.

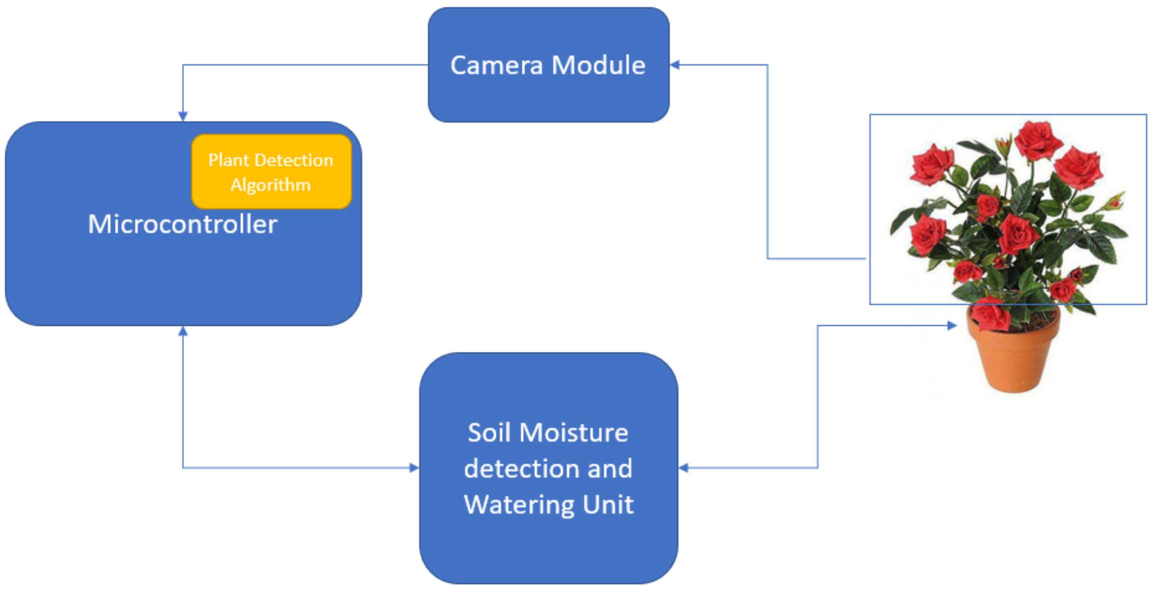

Camera Module
Plant Detection Algorithm
Microcontroller
Soil Moisture detection and Watering Unit

Figure 1 Block diagram of general Idea of model

1.3 Thesis Organization

The basic outline and overview of the thesis are discussed in this section.

Chapter Two

This chapter provides a review of existing literature about the MobileNet SSD V2 algorithm and why this algorithm is selected to run on the Embedded platform. Also, discuss why Jetson nano is chosen as an Embedded platform to run the object detection algorithm and integrate different sensors and actuators. This section also discusses the existing models for the automatic watering system developed on different embedded platforms and their advantages.

Chapter Three

Chapter Two has introduced the fundamentals of Artificial Intelligence (AI). The Step-by-step procedure and phases of Artificial Intelligence algorithm. The necessity of each phase is also discussed. After discussing the phases of the general AI algorithm, we drive to the Convolutional Neural Networks (CNN). The Convolutional Neural Networks (CNN) are the basic blocks of AI image processing technologies. All the high-level algorithms of AI image processing are built on the Convolutional Neural Networks (CNN). So, to understand the MobileNet SSD V2 algorithm which is used to train the images for the model implementation in chapter < > it is important to know the basic operations of Convolutional neural networks which are discussed in this chapter. The Convolutional Layer, rectified linear unit (ReLU) popularly known as ReLU layer, Pooling Layer, Flattening Layer and Fully connected Layer operations are also discussed.

Chapter Four

This chapter provides information on Hardware components used for the model implementation. The pin structure of the Jetson Nano Nvidia GPU and necessary details are discussed. The working principle of different sensors used like Capacitive Soil Moisture sensor and Water Level Sensor and the working of IC ADS1115 which we use on the PCB to convert the values of Water Level sensor. Also discussed the Add PCB details.

Chapter Five

This chapter provides information on custom Add on PCB design in the Kicad tool. The workflow of the Kicad tool, the design of the schematic diagram for the Add on PCB is discussed. The PCB wiring and structure of the PCB placement of components and remaining implementation notes are discussed.

Chapter Six

This chapter provides the information of the Object detection algorithm i.e, MobileNet SSD v2 chosen for the project. The implementation of depthwise separable convolutions and inverted residuals are discussed as they are the building block of MobileNet SSD v2. The architecture of the MobileNet SSD v2 and the trade-off parameter are discussed. How the SSDLite is implemented on the top of MobileNet v2 and the impact of it on the parameters to train and reduction of computational power because of it is discussed and compared with some real-time object detection algorithms like SSD 300, SSD 512, YOLO v2, and MobilNet SSD v1.

Chapter Seven

This chapter discusses how the (Automatic Watering Model) AWM API is implemented and also discusses important function implementation and flowcharts for some functions.

Chapter Eight

This chapter discusses the working model implementation and the flowchart of the working model. Also provides information on testing/validation of the implemented model. The testing and validation of the model are done at the unit level and also the integration testing of all units. The testing/validation steps are tabulated, and test evidence are provided in this chapter.

Chapter Nine

This chapter gives the information of experimental setup and results of the complete model in different conditions and finally concludes the requirement specifications are achieved.

2. LITERATURE REVIEW

1. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [1]

In this research paper, researchers have introduced MobileNets, a new model architecture that is based on depth-wise separable convolutions [4]. Researchers make the concentrate on the key design decisions that led to a successful model and show how to use a width multiplier and a resolution multiplier to create smaller and quicker MobileNets by trading off a respectable amount of accuracy for size and latency reduction. Authors have shown the comparison of MobileNets to other popular models in terms of size, speed, and accuracy to find how MobileNets performs in comparison to other models. By the comparison, it is clearly shown the effectiveness of MobileNets in a range of situations.

2. MobileNetV2: Inverted Residuals and Linear Bottlenecks [2]

In this work, authors have introduced MobileNetV2, a novel mobile architecture that enhances the performance of mobile models on a variety of tasks and benchmarks, as well as across a range of model sizes. A new framework is referred by authors as an SSDLite and also explains efficient techniques of using these mobile models to object identification. Researchers also show how to use a condensed version of DeepLabv3 called Mobile DeepLabv3 to create mobile semantic segmentation models.

DeepLabv3 is built on an inverted residual structure with shortcut connections between narrow bottleneck layers. As a source of non-linearity, the intermediate expansion layer filters features with lightweight depthwise convolutions. To retain representational power, authors have discovered that non-linearities in the thin layers must be removed. Researchers have shown how this increases performance and discuss the inspiration for this design.

Finally, the proposed method decouples the input/output domains from the transformation's expressiveness, providing a useful framework for further investigation. ImageNet classification, COCO object detection, and VOC picture segmentation are used to evaluate our performance. The trade-offs between accuracy and the number of operations evaluated by multiply-adds (MAdd), as well as actual latency and the number of parameters, are evaluated.

3. PERFORMANCE COMPARISON OF DEEP NEURAL NETWORKS ON IMAGE DATASETS [3]

The authors have made a detailed comparison of four deep learning models MobileNet, VGG16, ResNet, and Inception. Where one conclusion is made that MobileNet is not as accurate as VGG16, but it outperforms VGG16 in terms of resource usage versus accuracy. It may be utilized for a variety of purposes and by the entire community in areas where resources are limited. It takes less time, which is advantageous for some applications, such as traffic recognition, when precision is not necessary, but time is important.

4. Benchmark Analysis of Jetson TX2, Jetson Nano, and Raspberry PI using Deep-CNN [4]

Deep learning applications require hardware with minimal power consumption, excellent accuracy, and high performance. In high-performance deep learning applications, high-level graphics processing units (GPU) are widely used. Building a high-performance platform, on the other hand, costs a lot of money and consumes a lot of electricity. The performance of single-board processors in the NVIDIA Jetson Nano, NVIDIA Jetson TX2, and Raspberry PI is compared in this study using a CNN algorithm constructed with a dataset of fashion products photos. The case study is performed with the intention to achieve high accuracy with minimal hardware requirements in deep learning applications. The DeepFashion2 dataset was chosen for the model's training and testing. For detailing and Raspberry PI when the dataset size is 20K, this dataset is grouped in 5 separate sizes. The greatest memory use of the Jetson TX2 was 45K datasets. The power usage of both the CPU and GPU increases as the power consumption rises. In Jetson TX, power consumption has been found to be steadier as big data increases. In single-board computers, epoch=100 was found to be the best value for the models to get maximum accuracy. The studies revealed that the size of the dataset has a beneficial effect on accuracy in deep learning applications. With a 45k dataset on the Jetson TX2, the model has the greatest accuracy of 97.8%. In the 5K dataset performance tests, single-board computers achieved very close accuracy to one other. For five separate clusters, training and testing were done on each piece of hardware. Considering the results, it clearly says that, the Jetson TX2 uses more power it performs better in terms of speed, accuracy, and large data sets than other models. Based on the cost effectiveness and dataset (20K) performance test for the Jetson Nano is the best as compared with Raspberry PI

5. Automatic Plant Watering System [5]

Authors have introduced an automated plant watering system which is affordable and efficiently reduce the manual burden of farmers or guardian of the garden. Arduino Uno is used, and a water level sensor is used to measure the water level in the tank. Moisture level sensor measures moisture in the soil and based on the moisture value watering will be done to maintain moisture in the soil. If the water level is down using the GSM module user will be notified that water is low and fill the tank. GSM module is also used to notify the user that irrigation is completed. This prototype is systematically explained and implemented by the researchers.

6. Smart Watering of plants [6]

Intending to eliminate the requirement of manpower for crop irrigation researchers have implemented a model using Arduino and soil moisture sensor. The soil moisture sensor is used to refer present level of moisture in the soil and perform irrigation based on the threshold value of the soil. Arduino is used for the system controller to read soil sensor values and switch the water pump accordingly.

By literature review, we conclude that there are some models exist for doing watering automatically from Automatic Plant Watering System (APWS) [5] and Smart Watering of plants (SWP) [6]. The existing models start and stop the watering based on the threshold value of soil moisture content. The existing models cannot do the watering based on the plant type. The existing models can be modified by adding modules that can detect the plant type. The hardware used in the existing models can be changed to get accurate values. we can modify the model for the best outcome in terms of accuracy and cost-effectiveness. To improve accuracy to detect plants we are using MobileNet V2 which gives the best accuracy to detect plant features, and this is concluded from MobileNetV2: Inverted Residuals and Linear Bottlenecks [2]. To reduce hardware cost we are using Jetson nano which is cost-effective and works with high efficiency for our model that other alternatives (Raspberry PI and Jetson TX2).

3. Concept / State Of Research

Based on the problem statement to explain the state of research in better way plants are divided into types. They are Normal-type and Desert type. But the API is implemented based on the plant and threshold value of soil moisture which decreases the dependency on plant type. Normal type plants require more water, and the Desert type plants require less amount of water. The below Figure 2 explains the flow of ideas to solve the problem statement. Convolutional Neural Networks are used to detect the plant and plant type. The two types of plant datasets are collected and labelled. The labelled datasets are given to the SSD mobileNet_V2 algorithm for training. The trained model is used to detect the plant and plant type. If the detected plant is a normal type, then the soil moisture is calculated using the capacitive type of soil moisture sensor. If the soil moisture sensor reading is less than threshold value of normal plant (T_{NP}) (ADC value) then the watering of the plant will start and check the sensor readings again. If the soil moisture sensor reading value is greater than the threshold value of normal plant (T_{NP}) (ADC value) then it means the soil moisture content is enough and wait until further few hours. Similarly, if the detected plant is desert type, then the soil moisture is calculated. If the soil moisture sensor reading is less than threshold value of desert plant (T_{DP}) (ADC value) the watering of the plant will start and check the sensor reading again. If the soil moisture sensor reading is greater than threshold value of desert plant (T_{DP}) (ADC value) then it will wait until the next watering time. As the watering of the plant depends on the soil moisture value and type of plant, we have decided to set the threshold value as the input parameter. Instead of checking for desert plants or normal plants, giving the plant species and threshold value will be a good option to increase the flexibility of usage.

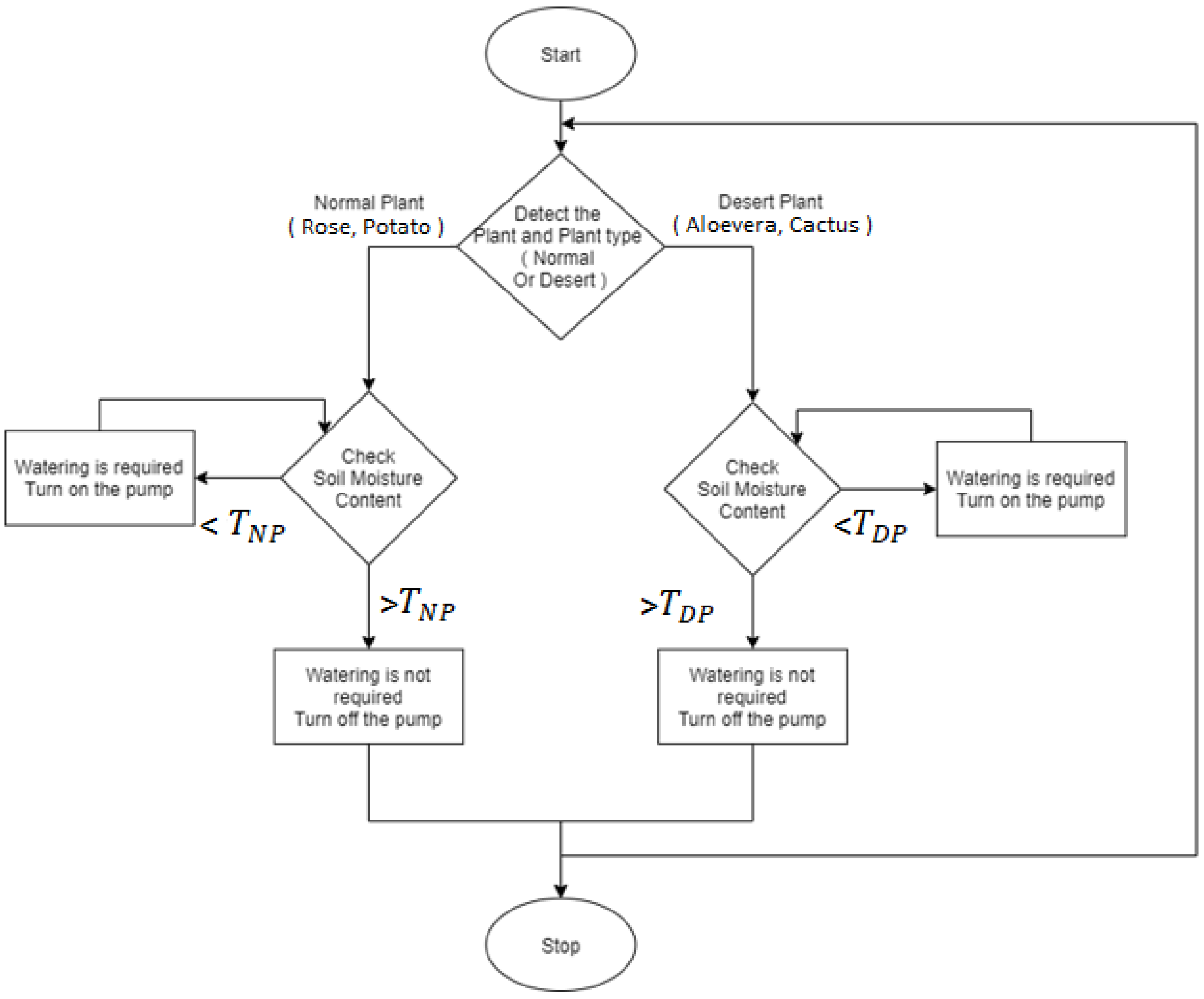

Start
Detect the Plant and Plant type (Normal Or Desert)
Normal Plant (Rose, Potato)
Desert Plant (Aloevera, Cactus)
Check Soil Moisture Content
Watering is required Turn on the pump
$< T_{NP}$
$>T_{NP}$
Watering is not required Turn off the pump
Check Soil Moisture Content
$<T_{DP}$
Watering is required Turn on the pump
$>T_{DP}$
Watering is not required Turn off the pump
Stop

Figure 2 FlowChart for State of research

4. THEORETICAL BACKGROUND

4.1 Fundamentals of Artificial Intelligent (AI):

Machine Learning is an integral part of Artificial Intelligent (AI) which enables the system to learn and adapt without explicit coding rather uses an algorithm and statistical models to analyse and conclude from the hidden pattern in the data [10].

There are three types of machine learning techniques: supervised learning, unsupervised learning, and reinforcement learning. Convolutional neural networks are part of the supervised learning paradigm. Object classification, object detection, and object segmentation are all accomplished with convolutional neural networks.

There are two stages to the machine learning process as shown in Figure 3. The training phase is the initial stage. The machine learning algorithm is fed training data during the training phase. The training data is used to train the machine learning algorithm. Images, sensor data, and other types of data make up the training data. The prediction phase is the second phase. The sensor's subsequent images or data are predicted using the trained model from the previous phase. The two phases and workflow of machine learning are depicted in the Figure 3.

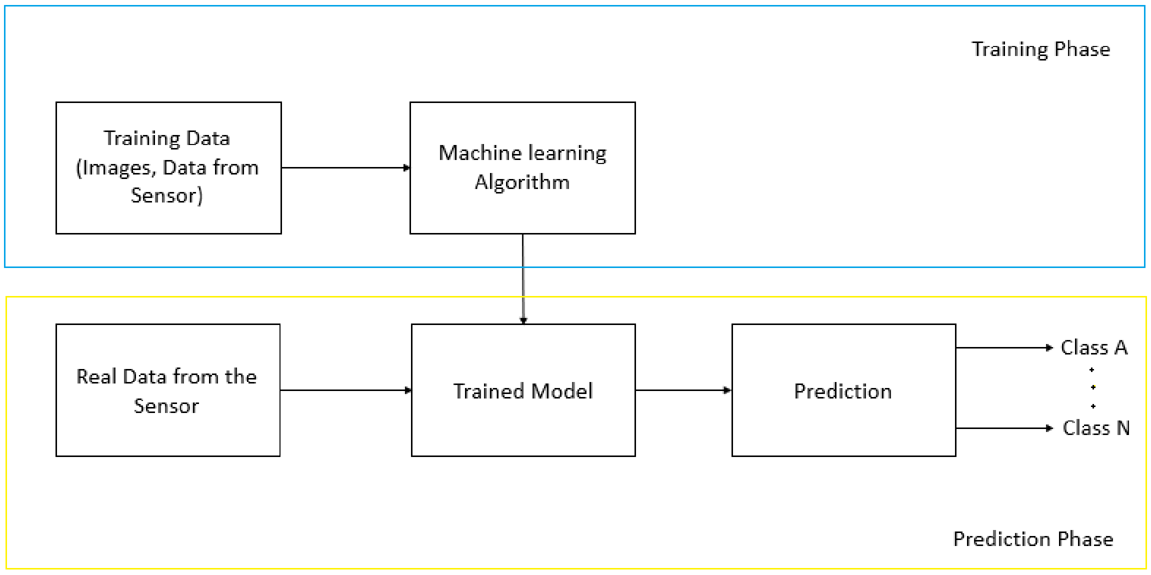

Training Phase
Training Data (Images, Data from Sensor)
Machine learning Algorithm
Real Data from the Sensor
Trained Model
Prediction
Class A
Class N
Prediction Phase

Figure 3 Phases of AI algorithm

4.2 Convolutional Neural Network (CNN)

The concept of a Convolutional neural network is used to solve the problems of computer vision. Computer vision is playing major in various fields like Robotics, Medical, etc. The convolutional neural networks use the information of each pixel of an image to formulate the features of the object in the image. To formulate the features of the object the convolutional neural networks use the discrete convolutions. The base components (or) layers of the convolutional neural networks are as follows as shown in Figure 4:

- Convolutional Layer
- ReLu Layer (Rectified Non-Linearity Layer)
- Pooling Layer
- Fully Connected Layer

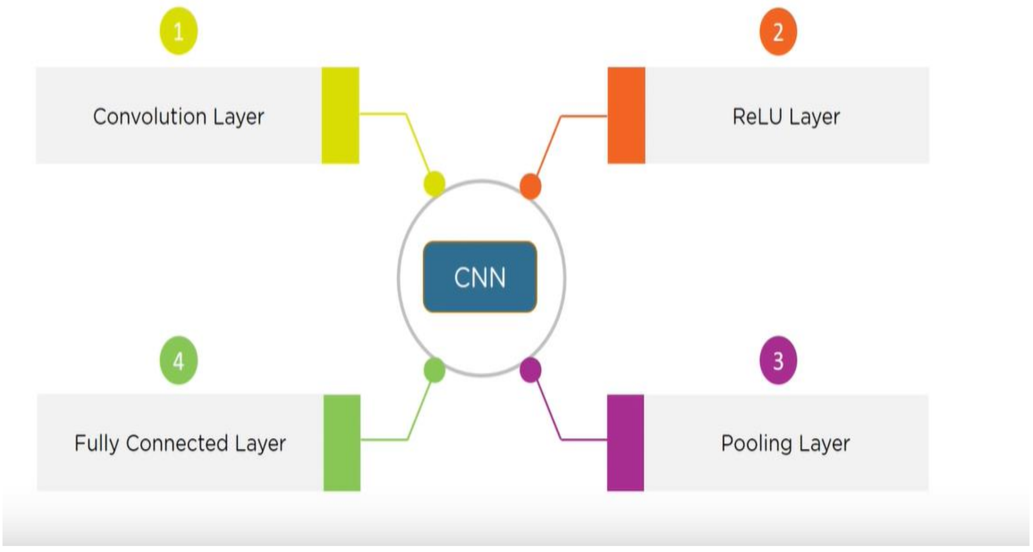


Figure 4 Layers of Convolutional Neural Network

4.2.1 Convolution Operation or Convolution Layer:

The convolution is an operation on two functions of real-valued arguments. Given an input x(t) and a kernel or weight w(a) the convolution operation is given in below equation 4.2.1.1.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \qquad \text{eq. 4.2.1.1}$$

The above equation 4.2.1.1 is for one dimensional data. The convolution is also applied on 2-dimensional or 3-dimensional or multidimensional data. The convolutional operation on 2-dimensional data is given in below equation 4.2.1.2

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n) \qquad \text{eq. 4.2.1.2}$$

Where

I(mn) is the input data.

