DESIGN OF LOW-COST OBJECT IDENTIFICATION MODULE FOR CULINARY APPLICATIONS

A PROJECT REPORT

Submitted by

VYSHNAVI MUPPIRALA [Reg No: RA1611004010067]
C.V PRANAV [Reg No: RA1611004010091]
REVANTH B [Reg No: RA1611004010237]

Under the guidance of

Mrs.A.ANILET BALA
(Professor, Department of Electronics and communication & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2020

SRM Institute of Science and technology

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**DESIGN OF LOW-COST OBJECT IDENTIFICATION MODULE FOR CULINARY APPLICATIONS**" is the bonafide work of " **VYSHNAVI MUPPIRALA [Reg No: RA1611004010067], C.V PRANAV [Reg No: RA1611004010091], REVANTH B [Reg No: RA1611004010237], ,** ", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs.A.ANILET BALA
GUIDE
Professor
Dept. of Electronics and communication & Engineering

SIGNATURE

Dr. T. Rama Rao
HEAD OF THE DEPARTMENT
Dept. of Electronics and communication Engineering

Signature of the Internal Examiner

Signature of the External Examiner

ABSTRACT

Using up the contents of your fridge often requires touch creativity, which is typically hard to come back by in a very busy world. This aim is to style a model that may scan the things in your fridge and recommend recipes that supported what you've got, even taking your preferences and dietary restrictions under consideration. Computers are commencing to process globe objects without the requirement for codes or human intervention in their own language. This project aims to create a module that uses image recognition to detect vegetables and fruits and display recipes that include those foods. This concept uses a camera that is installed in a refrigerator to show it into a sensible one. Using the camera, the system identifies objects that are placed inside using complex image recognition algorithms. If a vegetable or fruit is placed inside, the system will identify it and displays its name on the screen. It creates as an indication of the chances of image recognition. The platform enables users to put any number of various vegetables into the refrigerator. The system then checks a list of recipes that contain the ingredients on the table and filters those that feature the vegetables available. Together with this, we also aim to watch the contents of the refrigerator and find the freshness and age using various sensors and also provide inventory and warnings when they're on the verge of completion.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, Mrs.A.ANILET BALA her valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of her busy schedule, she has extended cheerful and cordial support to me for completing this research work.

Author

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

When we're shown a picture, the objects that are present inside the image are immediately. At the same time, it requires heaps of time as well as preparatory training data for enabling machines with the same recognition capabilities. With recent advances in the field of Deep Learning (DL), hardware in terms of processing power and the advent of Big Data, the Computer Vision domain has become a significantly simpler and increasingly instinctive. Object Detection and its applications have become pervasive across a gamut of fields. From assisting self-driving cars to drive in the presence of traffic to spotting unruly incidents and rough conduct in packed places, Analyzing and constructing scouting reports to help sports teams to making sure that optimum internal control of manufacturing parts is guaranteed, its applications have become ubiquitous. Object Detection can be realized in the most efficacious manner with the help of DL.

A pair of methodologies by which deep learning can surpass existing object detection techniques are listed next. Rather than acquiring patches from the underlying image, the image is skilled in a neural network to scale back the size. The neural network is accustomed to suggest selective patches of the image. Now rather than preparing multiple neural networks to tackle the different aspects of object detection, A single, deep neural network will be attempted that can understand and provide a solution to the entire problem. The upside of taking this approach is that each of the discrete modules of a neural network will aid in the optimization of the contrary parts of the identical neural network. This can facilitate the combined training of the entire deep model. Training a model from scratch is often cumbersome and consumes a lot of time and data, to ameliorate a situation like this Transfer Learning can be employed where we can complete the training process with fewer data points and lesser time . In transfer learning the understanding, patterns and learning attained from one task is utilized and applied in another task.

The idea is based on the logic the two similar tasks that appear completely unrelated, might have identical underlying patterns based on which the model detects the target

so, we make use of this training in our task directly, bypassing the repetition of the extensive training which was initially performed. The improvement in accuracy while decreasing training time and data by employing Transfer learning has been repeatedly, widely and reliably demonstrated now.