K(i – m j - n) is the kernel.

S(i,j) is the output or feature map.

In the below example Figure 5 simple image pixels are taken. The example is explained by taking a binary pixel image of 5 X 5 size. The image pixels are multiplied with a convolution filter or kernel or weights. the convolution operation (dot product) is performed by sliding the filter on the image pixels. The convolved feature is extracted as shown in the pixel [17].

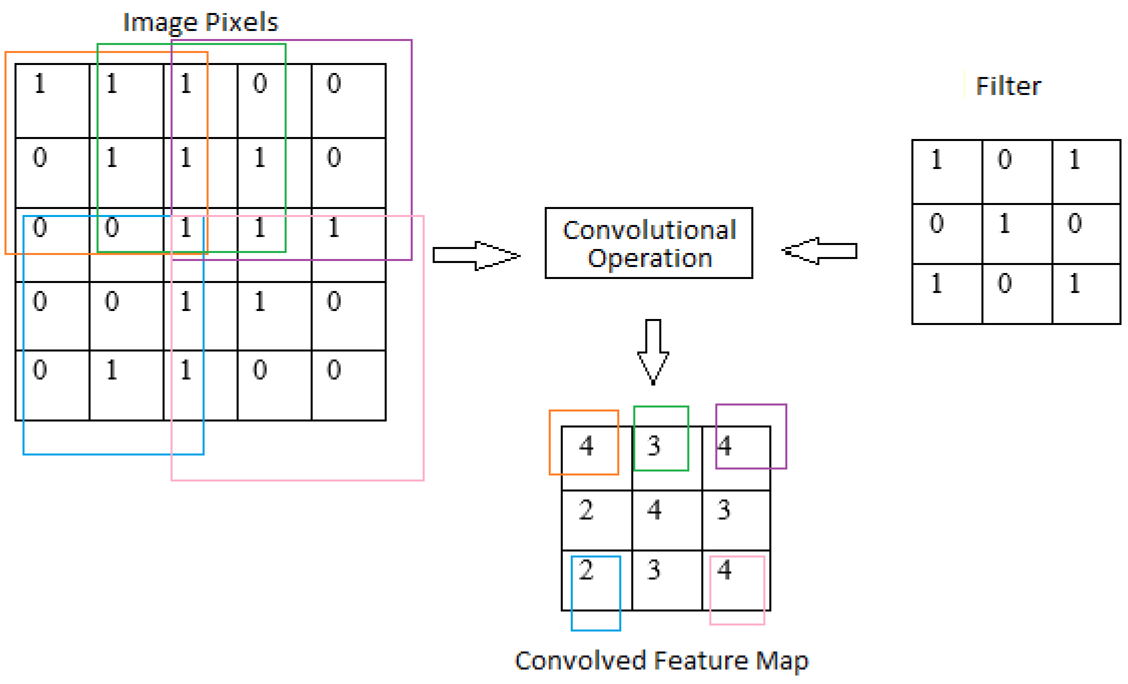


Figure 5 Convolutional Operation on 5X5 matrix with a filter size of 3X3

The convolution layer has several numbers of filters that perform convolution operations. The same operation is repeated with several filters.

4.2.2 Filters:

The convolution operation is performed with several filters to obtain different features of the image. Some of the filters we use in general are edge detection, sharpen, blur, Gaussian blur, and more. Some of the filters and the output feature maps are given in Figure 6 [18].

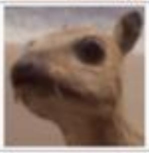
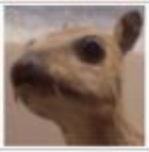
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 6 Commonly used filters in Convolutional operation [18]

4.2.3 Strides:

Defined as the number of pixels shifted over the input image to apply filter during convolution operation. The convolution operation is performed by shifting the filter with one pixel if the value of stride is selected as one. The convolution operation is performed by shifting the filter with two pixels if the value of stride is selected as two. the example for the stride value with two is explained in the below Fig. 4. In the below Figure 7, the input of size 7 × 7, a filter of size 3 × 3, and stride value of 2 are taken to perform the convolution operation [17].

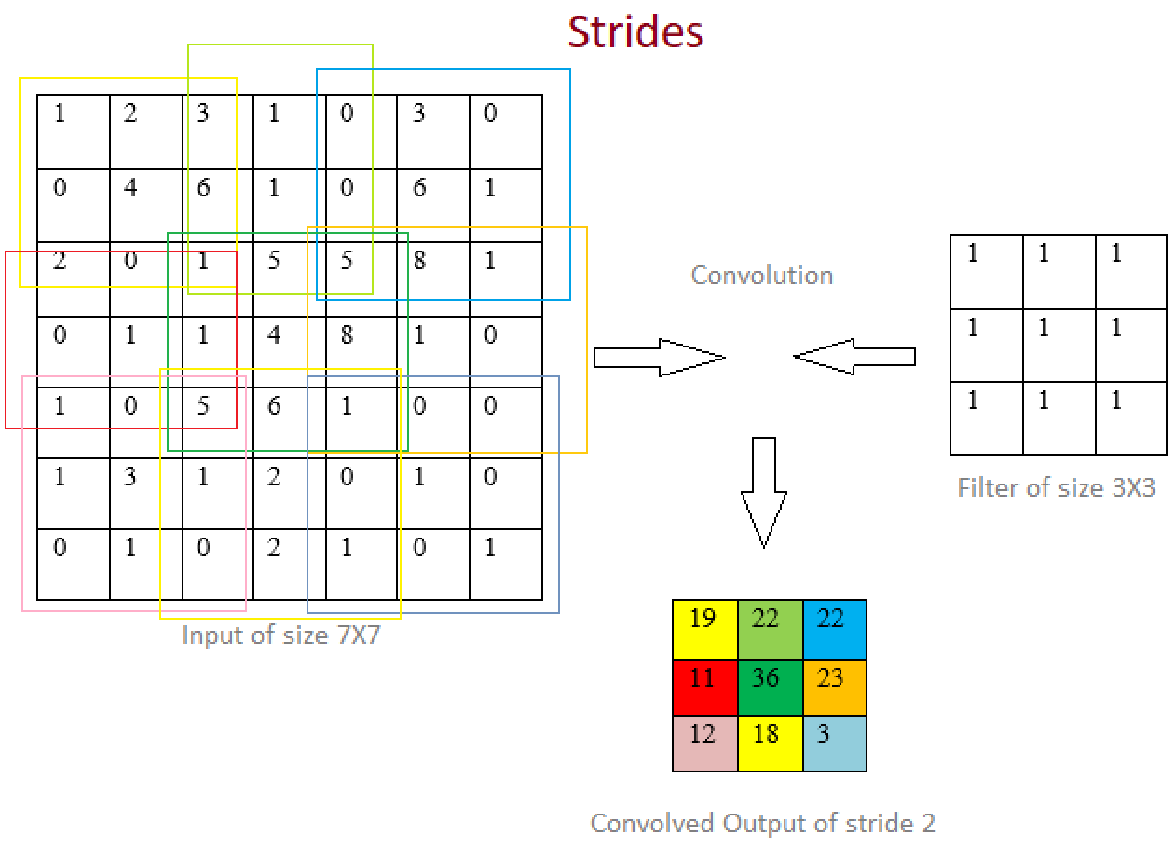


Figure 7 Example for Strides operation

4.2.4 Padding:

During convolution operation sometimes the filter is not perfectly fit the input image. To avoid this situation Padding operation is held on the input image. Two types of padding operations are applied to the input image during the convolution operation. [13]

They are as follows:

Zero Padding: Pad the input image with zeros as shown in the below Figure 8. The zero padding is done vertically or horizontally based on the requirement.

Valid Padding: In valid padding, the portion of the input image is dropped which is not perfectly fit the filter as shown in the below Figure 8.

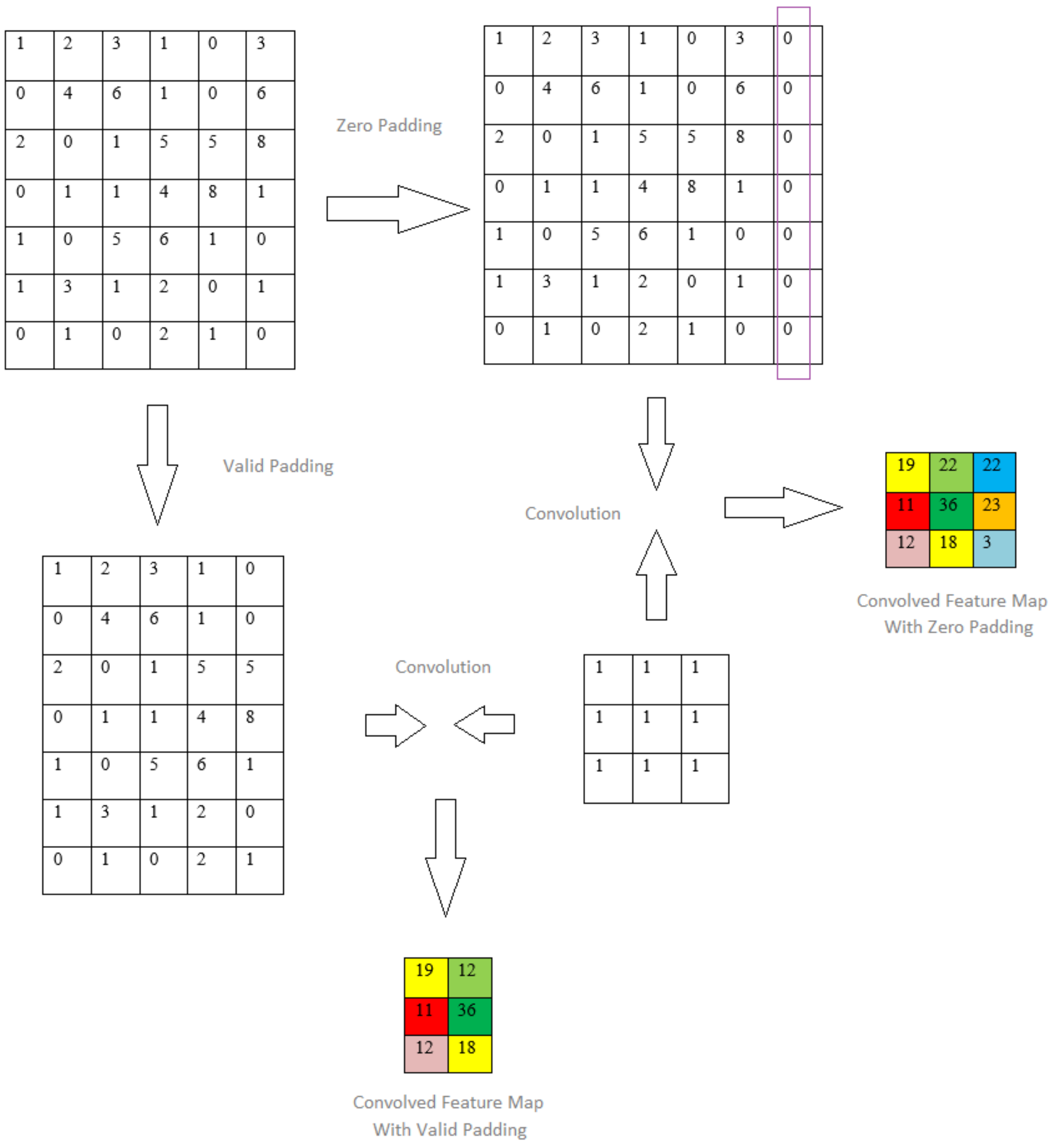


Figure 8 Example for Padding Operation

4.2.5 ReLU Layer:

The ReLU stands for the rectified linear unit where the rectifier is an activation function. The ReLU defines the positive part of the function arguments. The ReLU layer converts all the negative values to zero and all the positive values remain the same. The general equation of the ReLU is given as. [17]

$$f(x) = \begin{cases} x, & if\ x > 0 \\ 0, & otherwise \end{cases}$$

The output of the convolution layer is fed as input to the ReLU layer. The ReLU operation makes all the negative values to zero as shown in Figure 9. The output is called a Rectified feature map.

They are different types of ReLU.

Noisy ReLU: The Gaussian noise is added to the positive values and negative values are replaced with zero. The equation is given below. [15]

$$f(x) = \begin{cases} x + Y, & if\ x > 0 \\ \quad 0 \quad , & otherwise' \end{cases} \qquad Y = N(0, \sigma(x))$$

Leaky ReLU: The negative values are multiplied with the factor of 0.01 and the positive values remain the same. [15]

$$f(x) = \begin{cases} x, & if\ x > 0 \\ 0.01 * x, & otherwise \end{cases}$$

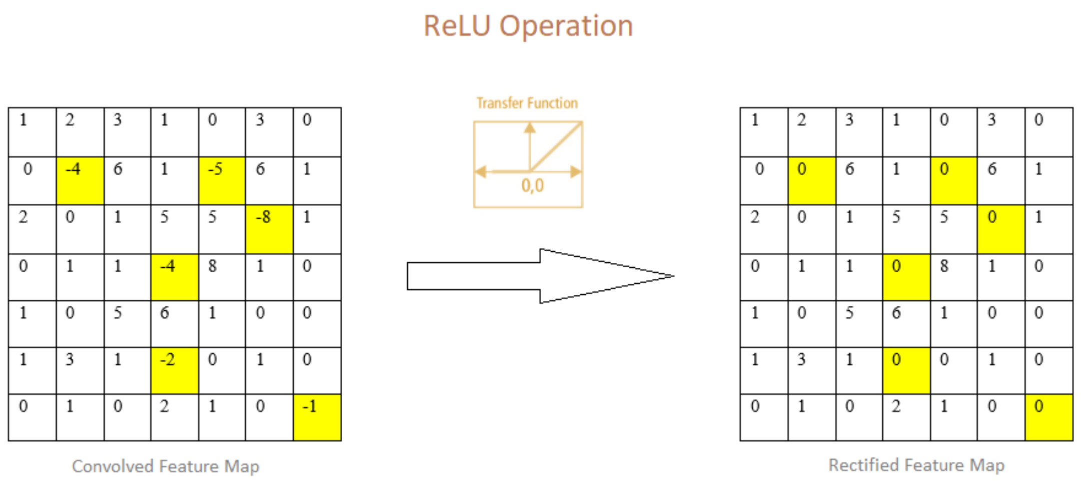
Parametric ReLU: The coefficient of leakage is made as the parameter, which helps to learn along with the other neural network. [15]

$$f(x) = \begin{cases} x, & if\ x > 0 \\ ax, & otherwise \end{cases}$$

ReLU 6: The positive values of above value 6 are set to 6 and negative values are set to zero and the in-between values are not changed.

$$f(x) = \begin{cases} 6, & if\ x > 6 \\ x, & if\ x > 0\ and\ x < 6 \\ 0, & otherwise \end{cases}$$

The output convolution layer is fed to the ReLU layer. The ReLU layer removes all the negative values such as all black elements and only keeps positive values such white and Gray elements the resulting image is shown in the below image.



1	2	3	1	0	3	0
0	-4	6	1	-5	6	1
2	0	1	5	5	-8	1
0	1	1	-4	8	1	0
1	0	5	6	1	0	0
1	3	1	-2	0	1	0
0	1	0	2	1	0	-1

1	2	3	1	0	3	0
0	0	6	1	0	6	1
2	0	1	5	5	0	1
0	1	1	0	8	1	0
1	0	5	6	1	0	0
1	3	1	0	0	1	0
0	1	0	2	1	0	0

Figure 9 Example of ReLU Operation on 7X7 matrix

4.2.6 Pooling Operation:

The pooling operation is used with the conjunction of the convolution layer. The main purpose of the pooling operation is to extract the location of the discovered feature. The pooling operation provides the translation in variance. We have different images of the same object and the object in the image is tilted with different angles. The convolution layer detects the object in the image. Now we have to detect the location of the object in the image, this is achieved with pooling operation [17].

They are different types of pooling operations

Max Pooling: It returns the max value of the window. The window is slid throughout the input. The output of the max pooling layer is as shown in Figure 10.

Mean Pooling: It returns the mean value of the window. The window is slid throughout the input. The Pooled feature map is shown in Figure 10.

Sum Pooling: It returns the sum of all values of the window and the window is slid throughout the input. The output of the sum pooling layer is as shown in Figure 10.

In most cases, max Pooling is used. In max pooling operation a window of size 2X2 is used as the input. The maximum value of the window is taken. As shown in the below Figure 10.

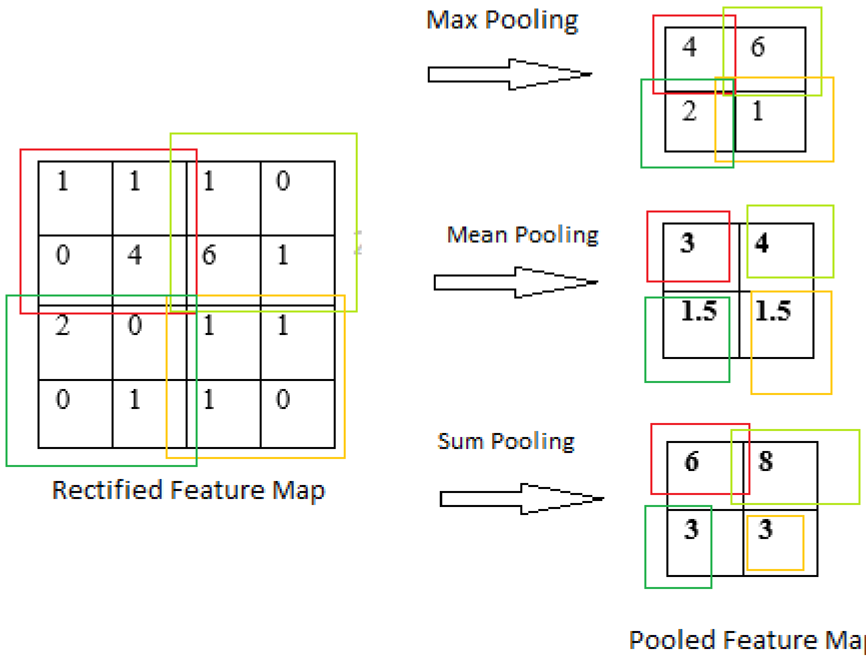


Figure 10 Example of Pooling operation on 4X4 matrix

4.2.7 Flattening Layer:

As the name states that we are going to flatten the input. The output of the pooling layer is fed as input to this layer. In the below Figure 11 the pooled feature map of size 3X3 is taken as input to the flattening layer. The flattening layer will convert the 3X3 pooled feature map to a 9X1 matrix. If the input is a 4X4 pooled feature map the output of the flattened layer is a 16X1 matrix.

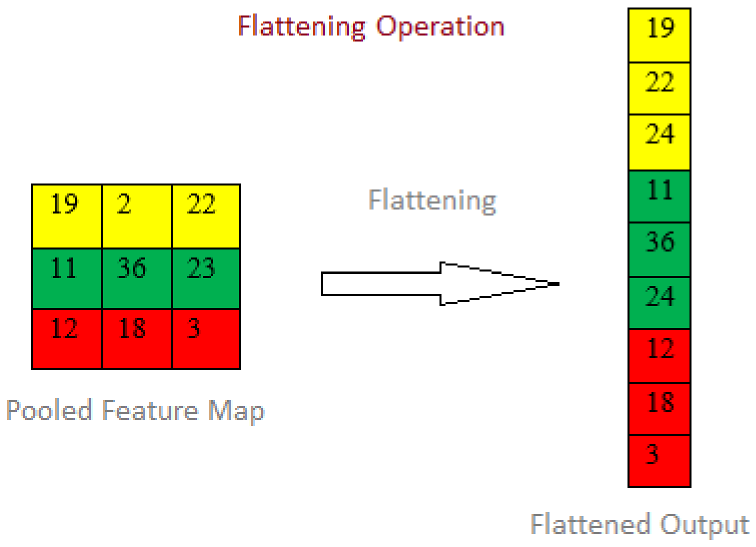


Figure 11 Example of Flattening operation on 3X3 matrix

All pooled feature maps are fed to the flattening layer. The flattening layer converts all the pooled feature maps to the flattened output. The flattened output is used as input to the next layer called the Fully connected layer. It can also say that flattened output is given to an artificial neural network (ANN) for the prediction process.

4.2.8 Fully Connected Layer:

The flattened outputs from the previous layer are given to the fully connected layer. In the fully connected layer is used for determining the error or loss of the net and class recognition. The fully connected layer aims to take the data from the previous steps and combine the features into a wide variety of attributes that make the convolution neural networks more capable of image classification. Image classification is the main goal of convolutional neural networks.

The input contains the data in the format of vector that was created in the previous flatten layer. The output will class of the image or prediction of the image.

The input image which is convolved, pool and flattened is passed through the fully connected layer. The artificial neural network in the convolutional neural network is named as a fully connected layer. The fully connected layer does the same function that an artificial neural network does. At the end of the fully connected layer issues its prediction. If the prediction of the fully connected layer is rose by 75 percent but the actual image is of another class, the error should be calculated. This is called cost function that is mostly mean squared error. But, in the convolutional neural network, it is called a Loss function. The loss calculation is done by using the cross-entropy function. The loss function is used to know how accurate our network is and to use increase the efficiency of the network. It means to change some things or parameters in our network to increase efficiency and to optimize the network. The parameters which are to be modified to increase the efficiency are weights which are in blue colour lines which are shown in the below Figure 12 which are also known as synapses and the feature detectors. The features are more important because each class will have some important features like leaf structure, stem structure, colour and so on which are used to detect new images. Class recognition of the input is done with the features The weights of the network are multiplied by the input vector. The output value is high the feature of input and feature is saved during the previous steps during training processes. When the rose image is passed through the network the rose features which are saved will give high value as shown in Figure 12. In the same way, the features of the rose class will get high value than the features of the cat class. Finally, class is recognized as the rose [19].

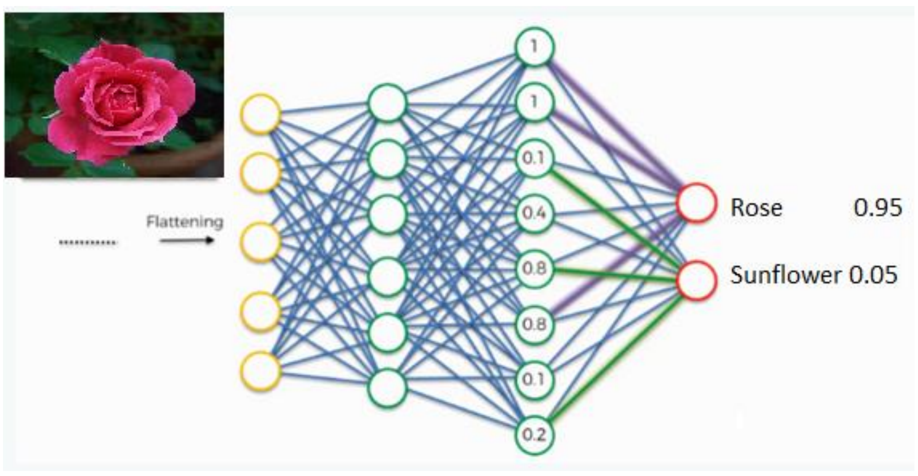


Figure 12 Image Classification using Fully connected layer

The object detection algorithm selected to run the microcontroller is MobilNet SSD v2. The MobileNet SSD v2 is selected based on the computational cost and parameters to train. The details are discussed in chapter 7 Implementation of MobileNet SSD v2

Batch Normalization: In the new architecture of Deep neural networks (DNN) batch norm is considered as one important layer. The batch normalization is also referred to as batch norm or batch norm layer. The batch norm layer is an algorithmic method. The batch norm makes the training phase faster in deep neural networks (DNN). The batch norm layer is placed before and after the nonlinear functions or layers like ReLU. The batch norm layer makes the algorithm learn more independently. The output from the previous layer is sent to the batch norm layer to normalize the data. Due to this normalization overfitting of the model can be avoided. The batch norm consists of normalizing activation vectors. The hidden layers of the batch norm layer use the first and second statistical moments such as mean and variance [35].

Hardware components: To implement the model we need to select specific hardware like microcontroller, soil moisture sensor, water level sensor, camera module, motor pump, and an Add on PCB to the microcontroller with sensors and actuators. For the model implementation, the USB camera is used, the detailed description of the remaining components is discussed in chapter 5 Hardware Components Used.

5. Hardware Components Used

5.1 Jetson Nano

Jetson Nano is an embedded development kit developed by NVIDIA. Jetson Nanos are compact in size and powerful computers. They can run multiple neural network algorithms in parallel. Mainly for the application such as classification, detection, and segmentation of input images and speech recognition processing. The Jetson Nano consists of a power jack for input, four USB ports, one Ethernet port, and forty GPIO pins to communicate with sensors and actuators as shown in the below Figure 13.

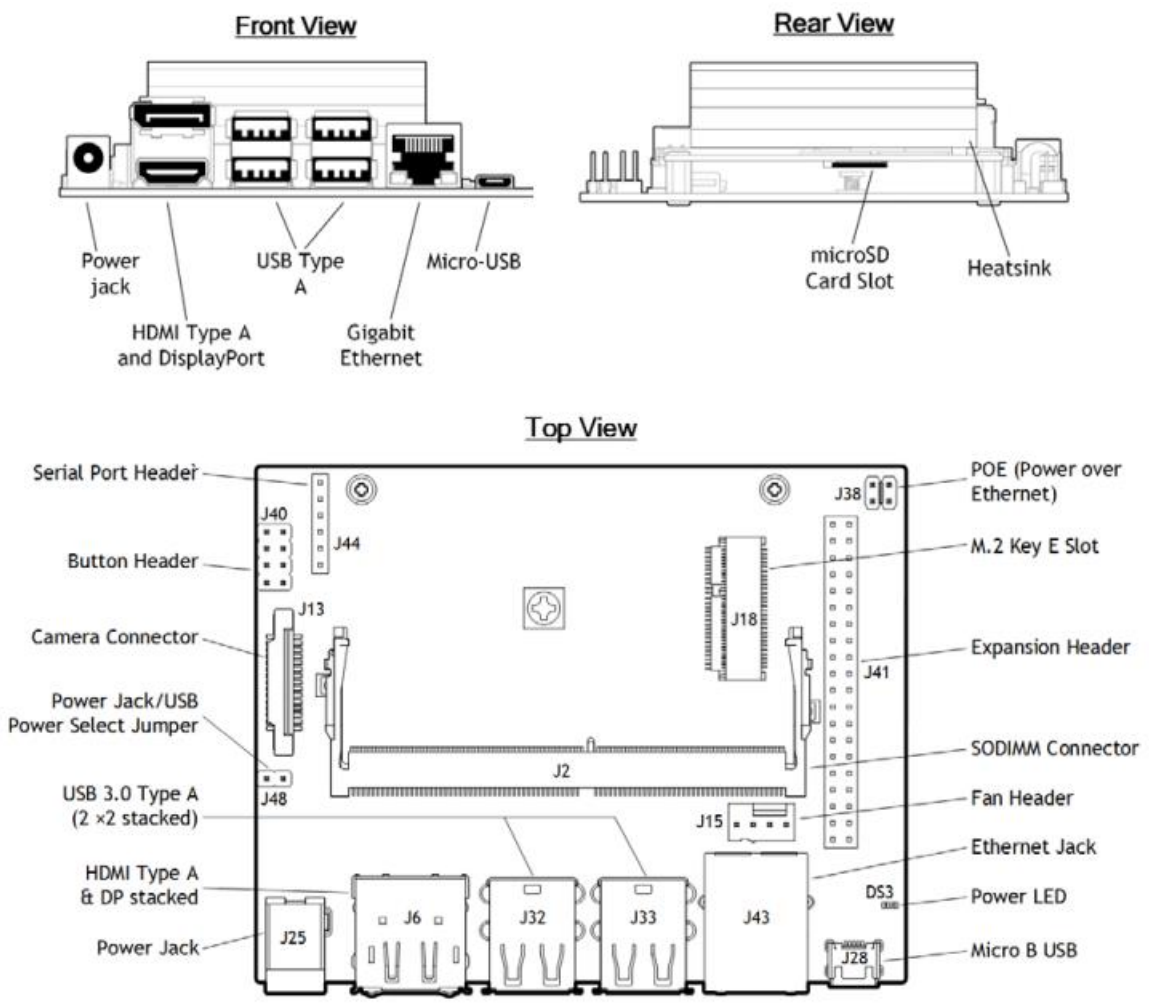


Figure 13 Jetson Nano Developer Kit [11]

Interface Details [11]

This section highlights some of the Jetson Nano Developer Kit interfaces.

1. The slot [J501] is used to place a MicroSD card loaded with JetPack OS.
2. The passive heatsink supports 10W module power usage at 25° C ambient temperature.
3. The [DS3] Power LED is used as a power indicator for the jetson nano developer kit.
4. The [J6] is used for HDMI and DP connector stack.
5. The [J15] consists of 4-pin for fan control header. The output is Pulse Width Modulation (PWM) which is used for the fan.
6. The camera module, Keyboard, Mouse, etc. are connected by the UDB 3.0 type A connector which is represented by the [J32 and J33].
7. The 40-pin [J41] includes Interface pins and two power pins are 3.3V and 5V. Except for a few pins others are used as GPIO pins at the 3.3V level. Pins 3 and 5 are used as SDA and SCL for I2C. The pins which are used for UART TX and RX are pins 8 and 10, respectively. By using the Micro USB, the power supply is 5V⎓2A which it is possible to use 2GB RAM of Jetson nano. To use 4GB RAM of jetson nano, Micro USB power is not sufficient to run the object detection algorithm as per our requirement. Therefore, the required power supply 5V⎓4A which is supplied by [J25] power jack. By using the power supply 5V⎓4A from J25 power jack adequate to use 4GB RAM of Jetson nano.

Based on the analysis of Jetson TX2, Jetson Nano, and Raspberry Pi it seems that Jetson Nano is sufficient and cost-effective for our model [4]. The Table 1 gives the comparison price, CPU, GPU, Memory, Accuracy in percentage with dataset size of 20K of three boards.

Table 1 Comparision of Jetson TX2, Jetson Nano, Raspberry PI [4]

Board Name	Price (Euro)	CPU	GPU	Memory	Accuracy (%) Dataset size of 20K
Jetson TX2	289 euros	Quad-Core ARM Cortex-A57 @ 2GHz + Dual-Core NVIDIA Denver2 @ 2GHz	NVIDIA Pascal 256 CUDA cores @ 1300MHz	8GB 128-bit LPDDR4 @ 1866Mhz, 59.7 GB/s	94.6
Jetson Nano	79.99 euros	Quad-Core ARM Cortex-A57 64-bit @ 1.42 GHz	NVIDIA Maxwell w/ 128 CUDA cores @ 921 MHz	4 GB LPDDR4 @ 1600MHz, 25.6 GB/s	94.5
Raspberry PI	39.99 euros	Quad-core ARM CortexA72 64-bit @ 1.5 GHz	Broadcom Video Core VI (32-bit)	8 GB LPDDR4	-

5.2 Capacitive Soil Moisture sensor

The resistive type of soil sensors is low-cost. These consist of two conductive prongs, sensor measurements are made according to the conductivity of the two prongs. The working of resistive type soil sensor is good at the starting but eventually, the sensor reads the values very bad, this is because of oxidization of the exposed metal. The resistivity measurement will be not even if the gold plats are used for the sensor. The re-calibration should be done constantly for the code. Also, resistive measurements in loose soil are not good [33]. This is the reason the capacitive type of soil moisture sensor is popular with lot of advantages [34].

The capacitor consists of three important parts, they are positive plate, negative plate, and dielectric material. The two electrically charged plates of the capacitor are separated by a dielectric medium. The capacitive soil moisture sensor works on the working principle of a capacitor. The soil moisture is measured by calculating the changes in the capacitance which is caused by the changes in the dielectric medium. The capacitive measurement uses only probes, because of this they do not have exposed metals and no DC currents will go-to plants. This capacitive soil moisture sensor consists of a built-in capacitive touch measurement system built into the ATSAMD10 chip. The range of values to be considered for the measurement is about 200 (very dry) to 2000 (very wet). This chip also reads temperature from the internal temperature sensor on the microcontroller. This sensor is not high precision but good to measure temperatures between + 60 to - 2 degrees Celsius [33].

The capacitive soil moisture sensor is shown in the below Figure 14. It consists of four pins one for GND, one for VCC, and the remaining two are SCL and SDA for I2c communication with the microcontroller.

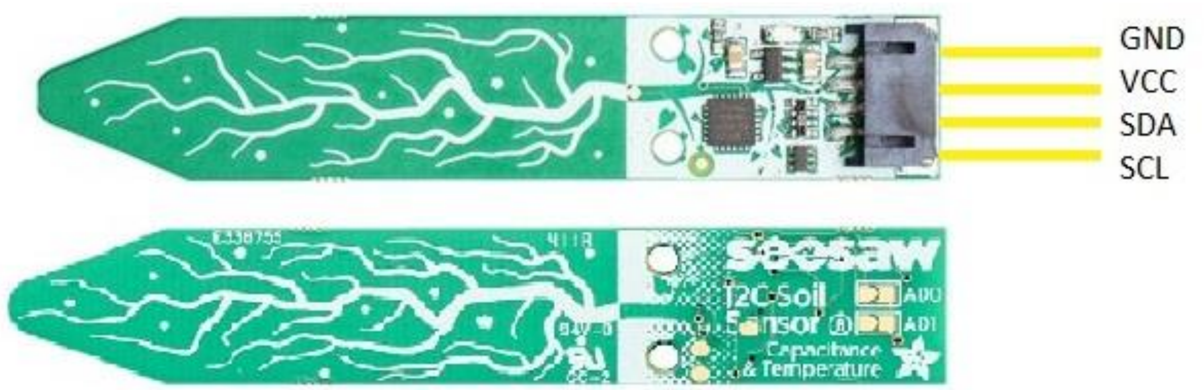


Figure 14 Capacitive Soil Moisture sensor

5.3 Water Level Sensor

The water level sensor is used to measure the level of water content in the water tank. The watering of the plant should be done when the water is available in the tank. When the pump is on without water it damages the pump. So, the water level in the tank must be monitored with a Water level sensor as shown in the below Figure 15. Water level sensors are not only used to measure water level but also used to monitor sump pits and detect rain fall and detect leakage [30].

The water level sensor consists of a series of ten copper traces. The then copper plates are exposed to the environment. Out of ten exposed copper traces, five traces are used as power traces and another five traces are used for sense traces. the respective red dot traces are power traces and blue dots represent sense traces as shown in the Figure 15. The traces are not connected physically. They are bridged when they are submerged in the water.

The water level sensor consists of three pins namely GND, VCC, S as shown in the Figure 15. The Pin S source gives the voltage reading according to the water level. The working principle is quite simple of variable resistance. The series of exposed parallel conductive traces together works as a variable resistor. The resistance varies as per the water level. The resistance increases when the water level is less, and the resistance decrease when the water level is high. The sensor immersed in more water level the conductivity increases results in less resistance. The sensor in less water level the conductivity decreases results in more resistance. The output analog voltage is read from the pin source or signal pin.

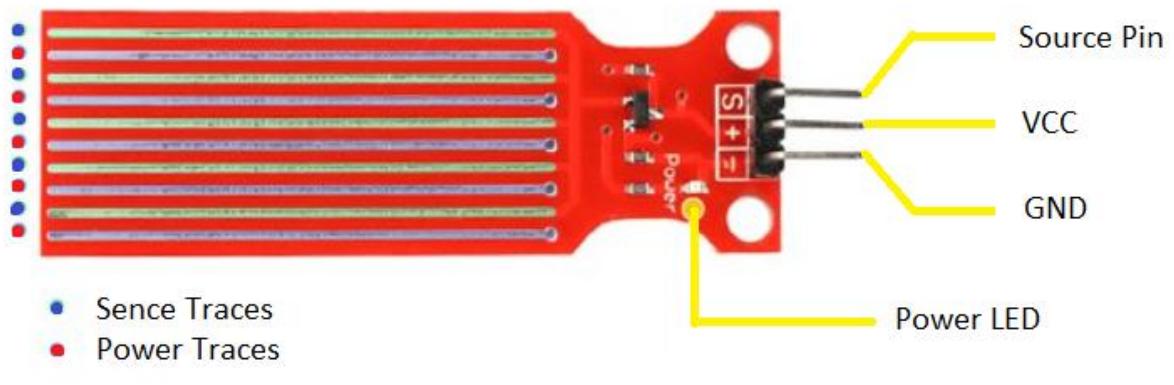


Figure 15 Water Level Sensor

5.4 Add on PCB board to connect sensors and actuators to Jetson Nano

The Add on PCB shown in the below Figure 16 is designed to integrate the capacitive soil moisture sensor, water level sensor, and motor pump with the jetson nano. The board consists of L2938 Driver IC and ADC converter IC. Each pin header is used for a specific purpose. The forty-pin header [J1] is used to connect the jetson Nano developer kit. The four-pin header [J8] is used to connect capacitive soil moisture sensor, the sensor pins should be connected to respective SCL, SDA, VCC, GND represented on the PCB board for the proper establishment of I2c communication between soil moisture sensor and jetson nano. The three-pin header [J9] is used to connect the water level sensor, the sensor pins should be connected to respective GND, VCC, S represented on the PCB board for the proper establishment of communication between ADS1115 (ADC with I2c chip) and sensor. The signal pin is connected to channel 0 in ADS1115 internally which converts the analog sensor values to digital and sends the values to jetson nano using I2c communication. The four-pin header [J10] is used to connect the four channels of ADS1115.

The terminal screws are attached to the two-pin headers of [J6] and [J7] to attach the motor pump. The communication between motor pumps and jetson nano is done through motor driver L293d which is connected to a sixteen-pin header [U1]. The diodes are connected to two-pin headers named [D1, …, D8] which act as a protection circuit to motor pump from back EMF and unusual current spikes. The pin header [J5] is connected to the power jack for the motor pump power supply. The four-pin header [J4] is used to connect jumper pins to provide VCC and ground to motor driver from jetson nano. The six-pin header [J2] and [J3] is used to connect jumper pins to provide inputs and enable (IN1, IN2 and ENABLE) logic to the motor driver from jetson nano to control the motor pumps direction which is connected to [J6] and [J7]. The two-pin headers [J11] and [J12] are used to connect the ground plates between jetson nano and individual isolated ground plates provided for motor and sensors. The design and implementation of the PCB are discussed in a later section.

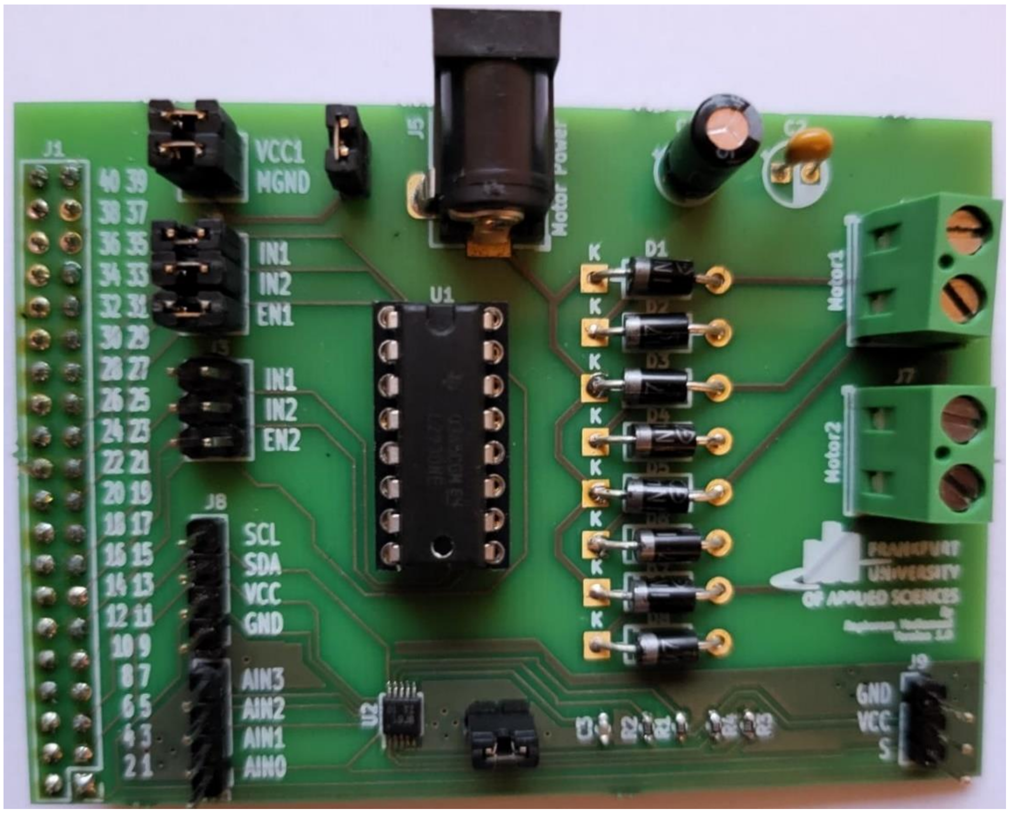


Figure 16 Add on PCB for Jetson Nano to integrate Sensors and Actuators

5.5 ADS1115 IC (16 Bit Analog to Digital converter with I2c communication)

The ADS1115 consists of a multiplexer, Programable gain amplifier (PGA), 16-bit Analog to digital converter, Oscillator, Voltage Reference, Comparator, and I2c interface units as shown in the below Figure 17. The multiplexer is connected to a 16-bit ADC unit through PGA. The oscillator is used to produce the clock signals for the internal units. I2c interface is used to send the data to master by proper selection of address value. This unit is connected to the pins SCL and SDA [12].

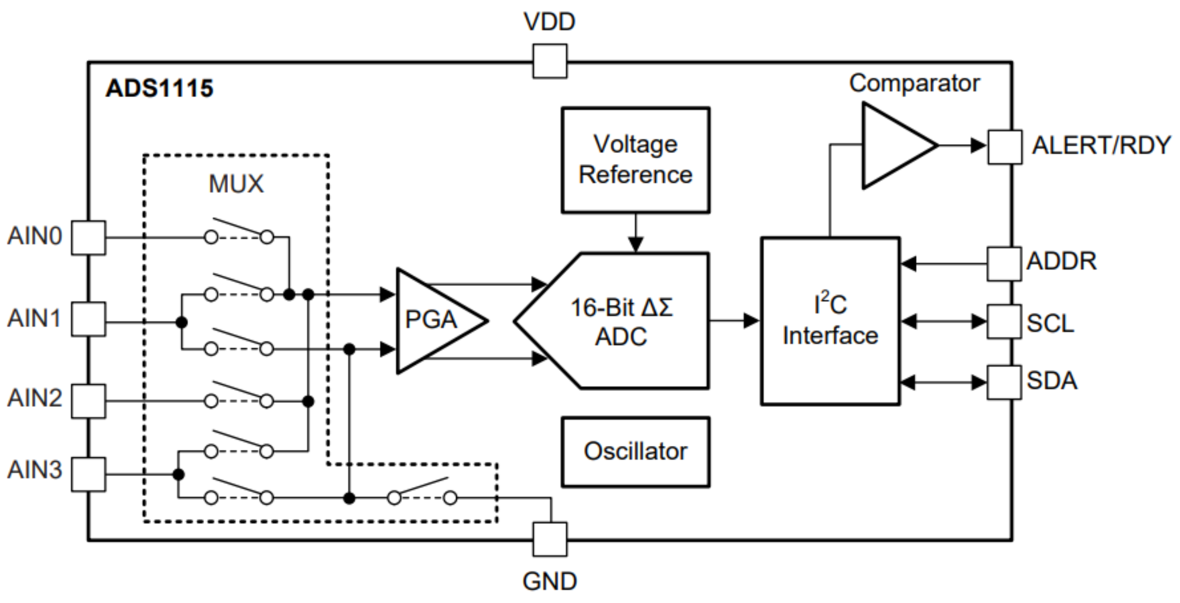


Figure 17 ADS1115 IC [12]

The ADS1115 IC consists of ten pins as shown in the Figure 18. Pin 1 is the ADDR pin, this pin is a digital input pin it is used to select the I2c slave address. The selection I2c address is based on the selected input to the ADDR pin as shown in the below Table 2. For the project purpose, the ADDR pin is connected to GND to make the address 1001000.