CHAPTER 2

LITERATURE SURVEY

2.1 Object Identification techniques

Zhong-qiu zhao et al. (2019) was referred to for understanding and analyzing object detection frameworks dependent on DL that tackle various aspects of the problem, like occlusion, clutter, and low resolution, on r-cnn. Xinyi zhou et al. (2017) was used to assess the impact of deep learning especially in the field of object detection utilizing faster r-cnn. From shaoqing ren et al. It was surmised that state of the art object detection networks relied on region proposal algorithms to estimate the positions of the objects. The running time of the detection networks has been substantially decreased in advanced networks like r-cnn and sppnet, this reveals insights about the computation power consumed by region proposal. An rpn can be described as follows, a fully convolutional network that estimates the boundaries of the objects and the scores reflecting the certainty that an object is contained, at every position, simultaneously. The training process of the rpn is 'end-to-end' in nature to realize accurate region proposals, these are further utilized by the fast r- cnn for detection. Earlier work regarding object detection frame it as a classification problem thereby utilizing classifiers to perform detection. By using yolo, we look at object detection as a regression problem where the target variables are bounding boxes that are isolated in space and the corresponding class probabilities. A single neural achieves this directly from full images by running it through just once. As the entire pipeline just one network, end-to-end, direct, optimization of the performance is made possible. It performs other detection methods like r-cnn and dpm, when generalizing from natural images to other domains like artwork. Unlike the rcnn models which use rpn, which require multiple passes on the image, the bbox predictor divides the image into grids and generates the boxes in one go, thus, the name 'You only look once'. Yolo works with images only once it is substantially faster than all models of rcnn .

2.2 Sensor Array and Integration Methods

Hsin-hanwu et al. (2017) suggest that a module can be attached to a traditional refrigerator in order to equip it with the functionalities of a smart refrigerator. Shouming qiao et al. (2017) experiments with the idea of using rfid and bar code scanners for detecting and adding food items to the inventory and haidawati nasir et al. (2018) proposes a system that can notify and alert users about the status of the contents inside their refrigerator using a WiFi module. These papers were used to gain insights for the design of an integrated sensor array that can monitor and notify the status of the contents of a refrigerator.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Statement of problem

There has been massive research in the field of Deep Learning in recent years along with efforts to use it in culinary applications. But there has been no practical implementation in refrigerators. A Low Cost module is required that can convert existing normal refrigerators into smart ones. An Autonomous Sensor array is required that can constantly monitor and notify users about the state of the refrigerators contents. A Light, Ac- curate and Low-Latency model is required that can perform object-identification and recommends recipes.

3.2 Objective

A hardware module is to be designed which when attached to the regular refrigerators, turns them into SMART REFRIGERATORS, comprising of the following : A camera that is connected to a processor to capture the real time images of the objects and send it to the object recognition module. Effective and efficient Object identification algorithm using Deep learning. Sensor array integrated with the processor ,consisting of several sensors for monitoring the food items.

3.3 Method of data collection

The images for training the Mobile Net model (Variant 1) were primarily downloaded from the Image-Net data set and prepossessed and augmented using Keras. This data set contained 500 training images for each class. The data which was used for the pre- trained YOLO model (Variant 2) training was obtained from Microsoft's Common Objects in Context (COCO) data set which contains 1000 training images for each class.

3.4 Tools for data analysis

The Pandas software library was used for data analysis along with data manipulation. Matplotlib was used for plotting and graphical analysis of data.

3.5 Limitation of study

Only 4 Classes have been defined. The system is dependent on user arrangement of commodities for efficient identification of the classes. The time required to display results i.e. Latency, depends on the speed of internet connectivity. The accuracy of the object detection primarily depends on the illumination provided by the refrigerator, the intensity of which might vary from one model to another, resulting in accuracy variations.

CHAPTER 4

SYSTEM MODEL DESCRIPTION

4.1 System Design

SOFTWARE SPECIFICATIONS:

I. Object Identification Module:

a) We use Convolutional Neural Network for object identification by implementing YOLO architecture. Python code is written on Google Collab Notebook.

b) Libraries required: pandas, numpy, os, pickle, matplotlib and darkflow

II. Deploying the model on Cloud:

a) Google AI platform – To train the object identification model.

b) Google Compute Engine - To deploy the trained model on the cloud.libraries and packages required: Google Cloud SDK, inotify tools . Google Cloud VM(Virtual Machine) Instance details- Debian Operating System, 3.75GB RAM, 10GB Storage Memory.