Table 2 ADDR pin connection and corresponding Slave address [12]

ADDR Pin connection	Slave Address
GND	1001000
VDD	1001001
SDA	1001010
SCL	1001011

Pin 2 is an ALERT/RDY pin, it is a digital output pin. It is used to get the status of the conversation ready and used as comparator output. Pin 3 is a GND pin, used for the ground. The pins 4, 5, 6, 7 are analog input channels AIN0, AIN1, AIN2, AIN3 respectively. These channels are used to take the analog inputs and send the data to the 16-bit analog to digital conversion unit through multiplexer as shown in the Figure 17. Pin 8 is a VDD pin this pin is used as a power supply to the ADS1115 IC. Pin 9, 10 are I2c interface pins SCL and SDA pins. These pins are used to connect the SCL and SDA pins of jetson nano to read the ADC values of the water level sensor.

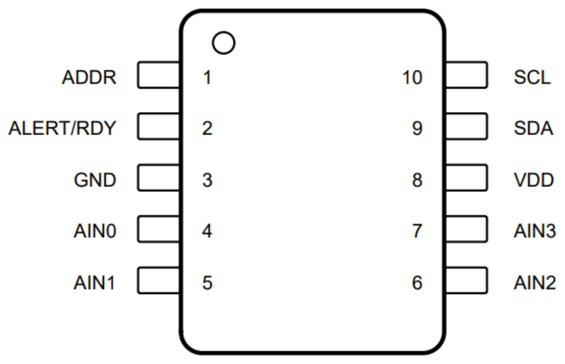


Figure 18 ADS1115 Pinout diagram [12]

5.6 Adafruit Water DC motor pump 12 volts

The Adafruit water pump is a geared-down 12 volts DC motor as showed in the Figure 19. The Dc motor pump has a lot of torque. The pump consists of a ‘clover’ pattern of rollers inside it. The clover presses when the motor turns. When the clover presses on the tube, then the fluid passes through it. The motor speed can be controlled by PWM input to the motor. Based on the motor speed the flow rate of the fluid depends. Higher the speed of the motor higher the flow rate, lower the speed of the motor lower the flow rate. The direction of the fluid flow can be changed by changing the direction of the motor. The pump can self-prime itself, it does not need to be primed [13]. The motor pump can be easily controlled with motor driver ICs like L293D.

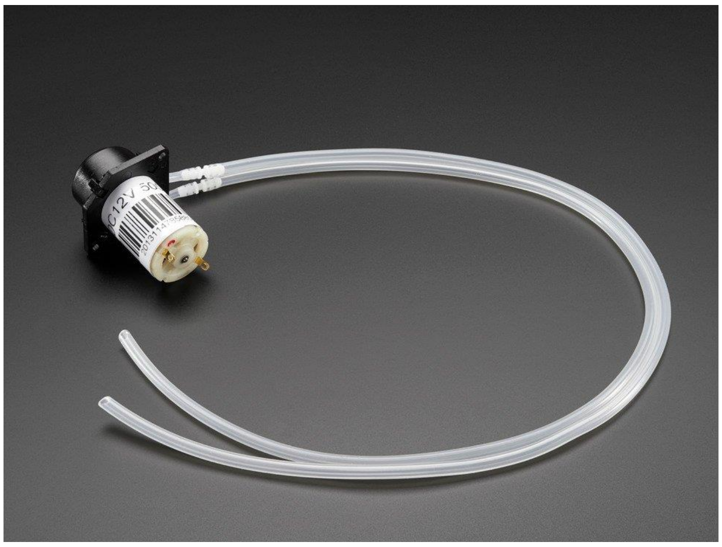


Figure 19 Water DC motor pump 12 volts [13]

6. DESIGN AND IMPLEMENTATION OF PCB

To fulfill the requirements of this project, a customized PCB is required. The customized PCB is designed to connect jetson nano with a capacitive soil moisture sensor, water level sensor, motor pump, and power supply to the motor pump. Design and implementation of PCB are made using the Kicad tool.

6.1 Kicad WorkFlow:

The Kicad is similar to other PCB software tools, but it is special because the workflow in the schematic design considers the components and footprints are separate entities. The proper mapping of the components and footprints should be done. Kicad has plenty of libraries available for different components and footprints. Kicad has a special feature to implement customized footprints. The Kicad workflow consists of two important tasks one is to make the schematic, and another is to layout the board [14]. In the schematic task the circuit diagram and mapping of components and footprints need to be done. In the layout (PCB View) section arrangement of components and connecting wire traces and ground plate arrangements will be done.

6.2 Design of Schematic diagram in Kicad

The first step is to make the schematic in the Kicad. The [J1] 2 x 20 pin header is selected to connect the jetson nano GPIO pins or jetson nano interface pins to the PCB. Each GPIO pin of Jetson nano represents the respective pin in the [J1] 2 x 20 pin header. How each pin of [J1] is connected to other pins is given in Table 3. The [U1] 2 x 8 pin header is used to connect the L293d motor driver IC. The pins which are connected to [U1] are given in Table 4. The [J2] and [J3] are 2 x 3 pin headers, which are selected to place jumpers and to establish the connection between input pins and enable pins of L293d motor driver and Jetson nano. The three pins of one side Pin 2, Pin 4, Pin 6 on the [J2] 2 x 3 pin header are connected to Enable1, Input1, Input2 of the L293d motor driver to control motor 1 the other three pins Pin 1, Pin 3, Pin 5 are connected to the Pin 23, Pin 21, Pin 19 of the [J1] jetson nano interface pins. The connections of [J2] are shown in Table 5. The three pins of one side Pin 2, Pin 4, Pin 6 on the [J3] 2 x 3 pin header are connected to Enable1, Input1, Input2 of the L293d motor driver for motor 2 the other three pins Pin 1, Pin 3, Pin 5 are connected to the Pin 15, Pin 13, Pin 11 of the [J1] jetson nano interface pins. The connections of [J3] as described are tabulated in Table 6. The [J4] 2 x 2 pin headers are selected to place jumpers to establish the power supply VCC and ground GND to the motor driver L293d from the jetson nano pins 4 (5V) and 34 (GND). The outputs of the L293d pins are connected to the [J6] and [J7] 1 x 2 pin headers through the diode bridge connection as shown in Figure 20. These pin headers are soldered with screw terminals to connect the motor pump terminals. The pin headers represented with [D1] to [D8] are used to connect the 1N4007 diodes, these diodes are used as a protection circuit to motor pump from back emf and unusual current spikes. The [J5] is pin header is used to connect the power jack for supply power to the motor pump. The [J11] and [J12] 1 x 2 pin header are used to connect the jumpers to make and break the ground to the motor ground plate and sensor ground plate. The current project requires two separate ground plates on the PCB to avoid noise while reading the sensor values, one ground plate if for the motor driver and motor pump, and the other is for sensors. Pin 2 on Jetson nano is a power pin with an output of 5V, pin 5 (SCL) and pin 3 (SDA) are used for I2c communication.

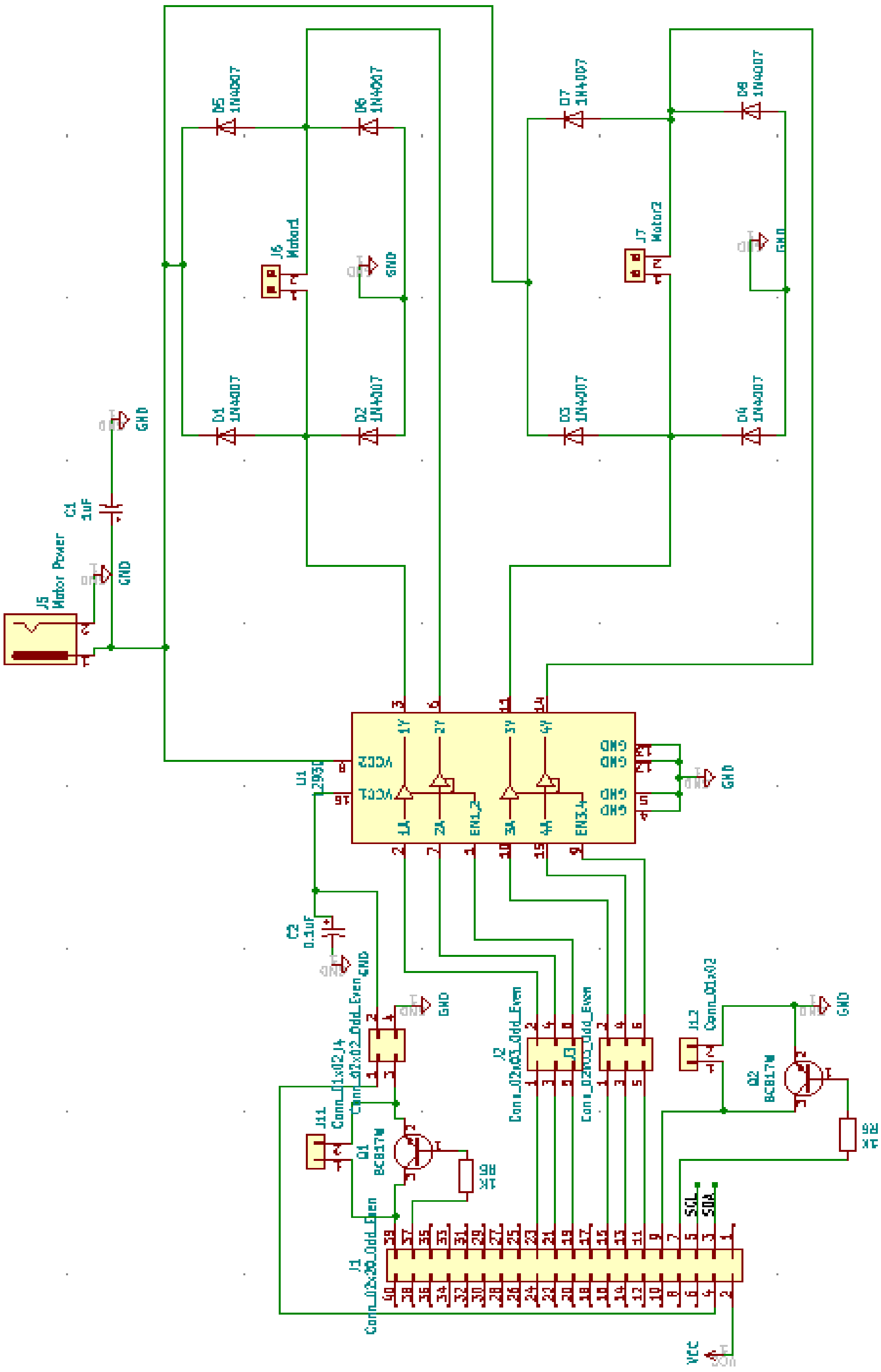


Figure 20 Schematic diagram 1

Table 3 J1 Conn_02x20 (40 pins) Pin header to connect Jetson Nano GPIO pins to PCB

PIN	Connected to
2 (VCC)	VCC (J1, PIN 2)
3 (SDA)	SDA (J8, PIN 3), SDA (U2, PIN 9)
4 (VCC)	(J4, PIN 1)
5 (SCL)	SCL (J8, PIN 4), SCL (U2, PIN 10)
11 (EN 2, Motor 2)	(J3, PIN 5)
13 (IN 2, Motor 2)	(J3, PIN 3)
15 (IN 1, Motor 2)	(J3, PIN 1)
19 (EN 1, Motor 1)	(J2, PIN 5)
21 (IN 2, Motor 1)	(J2, PIN 3)
23 (IN 1, Motor 1)	(J2, PIN 1)

Table 4 U1 Conn_02x16 (L293D)

PIN	Connected to
1 (EN 1, 2, Motor 1)	(J2, PIN 6)
2 (IN 1, Motor 1)	(J2, PIN 2)
3 (OUT 1, Motor 1)	(J6, PIN 1), D1, D2
4 (GND)	GND (Motor Ground plate)
5 (GND)	GND (Motor Ground plate)
6 (OUT 2, Motor 1)	(J6, PIN 2), D5, D6
7 (IN 2, Motor 1)	(J2, PIN 4)
8 (VCC 1)	VCC (J5, PIN 1)
9 (EN 3, 4, Motor 2)	(J3, PIN 6)
10 (IN 1, Motor 2)	(J3, PIN 2)
11 (OUT 1, Motor 2)	(J7, PIN 1), D3, D4
12 (GND)	GND (Motor Ground plate)
13 (GND)	GND (Motor Ground plate)
14 (OUT 2, Motor 2)	(J7, PIN 2), D7, D8
15 (IN 2, Motor 2)	(J3, PIN 4)
8 (VCC 2)	VCC (J4, PIN 2)

Table 5 J2 Conn_02x03 (Jumpers to connect Jetson Nano pins (J1) to L293D (U1) for Motor 1)

PIN	Connected to
1 (IN 1, Motor 1)	(J1, PIN 15)
2 (IN 1, Motor 1)	(U1, PIN 2)
3 (IN 2, Motor 1)	(J1, PIN 13)
4 (IN 2, Motor 1)	(U1, PIN 7)
5 (EN 1, Motor 1)	(J1, PIN 11)
6 (EN 1, Motor 1)	(U1, PIN 1)

Table 6 J3 Conn_02x03 (Jumpers to connect Jetson Nano pins (J1) to L293D (U1) for Motor 2)

PIN	Connected to
1 (IN 1, Motor 2)	(J1, PIN 23)
2 (IN 1, Motor 2)	(U1, PIN 10)
3 (IN 2, Motor 2)	(J1, PIN 21)
4 (IN 2, Motor 2)	(U1, PIN 15)
5 (EN 1, Motor 2)	(J1, PIN 19)
6 (EN 1, Motor 2)	(U1, PIN 9)

The [U2] 2 x 5 pin header is used to connect the ADS1115 IC. The VCC in the current Figure 21 represents the 5V power supply from pin 2 of the jetson nano interface pins J1. The pins 2, 9, 10 are connected to pin 2 of jetson nano via a 10k resistor as shown in Figure 21. The Pins 9 (SDA), Pin 10 (SCL) of the [U2] ADS1115 are also connected to pin 3 (SDA) and pin 5 (SCL) of the jetson nano interface pins respectively. The pin 1 ADDR of the [U2] ADS1115 is connected to the sensor ground plate. The Address pin 1 of ADS1115 IC is connected to the ground via a 10 k resistor to make the address of I2c communication as 1001000 (0x48). The pin 8 (VDD) of the [U2] ADS1115 is connected to pin 2 of the jetson nano interface pins. All the connections of [U2] are listed in Table 7. The [J10] 1 x 4 pin header connected internally to the pins 4, 5, 6, 7 of the [U2] ADS1115 IC respectively which are listed in Table 10. The [J8] 1 x 4 pin header is used to connect the capacitive soil moisture sensor, the pins 3, 4 are connected to SDA and SCL pins of the jetson nano interface pins. Pins 1, 2 are connected to the sensor ground plate and pin 2 (VCC) of the jetson nano interface pin. The connections of [J8] are listed in Table 8. The [J9] 1 x 3 pin header is used to connect the water level sensor. As listed in Table 9, Pin 1 and Pin 2 of [J9] are connected to the sensor ground plant and VCC, and Pin 3 is connected to

[U2] Pin 4 of ADS1115. All the connections are made by following the Aisler design rules for two-layer PCB [15].

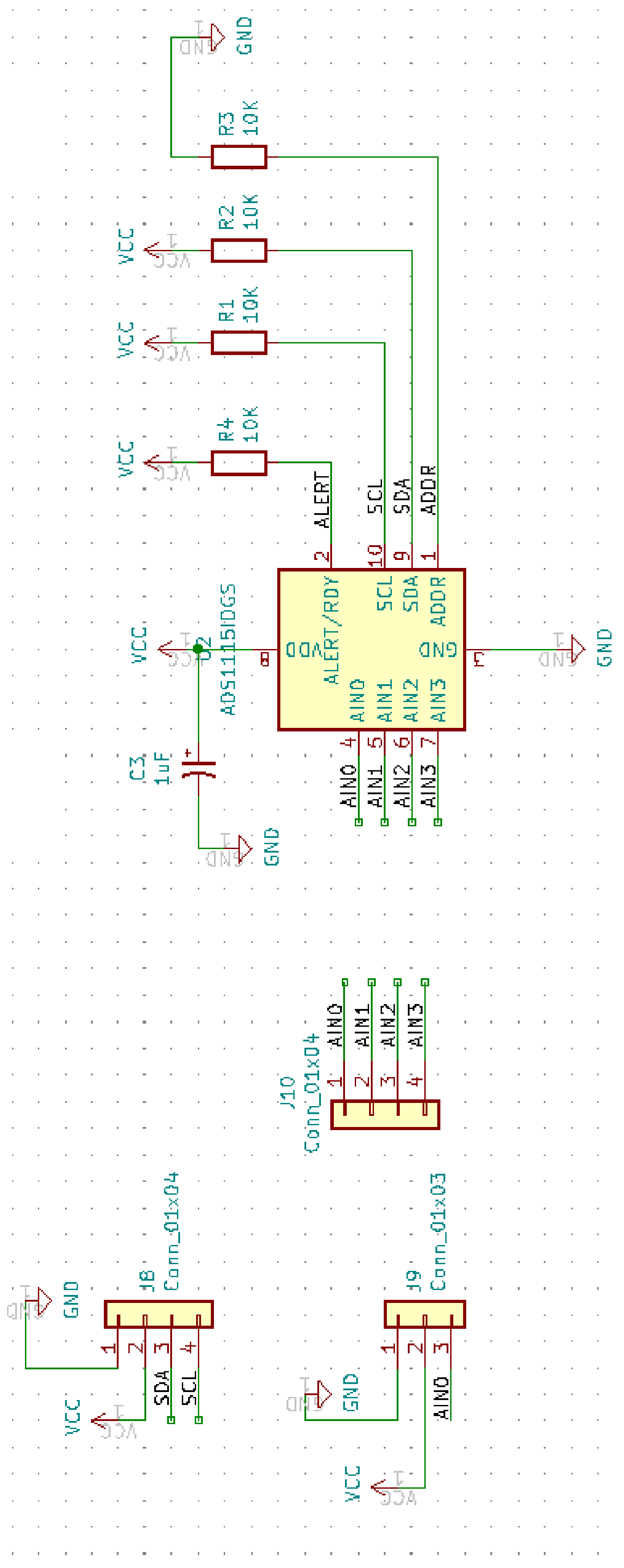


Figure 21 Schematic diagram 2

Table 7 U2 Conn_02x05 (ADS1115 IC)

PIN	Connected to
1 (ADDR)	GND (Sensor Ground plate) via R3
2 (ALERT)	VCC (J1, PIN 2)
3 (GND)	GND (Sensor Ground plate)
4 (ANI0)	AIN0 (U2, PIN 4)
5 (ANI1)	AIN1 (U2, PIN 5)
6 (ANI2)	AIN2 (U2, PIN 6)
7 (ANI3)	AIN3 (U2, PIN 7)
8 (VDD)	VCC (J1, PIN 2)
9 (SDA)	SDA (J1, PIN 3)
10 (SCL)	SCL (J1, PIN 5)

Table 8 J8 Conn_01x04 (Capacitive Soil Moisture Sensor)

PIN	Connected to
1 (GND)	GND (Sensor Ground plate)
2 (VCC)	VCC (J1, PIN 2)
3 (SDA)	SDA (J1, PIN 3)
4 (SCL)	SCL (J1, PIN 5)

Table 9 J9 Conn_01x03 (Water Level Sensor)

PIN	Connected to
1 (GND)	GND (Sensor Ground plate)
2 (VCC)	VCC (J1, PIN 2)
3 (SIGNAL)	AIN0 (U2, PIN 4)

Table 10 J10 Conn_01x04 (Addition Analog Input channels to ADS1115 IC)

PIN	Connected to ADS1115IDGS (U2)
1 (AIN0)	AIN0 (U2, PIN 4)
2 (AIN1)	AIN1 (U2, PIN 5)
3 (AIN2)	AIN2 (U2, PIN 6)
4 (AIN3)	AIN3 (U2, PIN 7)

The components which are used to place on the PCB such as diodes, capacitors, resistors, a power jack, ICs are listed in Table 11 with component values.

Table 11 Components List

Component	Value
C1, C3	1uF
C2	0.1uF
R1, R2, R3, R4	10K
D1, D2, D3, D4, D5, D6, D7, D8	1N4007
U1	L293D
U2	ADS1115
J5	Power Jack

6.3 Design of PCB view and 3D view of PCB

After drawing the schematic diagram of the required components, the components should be mapped with the proper footprints. For example, the [U1] 2 x 8 pin header should be mapped with motor driver IC L293D footprint properly by clicking the map footprint button in the GUI. Similarly, the [U2] 2 x 5 pin header should be mapped with ADS1115 IC footprint, also the remaining mounting pins and screw terminals and power jack and passive elements and SMDs should be properly mapped with their respective footprints. Then click on the PCB view option to view the real-time PCB view. Now arrange the components as per the requirements. For the project purpose, the components are assigned as shown below to minimize the size of PCB and be compatible with Jetson Nano GPIO pin headers. The wire tracings width of 2.5mm is selected which are connected to the motor pump and power supply jack. The remaining wire traces are of width 1.25mm is selected. The PCB consists of two ground plates. Because of the working voltage difference between motor pump and sensors to avoid noise while reading sensor values. The bottom plate and top plate of the PCB are as shown in Figure 22. After arranging the components in the desired place and wire traces click on the view then 3D view option to have a 3D view of the PCB. The 3D view of the PCB bottom side and the top side is shown in Figure 23. For the better view and high resolution of Figures 22 and Figure 23 please refer images in Appendix 1.

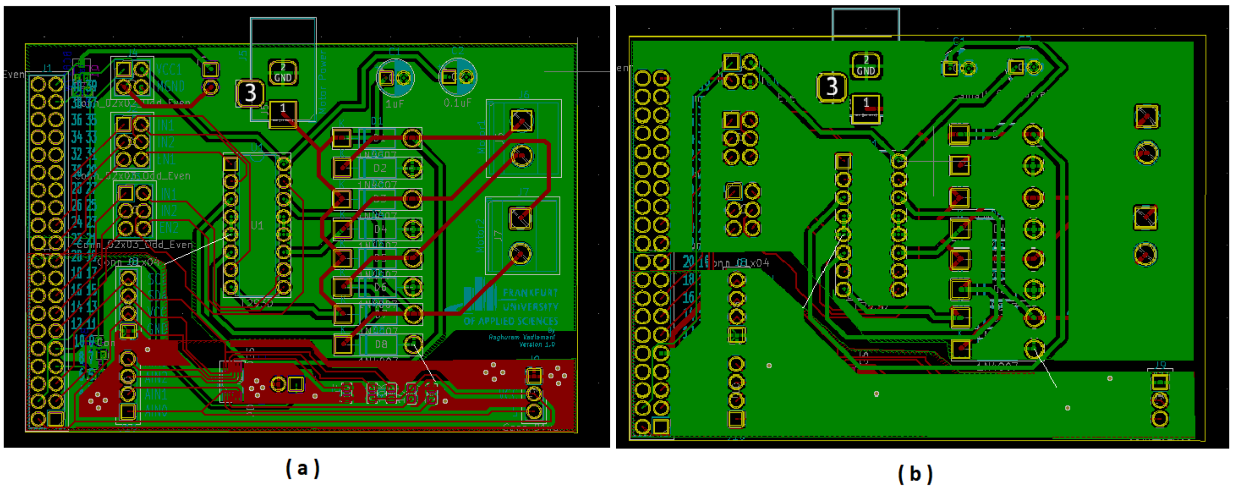


Figure 22 Component placement on PCB wiring (a) Top plate of PCB, (b) Bottom plate of PCB

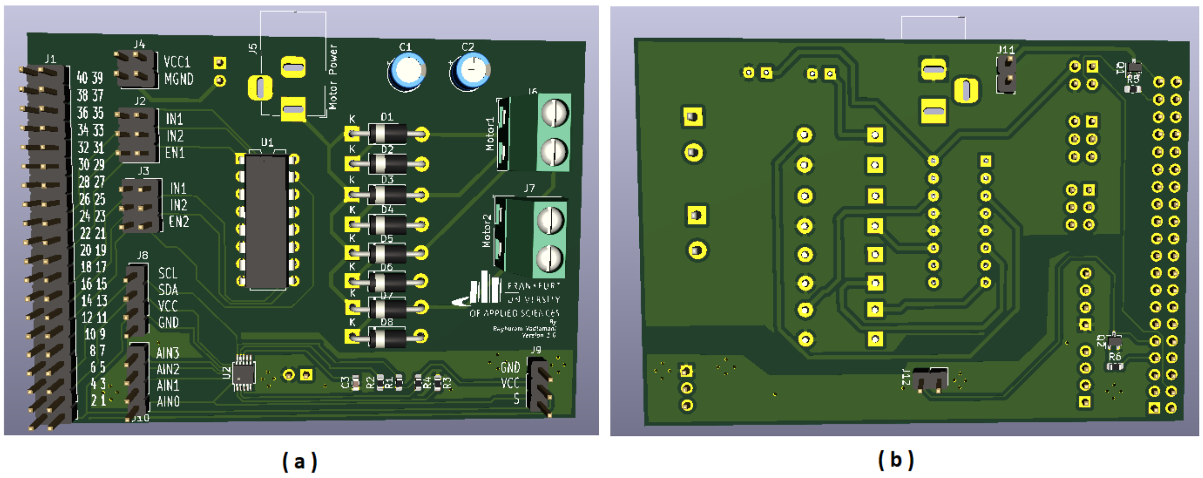


Figure 23 3D view of PCB (a) Top view of PCB, (b) Bottom view of PCB

7. Implementation of MobileNet SSD v2

The MobileNet v2 uses the inverted residuals and linear bottlenecks in the architecture. In mobile net v2, the input is sent through a 1 x 1 convolution layer, which is then processed using Relu6. To modify the dimensionality, the input is routed to a 1 x 1 convolution layer. The input transforms from a two-dimensional to a three-dimensional representation. The output of the 1 x 1 convolutional layer is sent to a 3 x 3 convolutional layer, channel-wise or depth-wise. Relu6 takes care of the depth-wise convolution. This layer's output is sent to a convolution layer with a size of 1 x 1. The output of the last layer is added to the first layer if the stride size is one; else, the output is not added input as indicated in the Figure 24. This is known as inverted residuals because it connects the first layer of low dimension to the last layer of low dimension with the middle layers of high dimension.

The input is a h x w x k vector, as indicated in the diagram below. Following the 1 x 1 convolution net, the input vector is processed via Relu6. This layer's output is h x w x (tk).

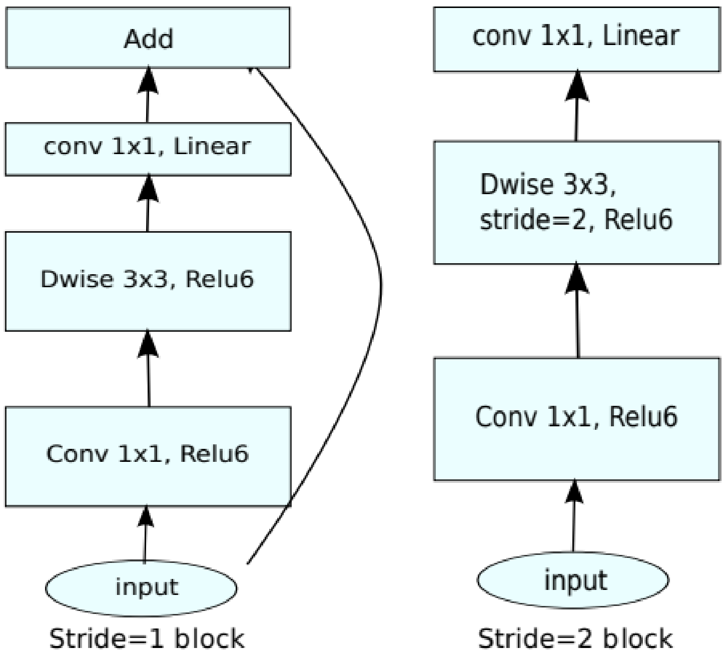


Figure 24 MobileNet V2 [2]

7.1 Depthwise Separable Convolution

The basic blocks of the MobileNet model id Depthwise separable convolutions. A depth-wise separable convolution is a form of factorized convolution. The standard convolution is factorized to depthwise convolution and 1 x 1 convolution. The 1 x 1 convolution is also called point-wise convolution. In depthwise convolution operation, a single filter is applied to each input channel. In pointwise convolution 1 x 1 convolution is applied. The 1 x 1 convolution is applied to combine depthwise convolution outputs. In depthwise separable convolution operation is split into two layers. One separate layer is to apply filters and the other layer is to combine the outputs. Whereas in standard convolution both filtering and combining take place in one step with a new set of outputs. Due to the factorization, there is a drastic change in the reduction of computation and model size.

The standard convolutional layer takes input dimensions of D_F x D_F x M where F is the input feature map. The standard convolutional layers produce the output with the dimensions of D_G x D_G x N where G is the output feature map.

Where

M – the number of input channels also called input depth

N – the number of output channels also called output depth

D_F – Spatial width and height of a square input feature map.

D_G – Spatial width and height of a square output feature map.

We are assuming the input and output spatial dimensions are the same because both feature maps are square.

The kernel K dimensions of the standard convolutional layer are D_k x D_k x M x N. Where D_k - Spatial width and height of a square kernel. M, N are the input and output depths as defined previously. The output feature map with stride one and padding of the standard convolution is given in below equation 7.1.1.

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} * F_{K+i-1,l+j-1,m} \qquad \text{eq. 7.1.1}$$

The computational cost of the standard convolutional operation is given in below equation 7.1.2.

$$D_k * D_k * M * N * D_F * D_F \qquad \text{eq. 7.1.2}$$

The computational cost depends multiplicatively on the kernel size Dk x Dk, the number of input channels M, the number of output channels N, and the size of the feature map Df x Df.

MobileNet models keep the interest on the above terms and their interactions. To produce outputs for the standard convolution operation based on the convolutional kernels and combining features that affect the filtering feature. To reduce the computational costs, the filtering and combining steps are split into two steps by using the concept of factorized convolutions which are called depthwise separable convolutions.

Depthwise separable convolution consists of two layers named depthwise convolutions and pointwise convolutions. The depthwise convolutions are used to apply a single filter per each input channel. Pointwise convolution is a simple 1 x 1 convolution. The 1 x 1 convolutions are used to create a linear combination of the output of the depthwise layer [1]. Both layers use batch norm and ReLU in mobileNets.

The depthwise convolution with one filter per each input channel in the mathematical equation is given in below equation 7.1.3

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m,} * F_{K+i-1,l+j-1,m} \qquad \text{eq. 7.1.3}$$

Where

$\hat{K}$ – depthwise convolutional filter of size D_k x D_k x M.

$\hat{G}$ - the filtered output feature map

The m^{th} filter in the kernel K is applied to the m^{th} channel in the input feature map F to produce the mth channel of the filtered output feature map G.

The computational cost of the depthwise convolutional layer is given in equation 7.1.4

$$D_k * D_k * M * D_F * D_F \qquad \text{eq. 7.1.4}$$

The depthwise convolution only filters the input channel but does not combine them. The pointwise convolution combines them to create a new feature. The pointwise convolution layer computes a linear combination of the outputs of the depthwise convolutional layer. The pointwise convolutional layer uses 1 x 1 convolution to generate the new features. This combination of depthwise convolution and pointwise convolution is called depthwise separable convolution. The computational cost of depthwise separable convolution is the sum of the computational cost of the depth-wise convolutional layer and the computational cost of the point-wise convolutional layer which is mathematically given below equation 7.1.5.

$$D_k * D_k * M * D_F * D_F + M * N * D_F * D_F \qquad \text{eq. 7.1.5}$$

The reduction of the computational cost of depthwise separable convolution and standard convolution is given in the below equation 7.1.6.

$$\frac{D_k * D_k * M * D_F * D_F + M * N * D_F * D_F}{D_k * D_k * M * N * D_F * D_F}$$

$$= \frac{1}{N} + \frac{1}{D_k^2} \qquad \text{eq. 7.1.6}$$

7.2 Inverted residuals

The bottleneck blocks appear similar to residual blocks. Each block contains an input followed by several bottlenecks then followed by expansion [36]. Here, there are shortcuts directly between the bottlenecks, inspired by the idea that the bottlenecks hold all of the relevant information, but the expansion layer is merely an implementation detail that comes with a non-linear tensor transformation [2].

The schematic visualization for the residual block and the inverted residual block is given in the below Figure 25. Improving the ability of a gradient to propagate across multiple layers is the motivation for inserting shortcuts. The shortcuts look similar to classical residual blocks. As compared to classical residuals, inverted residuals design is more memory efficient. The inverted residuals are slightly better in real-time.

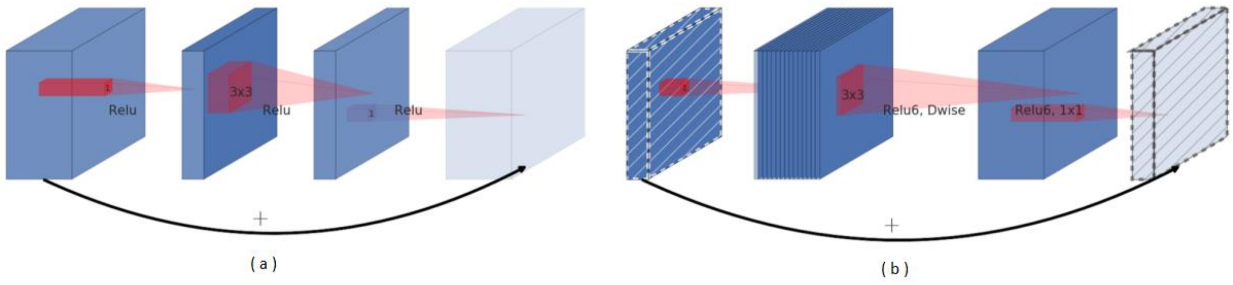


Figure 25 (a) Residual block (b) Inverted residual block [2]

7.2.1 Running time and parameter count for bottleneck convolution

Table 12 illustrates the basic implementation structure. Considering the block size is h x w, with the expansion factor t, input channel d', output channel d'' and the kernel size k. considering all the above the total number of multiply-add required is $h \times w \times d't(d' + k^2 + d'')$. This expression has an extra term compared to $h_i \times w_i \times d_i(k^2 + d_j)$. since we have an extra 1 x 1 convolution, but the structure of the networks allows us to use considerably smaller input and output dimensions.

Table 12 Bottleneck residual block [2].

Input	Operator	Output
$h \times w \times k$	1 x 1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3 x 3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	Linear 1 x 1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times \acute{k}$

7.3 Model Architecture

MobileNetV2 architecture is shown in Table 13. A bottleneck depth-separable convolution with residuals is the fundamental building block. The architecture contains initially fully convolution layers with 32 filters. The initial layers are followed by 19 residual bottle neck layers as shown in Table 13. Due to the robustness of ReLU6 when used with low-precision computation, it is used as the non-linearity. The kernel of size 3 x 3 is used as this is standard for modern networks. During training, this also uses the utilize dropout batch normalization [2].

Table 13 MobilNetV2 Architecture [2]

Input	Operator	t	c	n	s
$224^2 \times 3$	Conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	Conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	Avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	Conv2d 1x1	-	k	-	-

Here, a constant expansion rate across the network is implemented, except for the first layer. Expansion rates of 5 to 10 produce almost similar performance curves, with smaller networks performing better with smaller expansion rates and larger networks performing slightly better with larger expansion rates. Here, an expansion factor of 6 is applied to the size of the input tensor. For example, the bottleneck layer accepts a 64-channel input tensor and produces a tensor with 128 channels. So, 64 x 6 = 384 channels are in intermediate expansion layer.

7.4 Object Detection

As feature extractors, the performance of the MobileNetV2 and MobileNetV1 are compared and evaluated. For object detection, the MobileNet algorithms are compared. The MobileNet algorithm uses the SSD Lite version in the prediction layer. In SSDLite the regular convolutions are replaced with separable convolutions (depth-wise followed by 1 x 1 projection). This replacement makes the SSDLite a mobile-friendly variant. As compared to regular SSD the SSDLite has very less parameter count. In terms of computational cost also the SSDLite is more efficient than regular SSD. MobileNet algorithm with SSDLite in the prediction layer makes the overall design much more computationally efficient as shown in below Table 14 [5].

The fifteenth expansion layer of MobileNetV2 is used to attach the first layer of SSDLite. The fifteenth expansion layer gives the output with a stride of 16. The last layer of MobileNetV2 is with an output stride of 32. On the top of the last layers of MobileNetV2, the remaining layers of SSDLite are attached. The count of parameters and computational cost of different algorithms are given in Table 14. As compared to the MobileNetV1 + SSDLite has 5.1 million parameters to update and MobileNetV2 + SSDLite has 4.3 million parameters to update. The computational cost of MobileNetV1 + SSDLite and MobileNetV2 + SSDLite is 270ms and 200ms respectively to train an image. So, MobileNetV2 + SSDLite is considered for the mobile and real-time application because of its less parameters count and computational cost. The MobileNetV2 + SSDLite given effective training period and performance in object detection on Jetson Nano Nvidia GPU as compared to other algorithms.

Table 14 Comparision of MobileNet version [2]

Network	Params	MAdd	CPU
MobileNet v1 + SSDLite	5.1M	1.3B	270ms
MobileNet v2 + SSDLite	4.3M	0.8B	200ms

8. IMPLEMENTATION OF API

The methods in the API are implemented based on the SOLID principles. The end-user can modify the code easily based on the additional requirements without changing the existing code. The end-user can add the method for Watering<class>() if the new class is added in the training data. The additional modifications will not affect the existing code.

Some important methods implemented in AutomaticWatering.py are discussed below.

detectClassName():

```
def detectedClassName(Classes_list, Class):

    if any(Class in s for s in Classes_list):
        return Class

    else:
        return ''
```

The method detectedClassName is used to check the desired Class string is available in the array Classes_list detected by the object detection model. This method takes the Classes_list and Class as input arguments. The argument Classes_list contains the array of detected classes from the method run_model(). The argument Class is the string that will be checked whether the string is available in the Class_list array or not. If the Class string is available in the array Classes_list then the method returns string present in the variable Class. Else the method returns the null string.

readMoistTemp():

```
def readMoistTemp():
    i2c_bus = busio.I2C(SCL, SDA)
    ss = SensorSoil(i2c_bus, addr=0x36)
    Moist, Temp = ss.ssread()
    return Moist, Temp
```

The method readMoistTemp is used to calculate the Soil moisture and Temperature of the surrounding environment. This method returns the values of the soil moisture and temperature. The capacitive soil moisture sensor communicates with the I2c protocol. To establish the communication between capacitive soil moisture sensor and Jetson Nano the pin 3 SCL and pin 5 SDA are used as I2c bus. The method calls the sensor soil API with address

0x36. The capacitive soil moisture sensor I2c address is set to 0x36 as default. The soil moisture and temperature are read from the method ssread().

readWaterLevel():

```python
def readWaterLevel():
    i2c_bus = busio.I2C(SCL, SDA)
    ads = ADS1x151(i2c_bus)
    chan = ADC_I2c_output(ads, 2)
    ADC_WaterLevel = chan.value
    voltage_WaterLevel = chan.voltage
    return ADC_WaterLevel, voltage_WaterLevel
```

The method read water level is used to read the values of analog to the digital converted value of water level sensor and voltage readings of the water level sensor. The water level sensor is connected to ADS1115 IC. The ADS1115 IC converts the analog values of the water level sensor to digital values and gives the voltage readings of the sensor. The ADS1115 IC communicates with Jetson Nano through the I2c protocol. The ADDR pin of ADS1115 IC is connected to the ground to set the address to 0x48. This method calls the ADS1115 IC API to read the ADC value of the water level sensor and voltage readings. This method returns the ADC water level sensor readings and voltage readings of the water level sensor.

Watering():

```python
def Watering(className, Threshold,  SoilMoisture, SoilTemp, ADC_WaterLevel, voltage_WaterLevel):

    if className == 'Rose':
        WateringRose(Threshold, SoilMoisture, SoilTemp, ADC_WaterLevel, voltage_WaterLevel)

    if className == 'Sunflower':
        WateringSunflower(Threshold, SoilMoisture, SoilTemp, ADC_WaterLevel, voltage_WaterLevel)
```

The method Watering is called to call the appropriate method based on the detected class name from the object detection algorithm. This method takes the detected class name, the threshold value of soil moisture, Soil Moisture, Soil Temperature, ADC value of water level sensor, and voltage readings of water level sensor. Based on the input argument className example “Rose” the method WateringRose() method is called. If the new class is added, then the user just needs to add an if condition as shown in the above code. This method is structured in this way to make it easier to add additional classes in the future without disturbing the existing code which obeys the SOLID principles.

WateringRose():

```
def WateringRose(Threshold, SoilMoisture, SoilTemp, ADC_WaterLevel, voltage_WaterLevel):
```

The method WateringRose() is used to start and stop the motor pump based on the threshold value, soil moisture, and water level. The working of the WateringRose() function is shown in the flowchart Figure 26. This method takes the input arguments as Threshold, SoilMoisture, SoilTemparature, ADC_WaterLevel, and Voltage_WaterLevel. The program enters the while loop with the condition if the Soil moisture value is less than or equal to the threshold value. If the condition is not satisfied in the while loop it means the soil moisture is enough and watering is not required. So, it executes the command to turn off the motor pump and prints "I am happy a Plant". If the condition satisfies, the soil moisture value is less than the threshold value it means watering is required. Then it checks the condition if the soil moisture value is less than the threshold value again. If the condition soil moisture value is less than the threshold value is not satisfied, then it checks the else if condition soil moisture is greater than or equal to the threshold value. If the condition soil moisture value is less than the threshold value is satisfied, then it checks the condition if the water level in the tank is equal to true. If the water level is enough in the tank, then the method CheckWaterLevel returns true if not it returns false. If the water level in the tank is enough then the motor pump on condition will be executed and prints "Started Watering". After this, the program reads the soil moisture, soil temperature, ADC water level, voltage reading of the water level sensor and again checks the condition in the while loop. The loop will continue until the soil moisture value is greater than or equal to the threshold value. If the water level in the tank is equal to false, it means the water level in the tank is not enough, then the motor pump is stopped by printing "Not enough Water", "Please refill the water in the tank", "Cannot start watering". After again program reads the soil moisture, soil temperature, ADC water level, voltage reading of the water level sensor and again checks the condition in the while loop. The watering of the plant will resume once the water in the tank is refilled. The while loop continues the process until the soil moisture is greater than the threshold. The program also checks the water level in the tank after watering the plant, to ensure that the water level is enough for the next turn. This is how this method automatically checks the water level in the tank and soil moisture. Note: In the repot we discussed only WateringRose() method. Logic is the same for remaining plants as well. Based on the trained species the end-user has to add the same logic for the other.