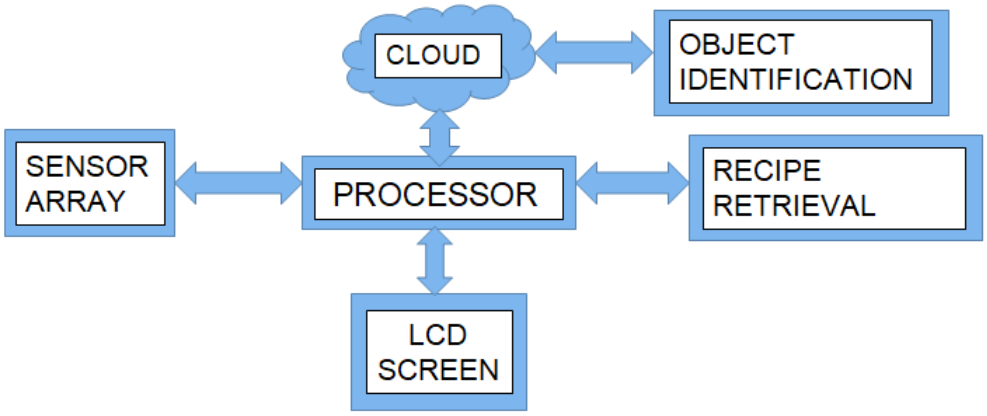


Figure 4.1: System design

4.2 Flow Chart

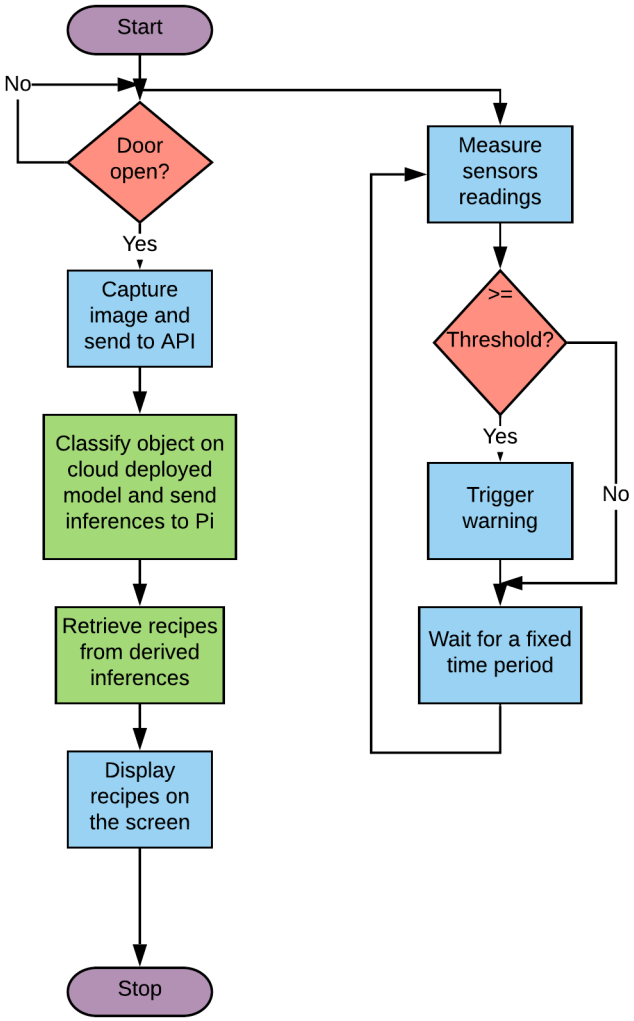
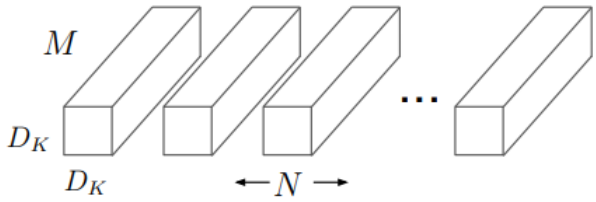


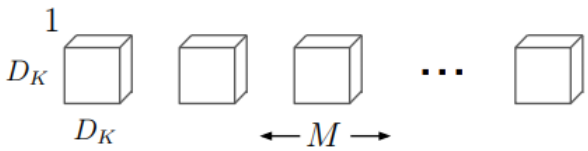
Figure 4.2: Flow chart

4.3 Architecture

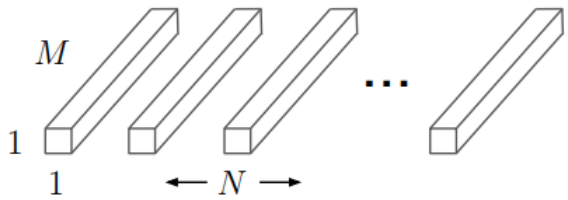
Variant 1(MobileNet): Depth-wise separable convolutions are utilized here which is tantamount to saying that it performs one convolution on each color channel. This is in contrast with the usual practice of combining all the channels and flattening. The paper's authors elucidate this as follows "For MobileNets the depthwise convolution applies one filter to every input channel. The pointwise convolution then applies a 1×1 convolution to mix the outputs of the depthwise convolution. a typical convolution both filters and combines inputs into a brand new set of outputs in one step. The depthwise separable convolution divides into two layers namely a separate layer for combining and a separate layer for filtering as in figure 4.3. This factorization drastically reducing computation and model size. "



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 4.3: Convolution filters

So, the architecture of MobileNet is as follows, consisting of 30 layers as in figure 4.4 with:

1. depth wise layer
2. depth wise layer with stride 2
3. pointwise layer that results in the doubling of the number of channels
4. convolutional layer with stride 2
5. pointwise layer that doubles the number of channels. The parameter and speci-

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 4.4: Mobile net architecture

fications of this pre-trained Neural Network (henceforth referred to as weights) were downloaded and modified to suit our purpose. This is done by training the Neural Network(NN) on the images that we might encounter during runtime. For this purpose, we train the NN on 3500 images belonging to 7 classes of fruits. These are tested during runtime, on 50 images belonging to the same 7 classes of fruits.

Variant 2 (YOLO): This variant of our model is an object detector because the name suggests it detects objects within images. It's different from the mobile net model in this it's ready to identify multiple objects within the identical image, as hostile a picture classifier, which only classifies opposed to. Our network uses features from the whole image to predict each bounding box. The bounding box is an imaginary, 2-dimensional, rectangular box, drawn around an object as in figure 4.6, such it encompasses it. Here the prediction of every bounding box in the picture, covering all the classes is performed at the same time. This suggests that the reasoning done by the network is done in a global manner regarding the complete image as well as each object that the image consists. The Yolo algorithm allows real-time speeds and end-to-end training and does this without affecting the average precision and ensuring that it is kept high as in figure 4.5. The system dissects the input image in the form of an s × s grid as in figure 4.7. In case the middle of an object lies

when the model thinks that there is no object inside the bounding region. Else, The

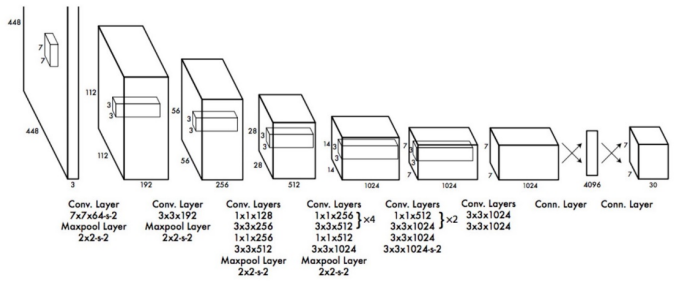


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Figure 4.5: YOLO architecture

output score should ideally be calculated by the intersection over union (IOU) of the ground truth with the predicted box. cell, irrespective of the amount of boxes O. Every