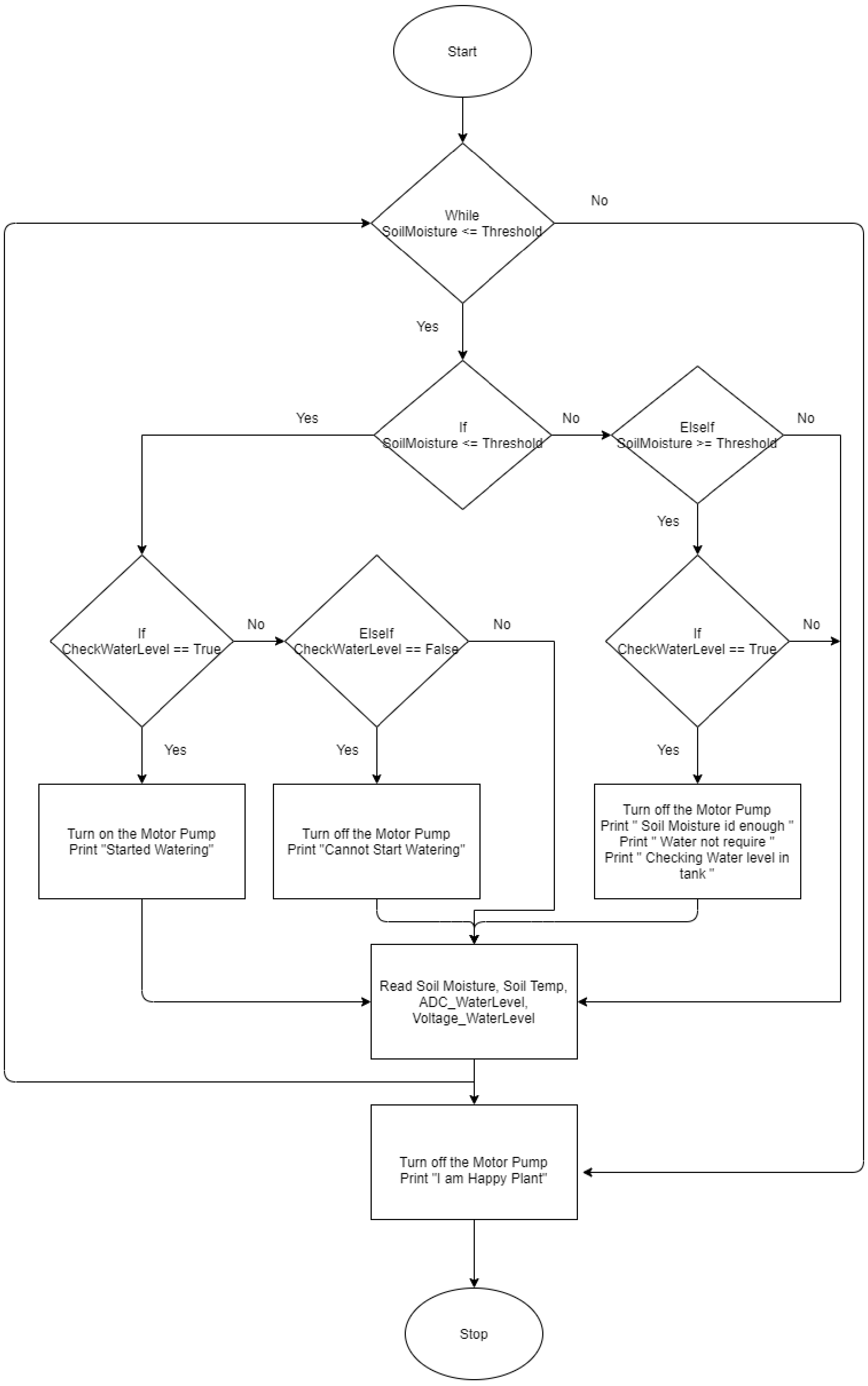

Start
While SoilMoisture <= Threshold
No
Yes
If SoilMoisture <= Threshold
Yes
No
ElseIf SoilMoisture >= Threshold
No
Yes
If CheckWaterLevel == True
No
ElseIf CheckWaterLevel == False
No
If CheckWaterLevel == True
No
Yes
Yes
Yes
Turn on the Motor Pump
Print "Started Watering"
Turn off the Motor Pump
Print "Cannot Start Watering"
Turn off the Motor Pump
Print " Soil Moisture id enough "
Print " Water not require "
Print " Checking Water level in tank "
Read Soil Moisture, Soil Temp, ADC_WaterLevel, Voltage_WaterLevel
Turn off the Motor Pump
Print "I am Happy Plant"
Stop

Figure 26 Flowchart WateringRose()

Some important methods implemented in my-detection.py are discussed below.

Load_model():

```
def load_model():


    net = jetson.inference.detectNet('ssd-mobilenet-v2',argv=['--model=/home/frost/jetson-inference/
python/training/detection/ssd/models/flowers/ssd-mobilenet.onnx','--labels=/home/frost/jetson-inference/
python/training/detection/ssd/models/flowers/labels.txt','--input-blob=input_0','--output-cvg=scores','--
output-bbox=boxes'],threshold=0.5)
    return net
```

This method is used to load the trained model. This method takes the model’s name, the path of the trained mode, path of the trained labels as arguments. The remaining arguments can be set default. This method returns the loaded model as a variable net.

Run_model():

Run_model method takes loaded network model “net” from Load_model() method as input argument. This method returns the list of classes or list of objects detected and the width of the class or object. The working of the Run_model() function is explained in the flowchart as shown in Figure 27. This method takes the video source from the USB camera connected, displays the live object detecting video with a bounding box in the new window. The image is a single frame. The continuous images or frames form a video. So, object detection with a live camera means continuous detection of objects in each frame in live video. The output of the object detection algorithm is around 30 frames per second. It means the algorithm can process around 30 images per second and gives the detected object of each frame.

The list of classes or list objects is noise-free. It means it does not contain the unexpected classes detected while running the algorithm. As because of the accuracy and other factors of the object detection model there is a chance of detecting some other class or object while running the algorithm. The occurrence of this will happen 5 to 15 times out of 100 times. This depends on the model train accuracy and model validation accuracy. To remove the unwanted detections the method checks 200 frames and gives the detected classes or objects. If the detected objects or classes are less than or equal to count 50 then it considers the detected object or class as unexpected detection or flaw detection and removes from the list of classes or list of objects detected. If the detected objects or classes are greater than count 50 then the detected object is added to the list of classes or list of objects detected and returns the list when the method is called.

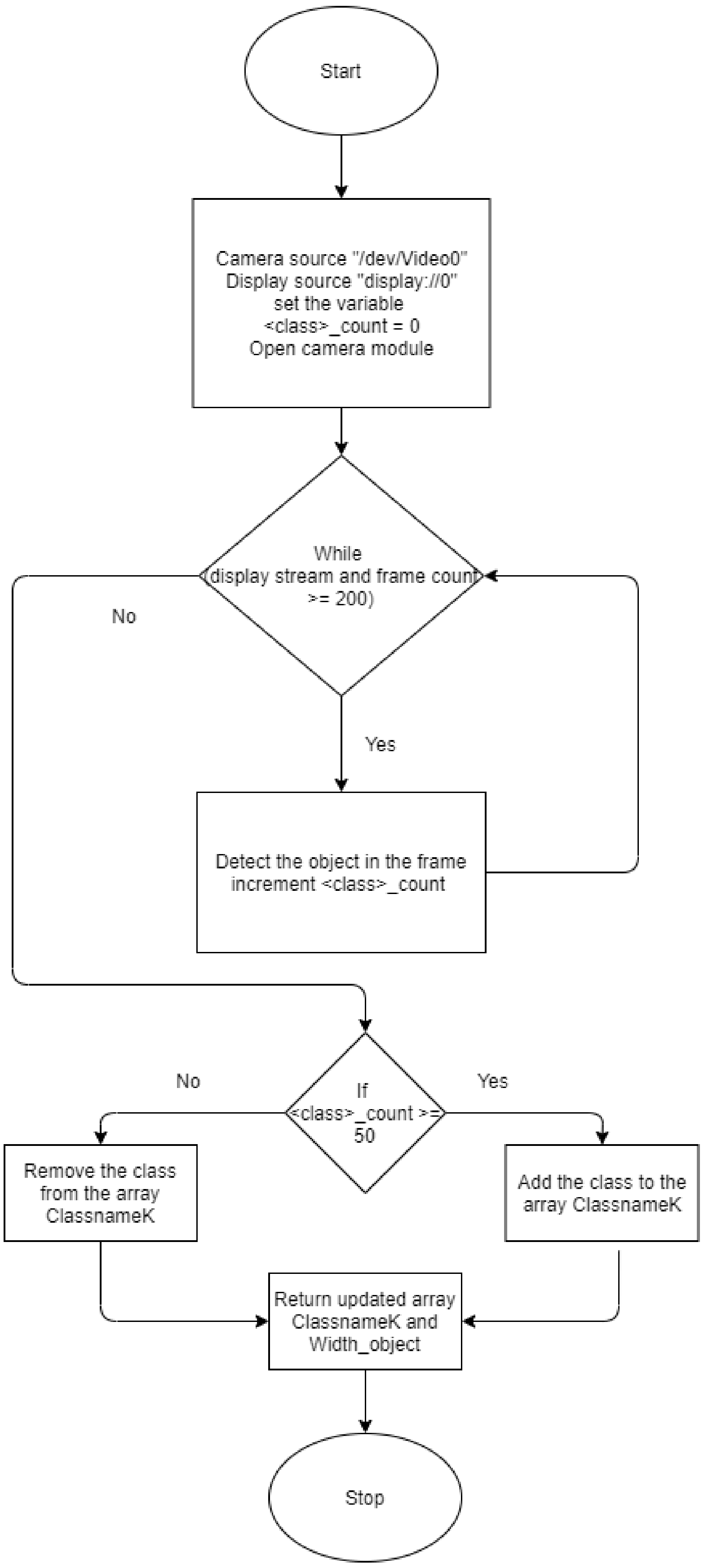

Start
Camera source "/dev/Video0"
Display source "display://0"
set the variable
<class>_count = 0
Open camera module
While
(display stream and frame count
>= 200)
No
Yes
Detect the object in the frame
increment <class>_count
If
<class>_count >=
50
No
Yes
Remove the class from the array ClassnameK
Add the class to the array ClassnameK
Return updated array ClassnameK and Width_object
Stop

Figure 27 Flowchart for Run_model()

Calibration_width():

```
def calibrate_width(net, width_pot_1, width_pot_2, distance_pot):
```

The water tank capacity should be dependent on the width of the flower pot. If the flower pot width is big the water tank capacity should be high. The flowchart for the Calibration_width function is shown in Figure 28. This method takes the input arguments of the load model, the width of the flower pot 1 and width of the flower pot 2, and the distance between the flower pot placed and the camera. For the current project, the width of the input of the flower pot 1 and flower pot 2 and the distance between the pot and camera are taken as 100 millimeters (10 centimeters) and 200 millimeters (20 centimeters), and 100 millimeters (10 centimeters) respectively and considering these values are standard. Calibration_width() function returns the width of the flower pot approximately. The calibration process takes place in two steps. In the first step, keep pot 1 with a standard width of ten centimeters at a distance of 10 centimeters and ask to wait for 15 seconds. At this time the object detection algorithm detects the flower pot of each frame. It takes 200 frames and calculates the number of pixels covered by the flower pot. The average number of pixels covered by the flower pot 1 for 200 frames is taken. The number of pixels occupied per one milli meter by the flower pot 1 in real-time is the average width covered by flower pot 1 divided by the ideal width of the flower pot 1. The same procedure is repeated for flower pot 2 with the standard width of 20 centimeters and kept at the standard distance of 10 centimeters. The object detection algorithm is and detection process is done for 200 frames. The average number of pixels covered by the flower pot 2 for 200 frames is calculated. The number of pixels occupied per one milli meter by the flower pot 2 in real-time is given by the average width covered by flower pot 2 divided by the ideal width of the flower pot 2. The calibrated average pixels occupied per 1 millimeter is given by the sum of the average number of pixels covered in flower pot 1 and the average number of pixels covered in flower pot 2 divided by 2. The approximate width of the flower pot in real-time is calculated by multiplying calibrated average pixels occupied per 1 milli meter and width of the flower pot in real-time.

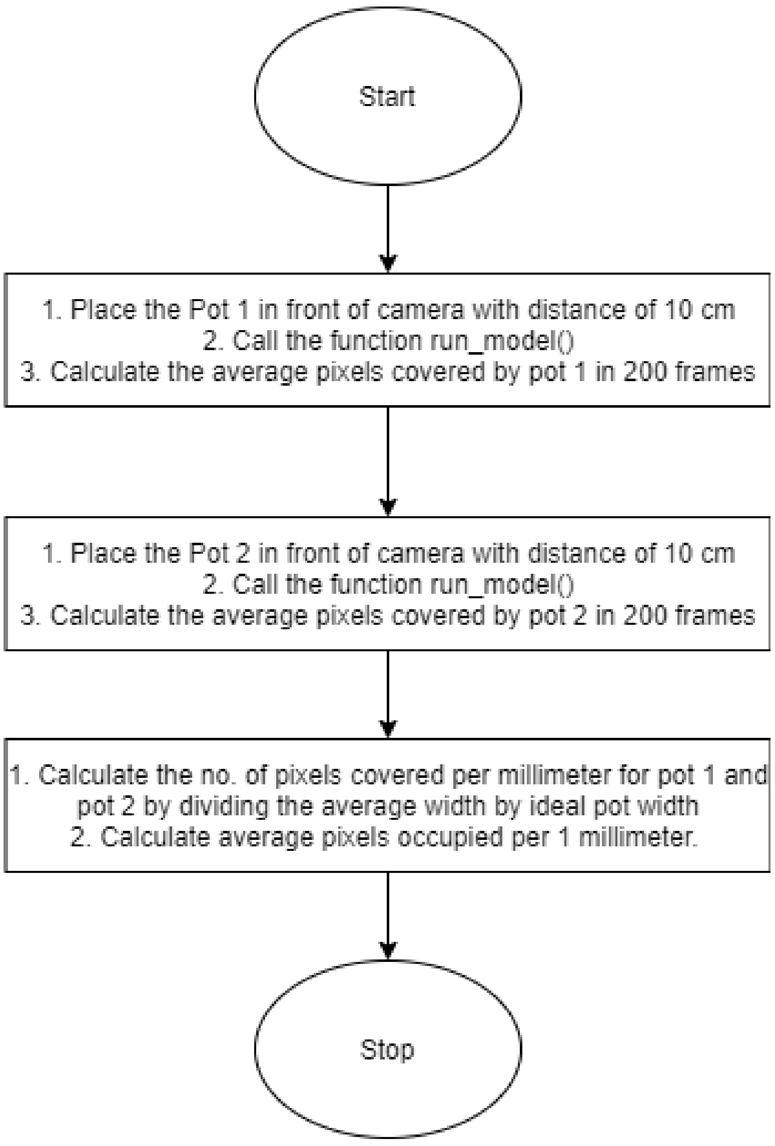

Start
1. Place the Pot 1 in front of camera with distance of 10 cm
2. Call the function run_model()
3. Calculate the average pixels covered by pot 1 in 200 frames
1. Place the Pot 2 in front of camera with distance of 10 cm
2. Call the function run_model()
3. Calculate the average pixels covered by pot 2 in 200 frames
1. Calculate the no. of pixels covered per millimeter for pot 1 and pot 2 by dividing the average width by ideal pot width
2. Calculate average pixels occupied per 1 millimeter.
Stop

Figure 28 Flowchart for Calibrate_width()

9. Implementation of Working Model

9.1 Block Diagram of Model

The block diagram of the complete model is explained in the below Figure 29. The block diagram explains the connections of each block and the flow of data based on the arrow indications. The first and foremost important block is Jetson Nano GPU. The Jetson Nano is connected to the USB camera and Add On PCB. The Jetson Nano has a bidirectional communication setup with Add On PCB to receive sensor data and to transmit control instructions to a 12V DC Motor pump. The Jetson Nano receives the frames from the USB camera and analyzes each frame by running an object detection algorithm whether the desired object is available in each frame or not. The second important block is Add On PCB. The Add On PCB communicates with Jetson Nano to transfer sensor data and receives instructions to control the motor pump. The Add On PCB consists of an L293D motor driver, ADS1115 IC. The Add On PCB receives the soil moisture value and water level value in the tank from the Capacitive soil moisture sensor and resistive type water level sensor respectively. The Capacitive soil moisture sensor is kept inserted into the soil in the pot to read the soil moisture content. The soil moisture values are then transmitted to Jetson Nano through Add On PCB through I2C communication. Similarly, the resistive type water levels sensor is attached to the walls of the tank. Based on the water level in the tank, the water level sensor reads the values and sends them to the Jetson Nano through Add On PCB. The water level sensor values are given to the ADC converter and then the ADC values and voltage values are transmitted to Jetson Nano through I2C communication.

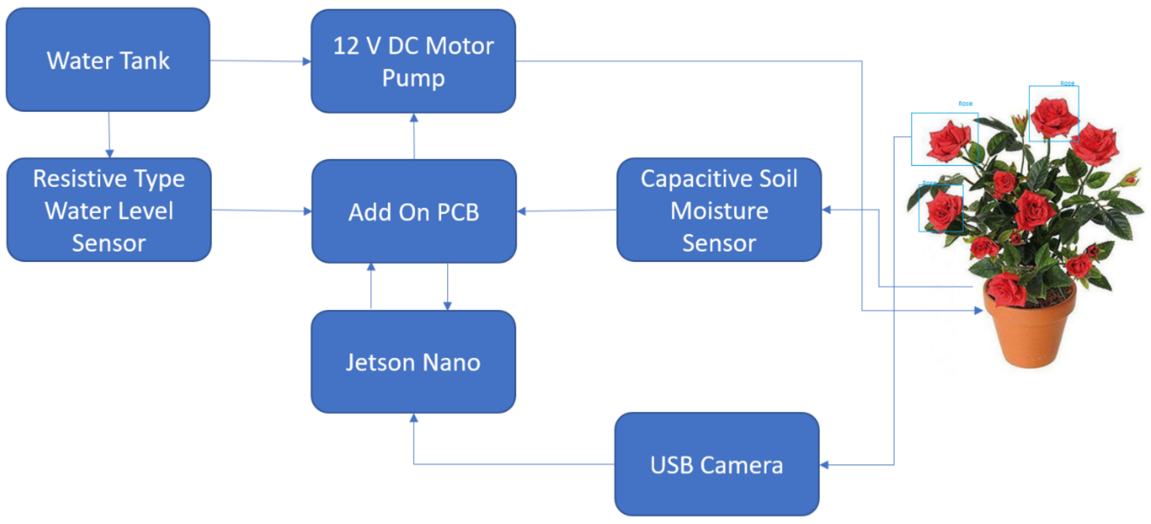

Water Tank
12 V DC Motor Pump
Resistive Type Water Level Sensor
Add On PCB
Capacitive Soil Moisture Sensor
Jetson Nano
USB Camera
Rose

Figure 29 Block Diagram of the Model

9.2 Working of the Model

Import the Automatic Watering System AWS API and necessary functions from the AWS API. The flowchart of the working model is shown in Figure 30. Open while loop with input condition true. Initialize the variable "net" and assign the function load_model(). The return values of load_model() will be assigned to the variable "net". Initialize the variables "Classes_list"," width". Assign the function run_model() to the variable "Classes_list"," width". The return values of function run_model() will be stored in the variables "Classes_list"," width". Check whether the Classes_list array consists of desired class or the desired flower plant name. If the desired flower plant is available in the Classes_list variable, then it prints the desired flower plant and reads the current soil moisture value and soil temperature values by calling the function readMoistTemp() and assigns the values in variables SoilMoisure and SoilTemp respectively. Now read the current values of water level in the tank by calling the function readWaterLevel() and assign the values in the variables ADC_WaterLevel and voltage_WaterLevel respectively. Call the Watering() function to start the automatic watering control. The input arguments of the function Watering() are className, the threshold value of the soil moisture content for the specific flower plant, SoilMoisture, SoilTemp, ADC_WaterLevel, and voltage_WaterLevel. Based on the input arguments the watering of the plant starts and main the soil moisture content based on the schedule. To schedule the next detection, watering, and checking the moisture content of soil the function

ScheduleNextWatering() is called. The input value of the function ScheduleNextWatering() is the time in minutes, after how many minutes the process should start again.

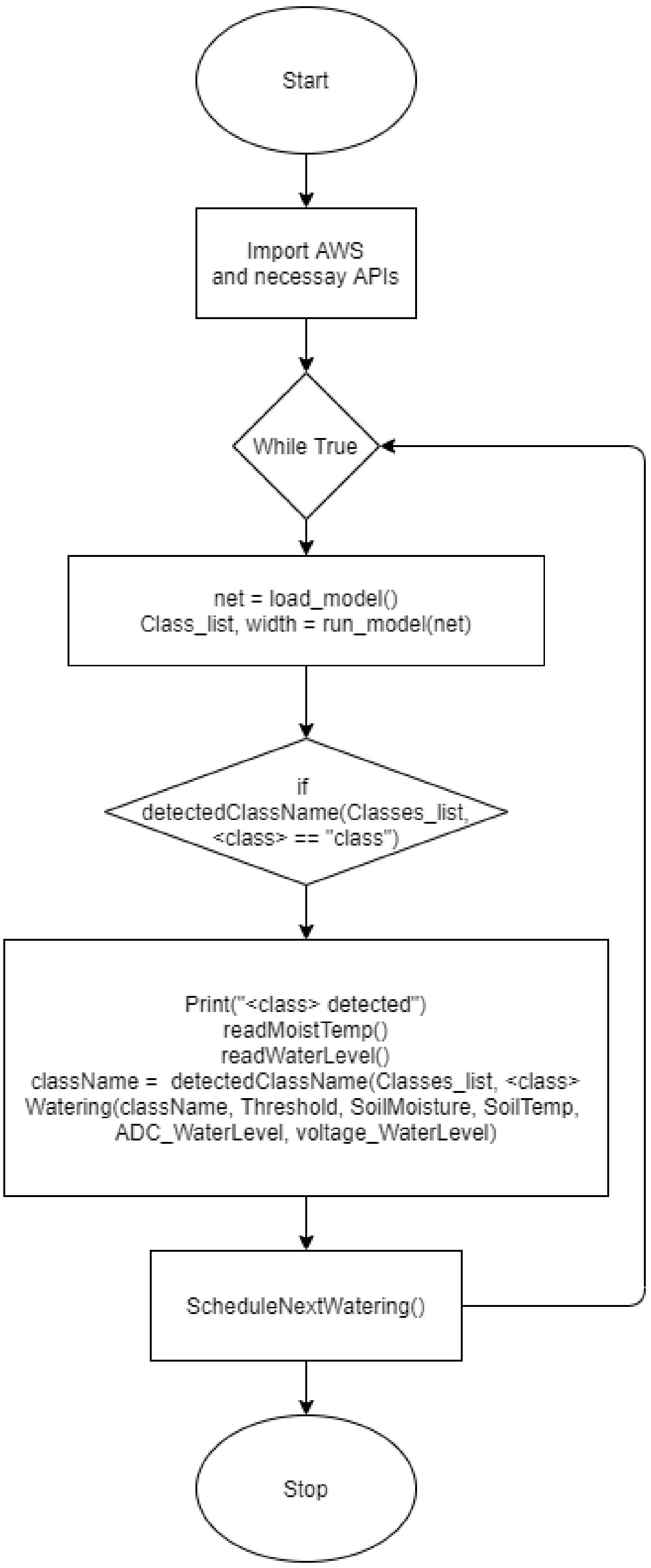

Start
Import AWS and necessay APIs
While True
net = load_model()
Class_list, width = run_model(net)
if detectedClassName(Classes_list, <class> == "class")
Print("<class> detected")
readMoistTemp()
readWaterLevel()
className = detectedClassName(Classes_list, <class>
Watering(className, Threshold, SoilMoisture, SoilTemp, ADC_WaterLevel, voltage_WaterLevel)
ScheduleNextWatering()
Stop

Figure 30 Flowchart for Working Model

9.3 Validation / Testing of Working Model

In the section, the testing of all functions implemented in API is discussed.