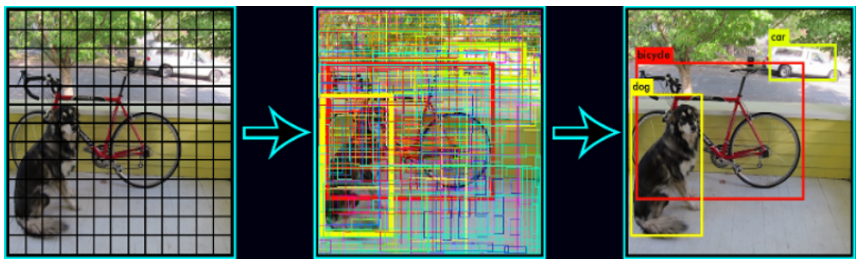


Figure 4.6: YOLO

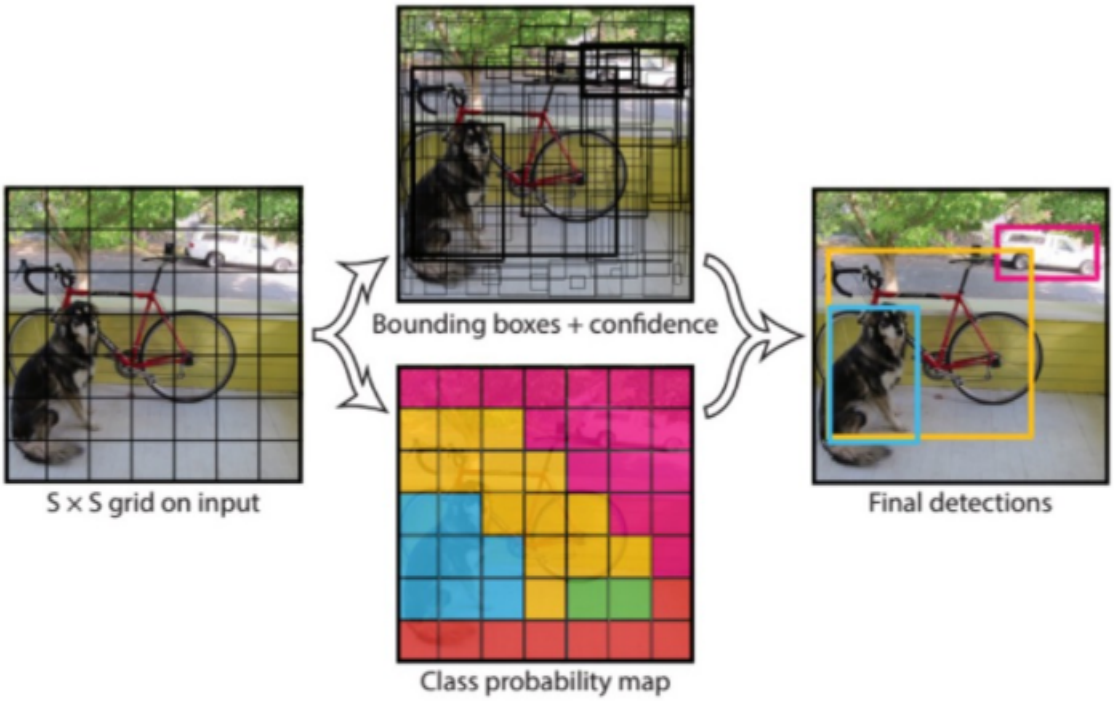


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Figure 4.7: YOLO mapping

bounding region outputs 5 predictions. They are, (x,y) which are the co-ordinates that represent the position of the bounding region's center with respect to the borders of the grid cell. (w,h) which stand for the width and height are anticipated with respect to the whole image. Also, every cell of the grid also estimates p probabilities which represent the conditional probability, i.e. given that an object is contained in the bounding box. This reflects the certainty with which the model thinks that the bounding box contains an object and that the accuracy which it attributes to the prediction that this object corresponds to a particular class.

4.4 Hardware description

The major concern pertaining to food wastage is Food hygiene and safety. The food standard must be checked and kept safe from rotting and spoiling by the climatic components like light, temperature, moistness. In this manner, it is valuable to introduce quality checking gadgets at food stores and homes. This gadget keeps a watch on the natural factor that causes rot of the nourishment. Refrigeration and vacuum storage controls many other natural factors. In this project, The quality checking device keeps a watch on natural factors like alcohol, spoilage, temperature, light, and humidity. The main processor board used is Node MCU- 8266. It is interfaced with sensors like MQ3 to detect food quality and spoilage. It is an IoT device that sends the sensor values to a database that is connected to a dashboard. The IoT dashboard is used to log and monitor the sensor's data. The dashboard used here is Cayenne. Because of the internet of things, the data can be logged from anywhere, anytime, and from any gadget.The position of these sensors are shown in figure 4.8

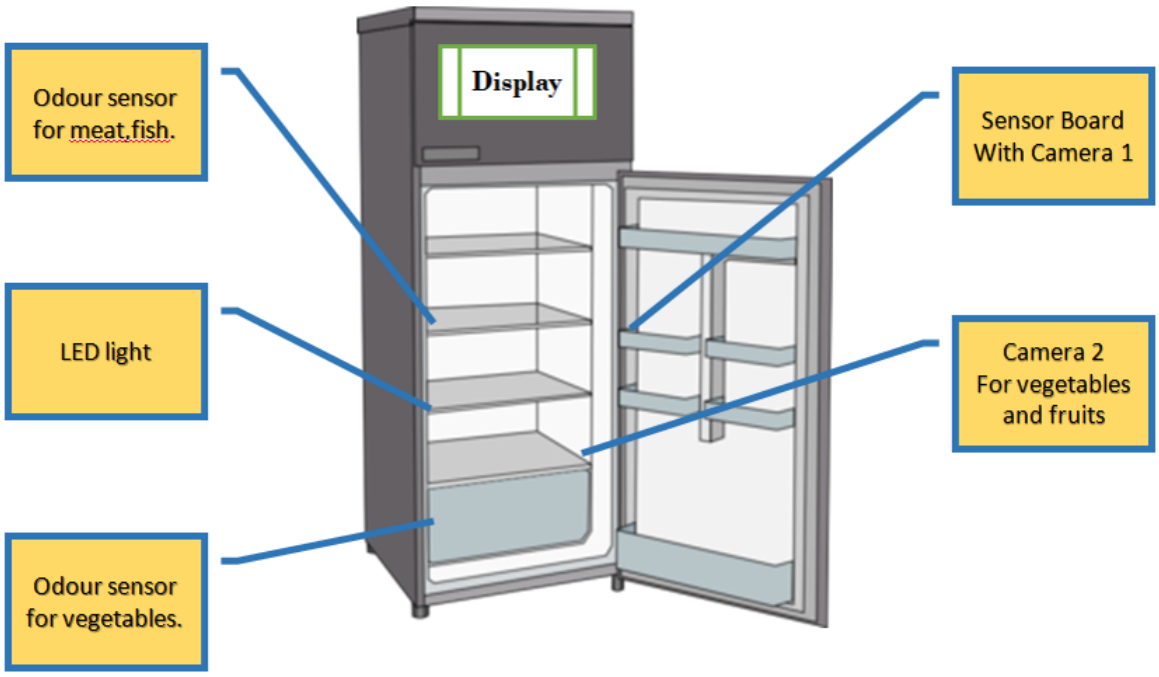


Figure 4.8: Position of hardware components

DHT 11 :DHT 11 is a humidity and temperature sensor as in figure 4.9. It consists of two components namely a humidity sensing material and an NTC temperature sensing device. It is basically a thermistor. A thermistor is a resistor that works on a principle that its resistance changes with change in temperature. The sensor components sense and send the data to the micro controller board. It has 3 pins – Vcc, data out,gnd. The Vcc and gnd are connected to common Vcc and GND respectively.

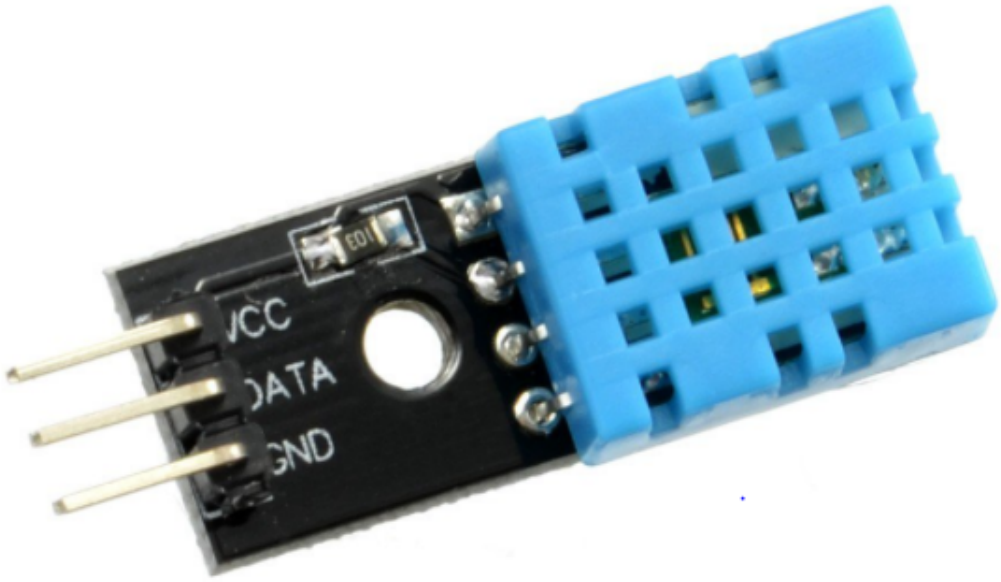


Figure 4.9: DHT11 sensor

MQ3 Sensor :MQ3 sensor is used to identify and detect the presence of ethanol and alcohol as in figure 4.10. Sno2 is used here as sensitive material. Its conductivity diminishes in clean air and increments when foul smell recognized. It's conductivity increments with the expansion in the concentration of ethanol and alcohol. It has great affectability to ethanol and great resistance from unsettling factors like fume, smoke, and gas. This sensor has analog and digital output pins. The sensor gives an ana-

log output to alcohol concentration. A decayed or rotten food releases ethanol gas which triggers the sensor. The sensor has four pins namely Vcc, and ground, analog out, digital output. The ground and Vcc are connected to common ground and Vcc. The analog output pin is connected to the A0 pin of nodemcu while the digital pin is left unconnected.