Test case 1:

TEST ID: AWS_TEST_ ID_001

Test Criteria: Testing the trained model detecting plant features properly.

Input: check Input arguments to run_model() are correct (path to the model and path to the label). Connect the USB camera. Run the python script Test_AWS by uncommenting code in test case 1 with test Id AWS_TEST_ ID_001.

1. Put the Rose flower in front of the camera.
2. Put the Sunflower in front of the camera

Output: Expected outputs are as below

1. Rose flower detected.
2. Sunflower detected.

Test case 2:

TEST ID: AWS_TEST_ ID_002

Test description: Checking the Soil moisture sensor reading the values correctly.

Input: Connect the soil moisture sensor to PCB properly. Connect the PCB to Jetson Nano. Run the python script Test_AWS by uncommenting code in test case 2 with test Id AWS_TEST_ ID_002.

1. Put the soil moisture sensor in wet soil.
2. Put the soil moisture sensor in dry soil

Output: Expected outputs are as below

1. The sensor reading should be above 700.
2. The sensor reading should be below 400.

Test case 3:

TEST ID: AWS_TEST_ ID_003

Test description: Checking the Water level sensor reading the values correctly.

Input: Connect the Water level sensor to PCB properly. Connect the PCB to Jetson Nano. Run the python script Test_AWS by uncommenting code in test case 3 with test Id AWS_TEST_ ID_003.

1. Put the Water level sensor in the tank with full water.
2. Put the Water level sensor in the tank with half water.
3. Put the Water level sensor in the tank with less water.

Output: Expected outputs are as below

1. The sensor voltage reading should be above 1.8.
2. The sensor voltage reading should be above 1.3.
3. The sensor voltage reading should be above 0.5 to 1.1.

Test case 4:

TEST ID: AWS_TEST_ ID_004

Test description: Checking the Water level sensor reading the values correctly.

Input: Connect the Motor pump to PCB terminal screws properly. Connect the PCB to Jetson Nano. Run the python script Test_AWS by uncommenting code in test case 3 with test Id AWS_TEST_ ID_004.

Output: Expected outputs are as below

1. The motor pump should be on for 10 seconds and off for 10 seconds.

Test case 5:

TEST ID: AWS_TEST_ ID_005

Test description: Checking the model behavior in different conditions.

Input: Connect all sensors and the motor pump to the PCB. Connect the PCB to Jetson Nano. Check the capacitive soil moisture sensor is kept in the soil and check the water level sensor is in the water tank properly. Run the python script Test_AWS by uncommenting code in test case 5 with test Id AWS_TEST_ ID_005.

1. Put Rose plant in front of the camera.
2. Put the soil moisture sensor in dry soil.
3. Put the water level sensor in the full tank.

Output: Expected outputs are as below

1. Rose flower detected
2. The motor pump starts.
3. The motor should be on until the soil moisture sensor value is above 700 and then the motor pump stops.

Test case 6:

TEST ID: AWS_TEST_ ID_006

Test description: Checking the model behavior in different conditions.

Input: Connect all sensors and motor pump to the PCB. Connect the PCB to Jetson Nano. Check the capacitive soil moisture sensor is kept in the soil and check the water level sensor is in the water tank properly. Run the python script Test_AWS by uncommenting code in test case 6 with test Id AWS_TEST_ ID_006.

1. Put Rose plant in front of the camera.
2. Put the soil moisture sensor in completely wet soil.
3. Put the water level sensor in the full tank.

Output: Expected outputs are as below

1. Rose flower detected.
2. The motor pump does not start
3. The soil moisture sensor value is above 700.
4. Prints “I am a Happy Plant” and also prints “Enough water in the tank for next schedule” or “Not enough water in the tank for next schedule” based on the water level in the tank.

Test case 7:

TEST ID: AWS_TEST_ ID_007

Test description: Checking the model behavior in different conditions.

Input: Connect all sensors and the motor pump to the PCB. Connect the PCB to Jetson Nano. Check the capacitive soil moisture sensor is kept in the soil and check the water level sensor is in the water tank properly. Run the python script Test_AWS by uncommenting code in test case 7 with test Id AWS_TEST_ ID_007.

1. Put Rose plant in front of the camera.
2. Put the soil moisture sensor in completely wet soil.
3. Put the water level sensor in the half tank.

Output: Expected outputs are as below

1. Rose flower detected.
2. The motor pump does not start
3. The soil moisture sensor value is above 700.
4. Prints “I am a Happy Plant” and “Please refill the water in the tank” and “Not enough water in the tank for next schedule”.

Test case 8:

TEST ID: AWS_TEST_ ID_008

Test description: Checking the model behavior in different conditions.

Input: Connect all sensors and the motor pump to the PCB. Connect the PCB to Jetson Nano. Check the capacitive soil moisture sensor is kept in the soil and check the water level sensor is in the water tank properly. Run the python script Test_AWS by uncommenting code in test case 8 with test Id AWS_TEST_ID_008.

1. Put Rose plant in front of the camera.
2. Put the soil moisture sensor in completely dry soil.
3. Put the water level sensor in the empty tank.
4. After 3 – 4 minutes refill the tank with water.

Output: Expected outputs are as below

1. Rose flower detected.
2. Prints “Please refill the water in the tank”.
3. Prints “Cannot start watering”.
4. The motor pump does not start.
5. Waits until the water in the tank is refiled.
6. After refilling the water in the tank, the motor pump starts watering.
7. The soil moisture sensor value is above 700.
8. Prints “I am a Happy Plant” and also prints “Enough water in the tank for next schedule” or “Not enough water in the tank for next schedule” based on the water level in the tank.

All the test results are tabulated as shown in Table 15.

Table 15 Validation / Testing results

S.No	Test Case	Test ID	Pass / Fail
1	Test case 1	AWS_TEST_ID_001	Pass
2	Test case 2	AWS_TEST_ID_002	Pass
3	Test case 3	AWS_TEST_ ID_003	Pass
4	Test case 4	AWS_TEST_ ID_004	Pass
5	Test case 5	AWS_TEST_ ID_005	Pass
6	Test case 6	AWS_TEST_ ID_006	Pass
7	Test case 7	AWS_TEST_ ID_007	Pass
8	Test case 8	AWS_TEST_ ID_008	Pass

10. Implementation of Working Model

10.1 Experimental Setup:

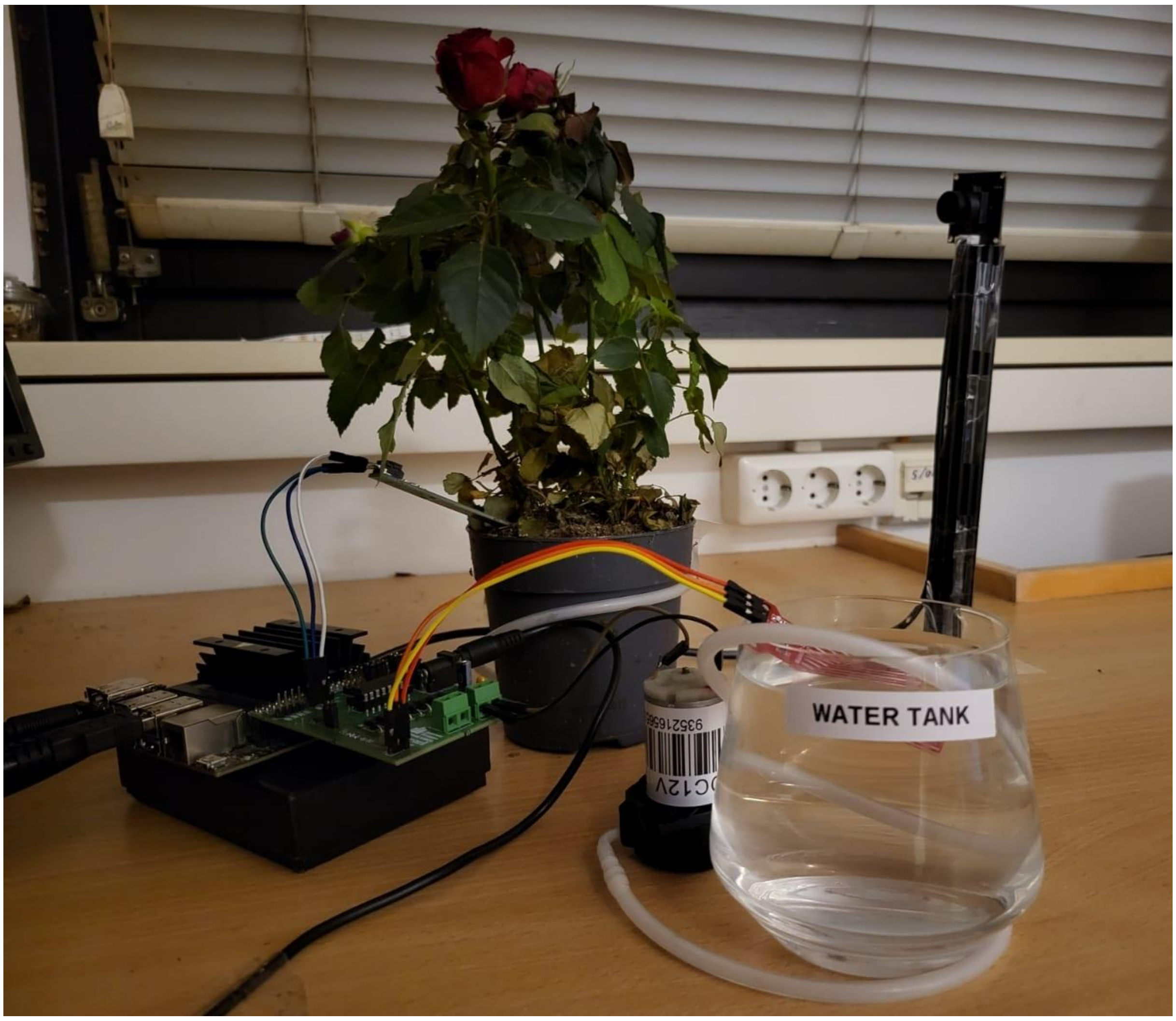
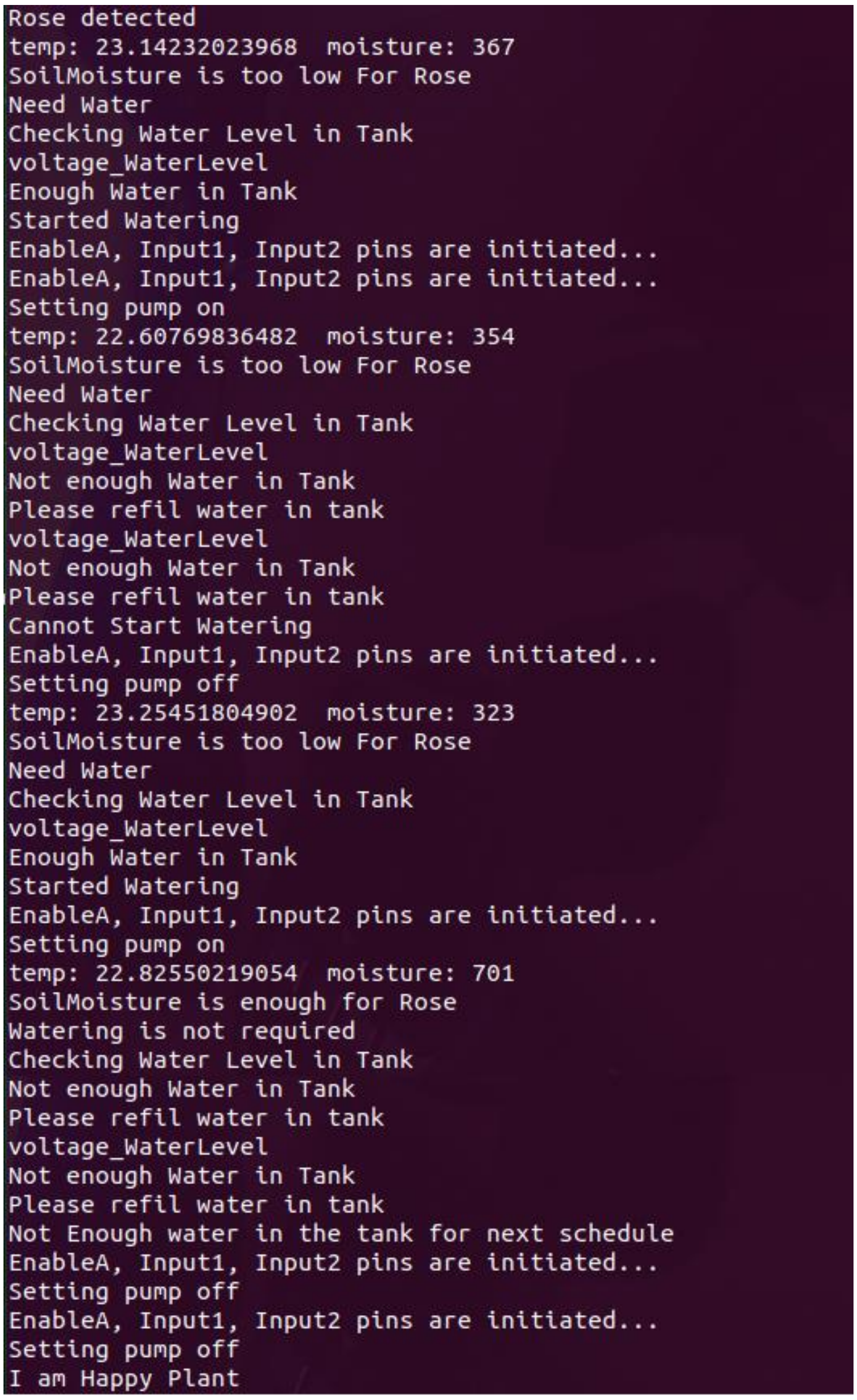


Figure 31 Experimental Setup

Attach the add-on PCB to Jetson Nano and attach all the sensors, the motor pump to the add-on PCB. Keep the water level sensor in the water tank and insert the soil moisture sensor in the pot. Put one pipe of the motor pump in the water tank and the other pipe in the flowerpot as shown in Figure 31.

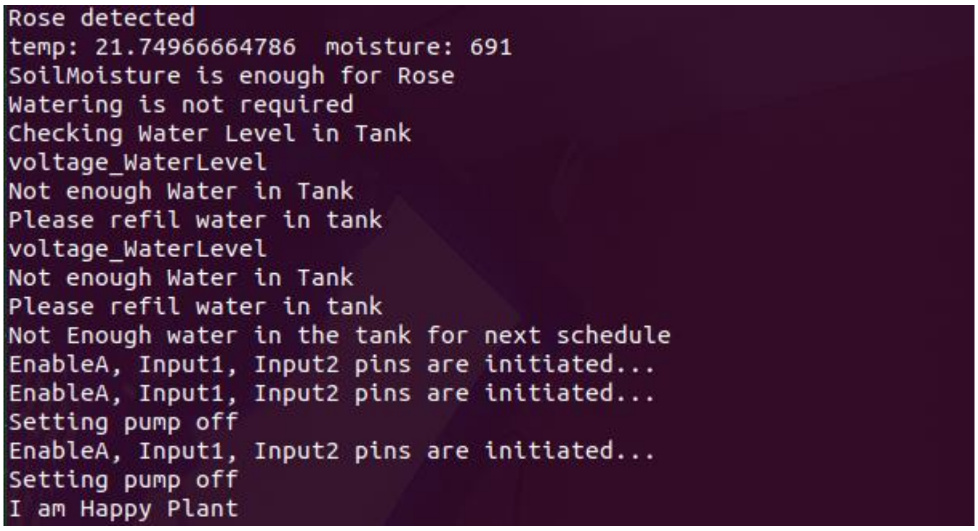
Run the python script in the terminal, first it starts with plant detection and prints the detected plant as shown in Figure 32. Then it starts checking the water level in the tank and soil moisture content. Figure 32 is the output screenshot with the initial condition with very little water in the tank and less soil moisture content. As the soil moisture content is less the model starts the watering but the water in the tank is sufficient to water. Once the water in the tank is finished the watering stops and waits until the water is refilled in the tank. The algorithm prints "Not enough water in the Tank" and "Please refill the water in the Tank" So, the algorithm waits until the water is refilled in the tank. Once the water is refilled, watering is started by turning on the motor pump. Once the soil moisture content meets the threshold value the motor pump stops and prints "I am a Happy Plant" and checks the water level in the tank. If the water level is not sufficient then prints "Not enough water in the tank for next schedule" and wait for the next scheduled watering.



```
Rose detected
temp: 23.14232023968  moisture: 367
SoilMoisture is too low For Rose
Need Water
Checking Water Level in Tank
voltage_WaterLevel
Enough Water in Tank
Started Watering
EnableA, Input1, Input2 pins are initiated...
EnableA, Input1, Input2 pins are initiated...
Setting pump on
temp: 22.60769836482  moisture: 354
SoilMoisture is too low For Rose
Need Water
Checking Water Level in Tank
voltage_WaterLevel
Not enough Water in Tank
Please refil water in tank
voltage_WaterLevel
Not enough Water in Tank
Please refil water in tank
Cannot Start Watering
EnableA, Input1, Input2 pins are initiated...
Setting pump off
temp: 23.25451804902  moisture: 323
SoilMoisture is too low For Rose
Need Water
Checking Water Level in Tank
voltage_WaterLevel
Enough Water in Tank
Started Watering
EnableA, Input1, Input2 pins are initiated...
Setting pump on
temp: 22.82550219054  moisture: 701
SoilMoisture is enough for Rose
Watering is not required
Checking Water Level in Tank
Not enough Water in Tank
Please refil water in tank
voltage_WaterLevel
Not enough Water in Tank
Please refil water in tank
Not Enough water in the tank for next schedule
EnableA, Input1, Input2 pins are initiated...
Setting pump off
EnableA, Input1, Input2 pins are initiated...
Setting pump off
I am Happy Plant
```

Figure 32 Output screenshot with initial values of soil moisture is less than the threshold and water level is medium

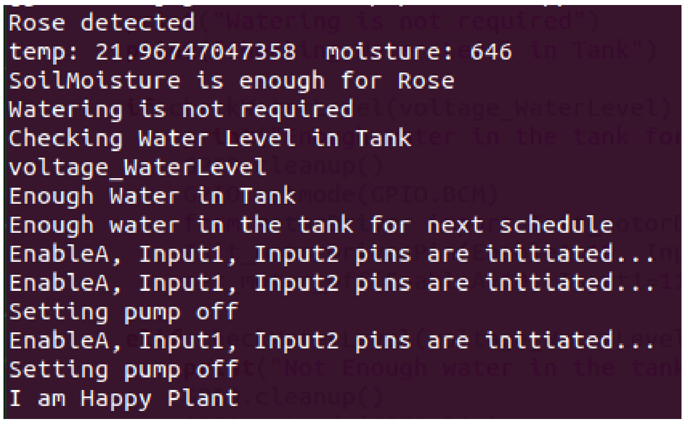
In the next schedule, it starts detecting the plant again. This time the soil moisture content is enough more than the threshold value. So, the model prints “I am Happy Plant” and checks the water level in the tank. If the water level is less than the threshold level then it prints “Please refill water in the tank”, “Not enough water for the next scheduled” waits for the next Schedule as shown in Figure 33.



```
Rose detected
temp: 21.7496664786  moisture: 691
SoilMoisture is enough for Rose
Watering is not required
Checking Water Level in Tank
voltage_WaterLevel
Not enough Water in Tank
Please refil water in tank
voltage_WaterLevel
Not enough Water in Tank
Please refil water in tank
Not Enough water in the tank for next schedule
EnableA, Input1, Input2 pins are initiated...
EnableA, Input1, Input2 pins are initiated...
Setting pump off
EnableA, Input1, Input2 pins are initiated...
Setting pump off
I am Happy Plant
```

Figure 33 Output screenshot with Soil moisture is enough but the water in the tank is less for the next schedule

If in the next schedule again the soil moisture content is enough for the detected plant. But the water level is greater than the threshold level then it prints "Enough water in the tank", "Enough water for the next schedule" waits for the next Schedule as shown in Figure 34.



```
Rose detected
temp: 21.96747047358  moisture: 646
SoilMoisture is enough for Rose
Watering is not required
Checking Water Level in Tank
voltage_WaterLevel
Enough Water in Tank
Enough water in the tank for next schedule
EnableA, Input1, Input2 pins are initiated...
EnableA, Input1, Input2 pins are initiated...
Setting pump off
EnableA, Input1, Input2 pins are initiated...
Setting pump off
I am Happy Plant
```

Figure 34 Output screenshot with Soil moisture is enough but the water in the tank is enough for the next schedule

10.2 Comparison of the proposed model with existing models

The proposed model in the current project is compared with the existing models proposed in the papers Automatic plant watering system (APWS) and Smart watering plants (SWP) in Table 16. The board/microcontroller used in the proposed model is Jetson Nano which is advanced than then Arduino which is used in APWS and SWP. Jetson Nano is capable of running complex algorithms such as object detection. The capacitive soil moisture sensor used in the proposed model will not undergo corrosion whereas the resistive type will undergo corrosion. Add on

PCB designed in the proposed model can communicate with a capacitive soil moisture sensor, water level sensor, and motor pump whereas PCB proposed in APWS can only communicate between Arduino and motor pump. No PCB is used in the SWP model. The special feature that differs the proposed model from other models is plant type detection which doesn't exist in other models. The proposed model makes the decision of watering to plant, based on plant type and soil moisture content.

Table 16 Comparison of the proposed model with existing models

Model	Board/Microcontroller	Soil Moisture sensor	Water level sensor	Add on PCB	Plant type detection
Proposed Model	Jetson Nano	Capacitive Soil Moisture sensor	Resistive type Water level sensor	PCB can communicate with all sensors, motor pump, and Jetson Nano	Yes
Model proposed in the paper (APWS) [8]	Arduino	Resistive type Soil Moisture sensor	Resistive type Water level sensor	PCB communicates with Arduino and water pump	No
Model proposed in the paper (SWP) [9]	Arduino	Resistive type Soil Moisture sensor	Resistive type Water level sensor	No PCB is proposed	No

Conclusion

The proposed model successfully detected the trained plant and values of soil moisture sensor and water level sensor from the add-on PCB. The motor is controlled properly based on the moisture content of the soil and water level in the tank. The add-on PCB is successfully communicating with the Jetson Nano development kit. The developed API for object detection is successfully eliminating the false detections over two hundred frames. Finally, all the functions in the API are communicating properly with the prototype and working in all types of soil conditions in any country. Compared the proposed model with the existing models proposed in papers Automatic plant watering system (APWS) and Smart watering plants (SWP) and tabulated the difference.