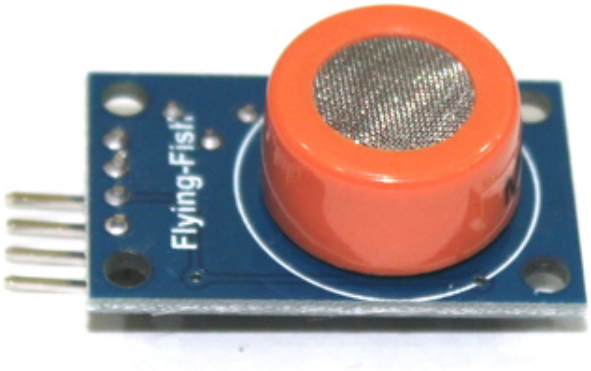


Figure 4.10: MQ3 sensor

NODEMCU :Nodemcu is a firmware that open-source prototyping board design as in figure 4.11. The board functions as a dual in-line package(dip). It is a small board with an integrated micro controller. This is a surface mount device that can be used on breadboard. It has a wi-fi soc support by esp – 12 . The soc is integrated to Tensilica Extensa lx106 core which is mainly used for IoT applications.

Figure 4.11: Nodemcu

RASPBERRY PI : The latest version of the raspberry pi series is Raspberry Pi 4. A raspberry pi is very similar to a personal computer's CPU as in figure 4.12. It has its own processor, memory devices, buses, etc. It is very cheap and compact. It is the size of your debit card. It has to be connected to peripheral devices to work with it.

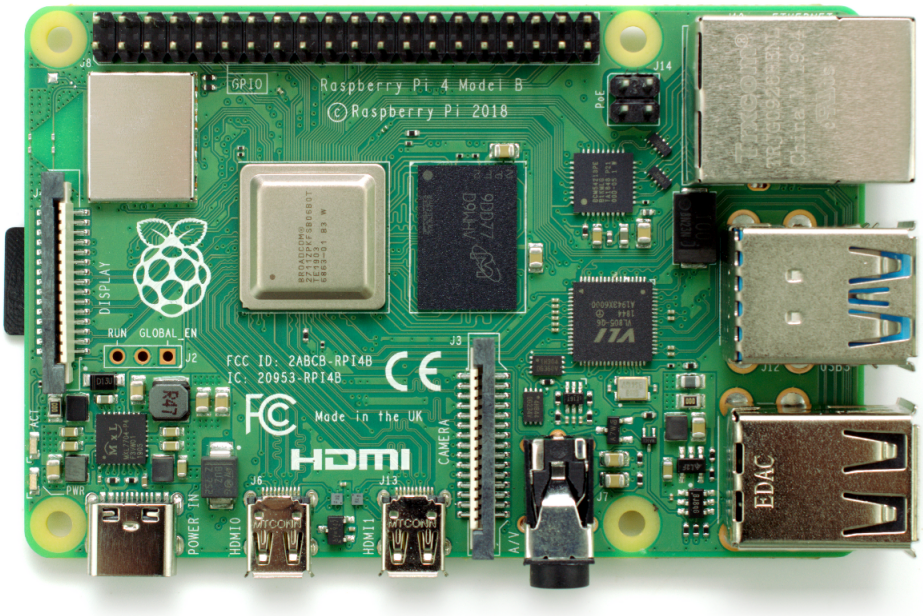


Figure 4.12: Raspberry Pi

Specifications

• System on chip: Broadcom BCM2711

• GPU: videocore VI

• CPU used : Quad-core 1.5ghz Arm Cortex-A72 based processor

• Memory size: 4GB LPDDR4 RAM

• Connectivity : Bluetooth 5.0/802.11ac Wi-Fi, Gigabit Ethernet

• Video and sound: 2 x micro-HDMI ports supporting 4K@60Hz displays via HDMI 2.0, MIPI DSI display port, MIPI CSI camera port, 4 pole stereo output and composite video port

• Ports: USB 3.0 - 2, USB 2.0 - 2

• Power : 5V/3A via USB-C, 5V via GPIO header

• Expandability : 40-pin GPIO header

PICAM : : The 8mp camera module is able to capture still images and 1080p video that interfaces with the Raspberry Pi. The camera is a plug and the capture compatible latest version to Raspbian software which is perfect for recording video, time-lapse photography, security, and motion detection. The board is compact and small with size 25mm x 23mm x 9mm, and weighing around 3g as in figure 4.13, which is perfect for mobiles or other applications where size is significant. The sensor in the camera has an 8-megapixel resolution and includes a hard and fast focus lens onboard. The camera can capture 3280 x 2464 pixel static images and 1080p30, 720p60 and 640x480p90 video.