Future Scope

The calibration_width() function can be properly planned to use whenever the plant and pots are changed without resetting the algorithm. The capacitive soil moisture sensor also reads temperature values which can be used to make the model can work in any climatic conditions. This model with slight modifications and improvements in soil analysis sensors can be used to monitor the plant growth on the mars not only earth which is the vision of this idea.

BIBLIOGRAPHY

[1] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks

[3] Tejasv Agarwal, Himanshu Mittal, Performance comparison of deep neural networks on image datasets

[4] Ahmet Ali Süzen, Burhan Duman, Betül Şen, Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN

[5] Manisha Mayuree, Priyanka Aishwarya, Bagubali A, Automatic Plant Watering System, 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN) 978-1-5386-9353-7/19/$31.00 ©2019 IEEE

[6] Kotni.Naga Siva, Raj Kumar.G, A.Bagubali, Kishore V Krishnan, Smart watering of plants, 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN) 978-1-5386-9353-7/19/$31.00 ©2019 IEEE

[7] T. Tomizawa, A. Ohya, S. Yuta, Remote Book Browsing System using a Mobile Manipulator. IEEE, 2003. Intl Conf. on Robotics and Automation

[8] Tetsuo Tomizawa, Akihisa Ohya and Shinichi Yuta, Remote Food Shopping Robot System in a Supermarket. IEEE, 2007.

[9] B. H. Kim, D. K. Roh, Jang M. Lee Localization of a Mobile Robot using Images of a Moving Target. IEEE, 2001.

[10] Thomas M. Mitchell. Machine Learning. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 9780070428072

[11] Jetson_Nano_Developer_Kit_User_Guide.pdf

[12] ads1115.pdf

[13] https://www.adafruit.com/product/1150

[14] https://docs.kicad.org/master/en/getting_started_in_kicad/getting_started_in_kicad.pdf

[15] https://aisler.net/help/design-rules-and-specifications/design-rules

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhad , “You Only Look Once: Unified, Real-Time Object Detection” https://arxiv.org/pdf/1506.02640.pdf

[17] N. Ketkar, “Deep Learning with Python”, DOI 10.1007/978-1-4842-2766-4_5

[18] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[19] https://towardsdatascience.com/derivation-of-convolutional-neural-network-from-fully-connected-network-step-by-step-b42ebafa5275

[20] Kota Ando, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, Masato Motomura,”A Multithreaded CGRA for Convolutional Neural Network Processing”, 2017.

[21] Yasaka K, Akai H, Abe O, Kiryu S ,”Deep learning with convolutional neural network for differentiation of liver masses at dynamic contrast-enhanced CT: a preliminary study. Radiology”, 2018.

[22] Lakhani P, Sundaram B,” Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks. Radiology”, 2017.

[23] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: “Largescale video classification with convolutional neural networks. In: Computer Vision and Pattern Recognition (CVPR)”, IEEE Conference on. pp. 1725–1732, 2014.

[24] Shuiwang Ji, Wei Xu , Ming Yang, Kai Yu, “3d convolutional neural networks for human action recognition. Pattern Analysis and Machine Intelligence”, IEEE Transactions on 35(1), 221–231 ,2013.

[25] Cires¸an, D.C., Meier, U., Gambardella, L.M., Schmidhuber, “Convolutional neural network committees for handwritten character classification. In: Document Analysis and Recognition” , International Conference on. pp. 1135–1139. IEEE, 2011.

[26] Szegedy, C., Toshev, A., Erhan, D, “Deep neural networks for object detection. In: Advances in Neural Information Processing Systems”, pp. 2553–2561 ,2013.

[27] https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-objectdetection/

[28] http://cs231n.github.io/convolutional-networks/

[29] Reagan L. Galvez , Argel A. Bandala, Elmer P. Dadios, Ryan Rhay P. Vicerra, Jose Martin Z. Maningo, “Object Detection Using Convolutional Neural Networks”, IEEE Region 10 Conference, Proceedings of TENCON 2018.

[30] https://lastminuteengineers.com/water-level-sensor-arduino-tutorial/

[31] Chaitanya P. Gharpure Vladimir A. Kulyukin Robot-assisted shopping for the blind: issues in spatial cognition and product selection Springer-Verlag, 2008.

[32] https://www.ti.com/lit/ds/symlink/l293.pdf

[33] https://learn.adafruit.com/adafruit-stemma-soil-sensor-i2c-capacitive-moisture-sensor/downloads

[34] https://www.seeedstudio.com/blog/2020/01/10/what-is-soil-moisture-sensor-and-simple-arduino-tutorial-to-get-started/

[35] https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Book in preparation for MIT Press, 2016. URL http://www.deeplearningbook.org.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385, 2015.

[38] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In David Blei and Francis Bach, editors, Proceedings of the 32nd International Conference on Machine Learning (ICML-15), volume 37, pages 448–456. JMLR Workshop and Conference Proceedings, 2015.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[40] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Neural Networks: Tricks of the Trade, pages 9–50. Springer-Verlag, 1998. ISBN 3-540-65311-2.

[41] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Lecun Y., Bengio Y., and Hinton G. Deep learning. Nature, 521(7553):436–444, 2015.

[42] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. CoRR, abs/1312.4400, 2013. URL http://arxiv.org/abs/1312.4400.

[43] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the ImageNet. CoRR, abs/1606.02228, 2016. URL http://arxiv.org/abs/1606.02228.

[44] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex A. Alemi. Inception-v4, InceptionResNet and the Impact of Residual Connections on Learning. In ICLR 2016 Workshop, 2016. URL https://arxiv.org/abs/1602.07261

[45] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. In ICLR (workshop track), 2015

[46] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR ’14, pages 1717–1724. IEEE Computer Society, 2014

[47] ”What is a plant” Science Learning, 18 October 2010 Available: https://www.sciencelearn.org.nz/resources/1102-what-is-a-plant

[48] ”What is in soil?” Science Learning, 30 June 2015 Available: https://www.sciencelearn.org.nz/resources/890-what-is-in-soil.

[49] Itructables.com. (2018). Arduino Soil Moisture Sensor: http://www.instructables.com/id/ArduinoSoil-Moisture-Sensor/

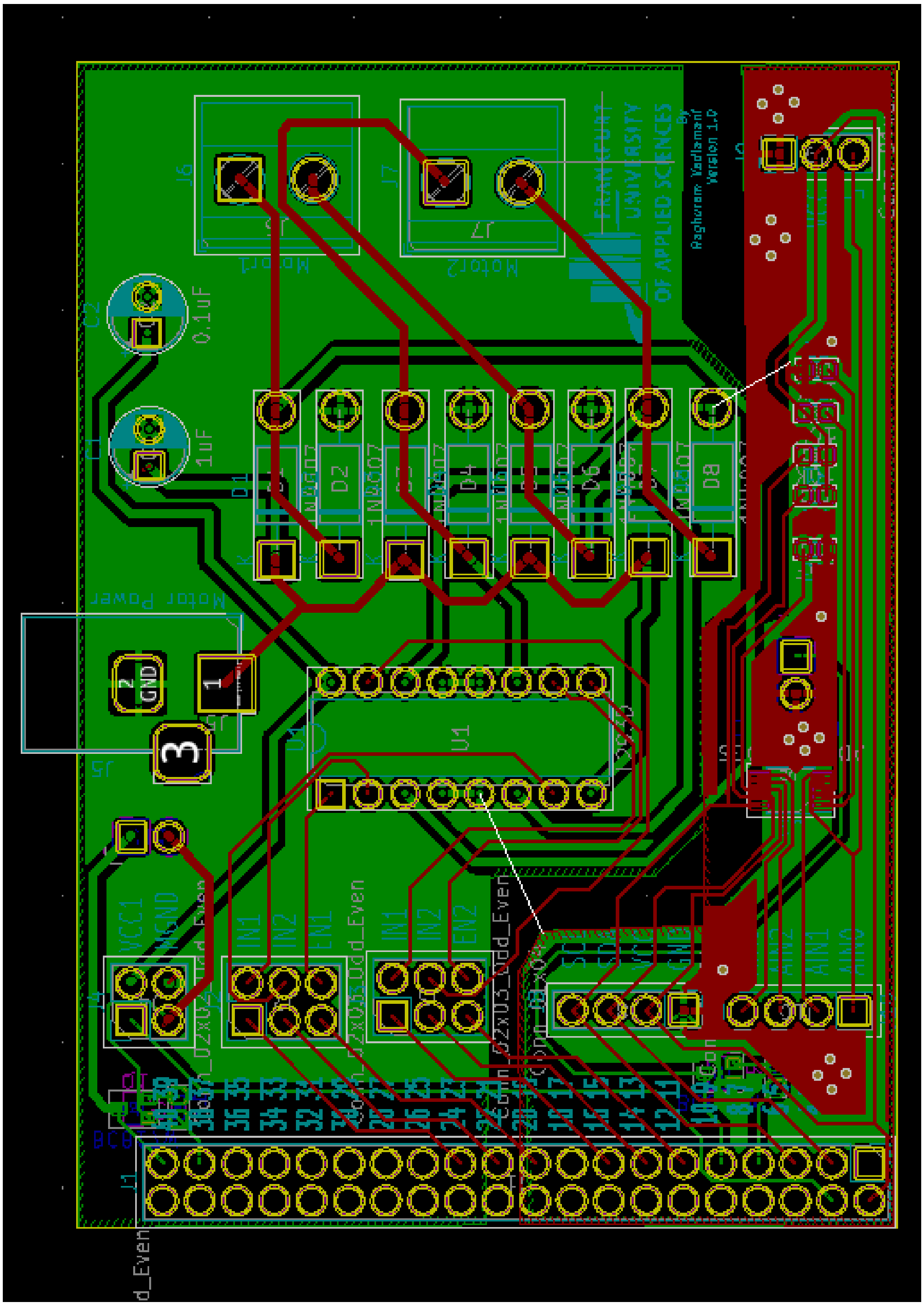
[50] Tina Bringslimark, Terry Hartig, Grete Grindal Patil ”Psychological Benefits of Indoor Plants in Workplaces: Putting Experimental Results into Context” Norwegian University of Life Sciences and Uppsala University, 46(5):744–752, 2011.

[51] Jane Dyrhauge Thomsen, Hans K.H. Søndenstrup-Anderssen, Renate M¨uller ”People–plant Relationships in an Office Workplace: Perceived Benefits for the Workplace and Employees” University of Copenhagen and Roskilde University, 46(5):744–752, 2011.

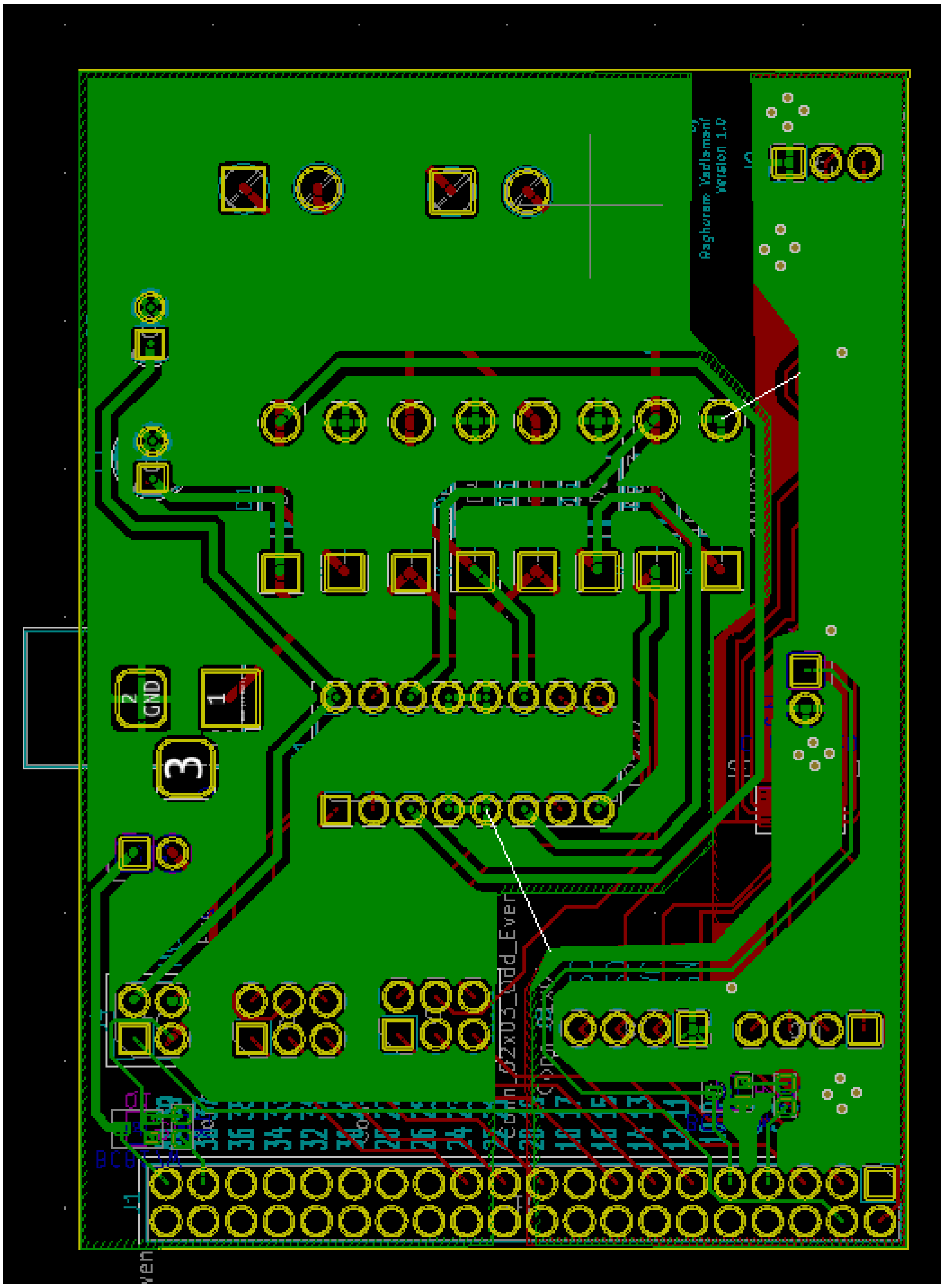
[52] Michael D. Dukes, Rafael Munos-Carpena, Herbert Bryan, Waldemar Klassen ”Automatic soil moisture-Based drip irrigation for improving tomato production” University of Florida , 116:80-85, 2003

APPENDICES

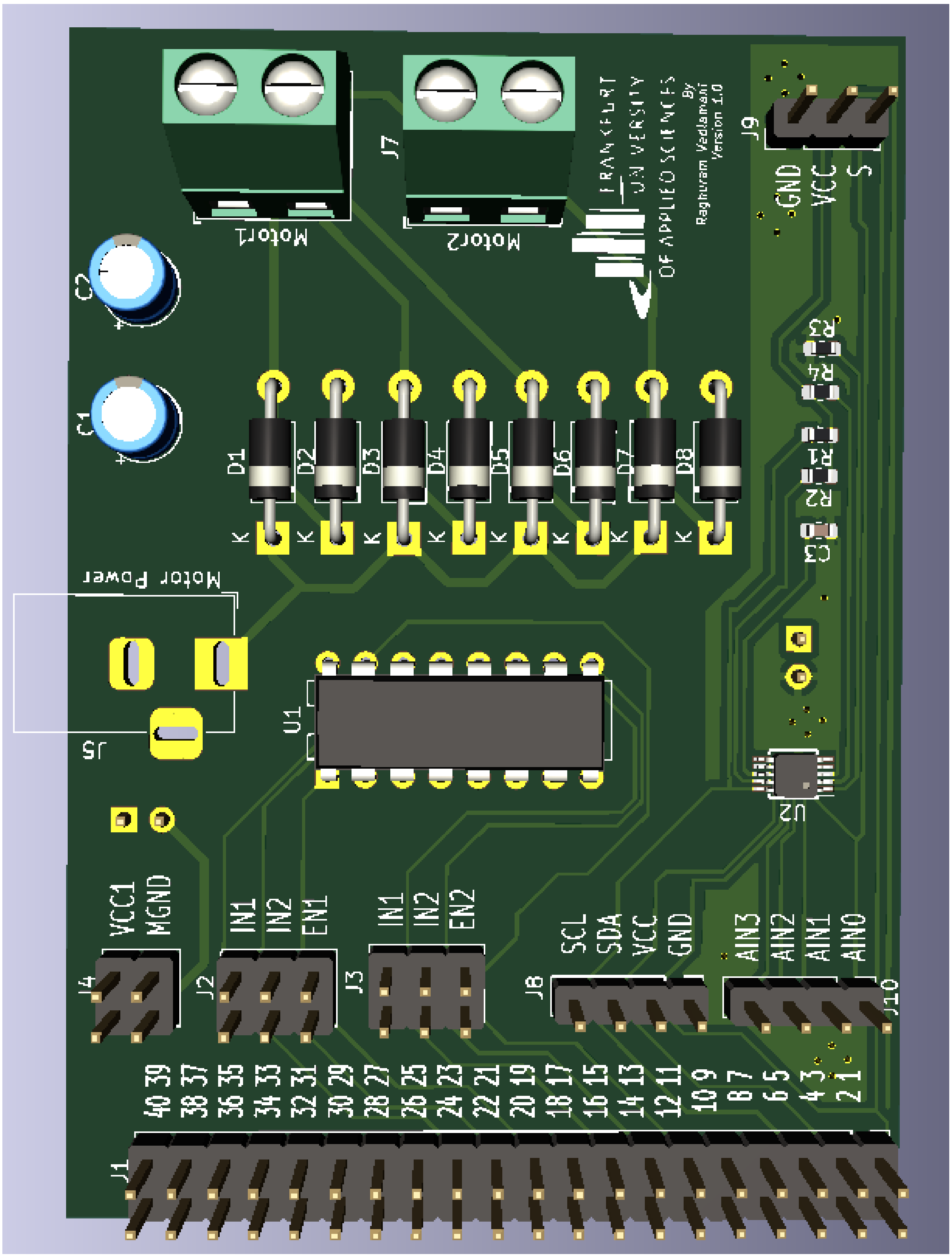
Appendix 1 – Add On PCB



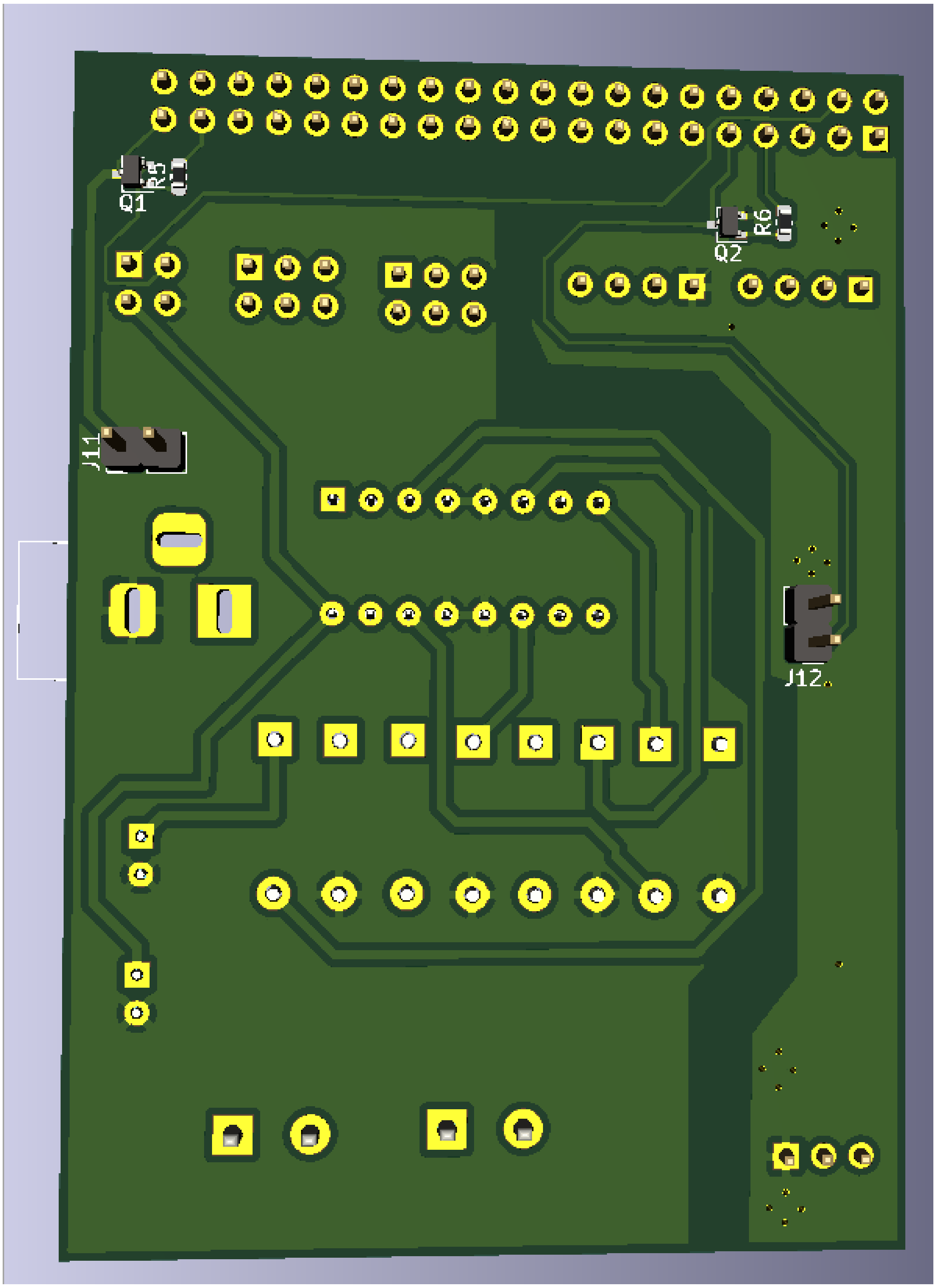
PCB View (Top plate of PCB)



PCB View (Bottom plate of PCB)



3D view of PCB (Top view)



3D view of PCB (Bottom view)