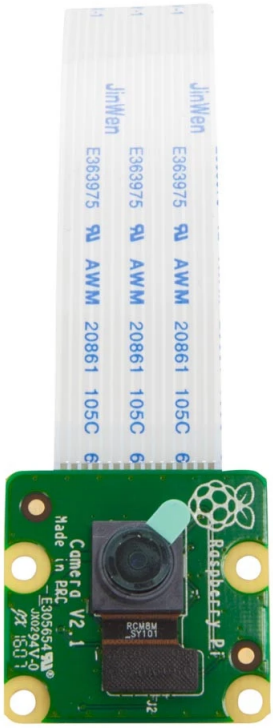


Figure 4.13: Pi camera

4.5 Interpretation

Our system has Raspberry Pi (local system) , camera(s) , sensors array inside the refrigerator.The screen is outside the fridge to have access to the data. Once it is powered through, it connects to the strongest wi-fi based on user input.

The camera is placed to achieve a bird's-eye view of the vegetables cabinet in our model, but in real time refrigerators, we will have cameras in multiple positions to cover all possible locations inside the refrigerator. When the refrigerator door is opened, the lights turn on and LDR detects the light that triggers the camera (connected to Raspberry Pi) . The camera captures the picture of the cabinet and sends it to the Pi as shown in figure 5.3.

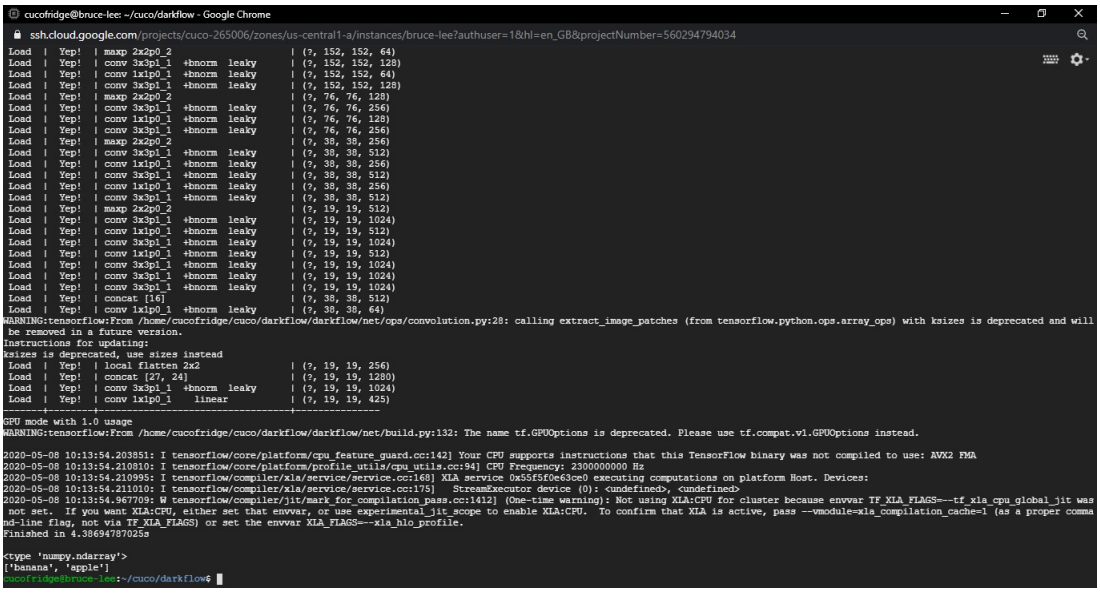


Figure 4.14: Cloud computation

Our object recognition model (YOLO v2) is deployed on the cloud (Google Cloud Platform) for fast and efficient processing. Hence, the Pi uploads the captured image to the Cloud Compute virtual machine which contains our object detection model. Here, the compute machine performs complex computations on the uploaded picture by passing it through the model for object identification and it takes approximately 8 seconds to recognise the items as shown in figure 4.14.

Once the recognition is done, the output is received by the local system (Raspberry Pi) in the form of a list as shown in the figure. This output is then used in the recipe retrieval process (on the Pi using an online browser) and the recipes for the objects detected are displayed on the screen as shown in the figure 5.4. This uploading and downloading of the image, and the recipe retrieval display consumes approximately 20 seconds bringing the total time of the complete process to nearly 30 seconds.

The IoT device is installed inside the fridge. The sensors read the data continuously and transfer them via wi-fi. The DHT 11 is a temperature and humidity sensor as mention before. It is a digital sensor that detects temperature and humidity every 2 seconds. The sensor with a voltage supply of 3.5 to 5.0 v and the temperature ranges from 00 c to 500 c. The sensor operates on a 1 – wire protocol which is implemented on the firmware because of which the sensor cannot interface with digital pins. The 40 bytes data read by the sensor consists of temperature and humidity. The mq3 sensor detects gases like ethanol and alcohol. It has to be placed where conservative foods(fruits and vegetables) are kept. The sensor keeps on detecting the concentration of ethanol. Once the con-

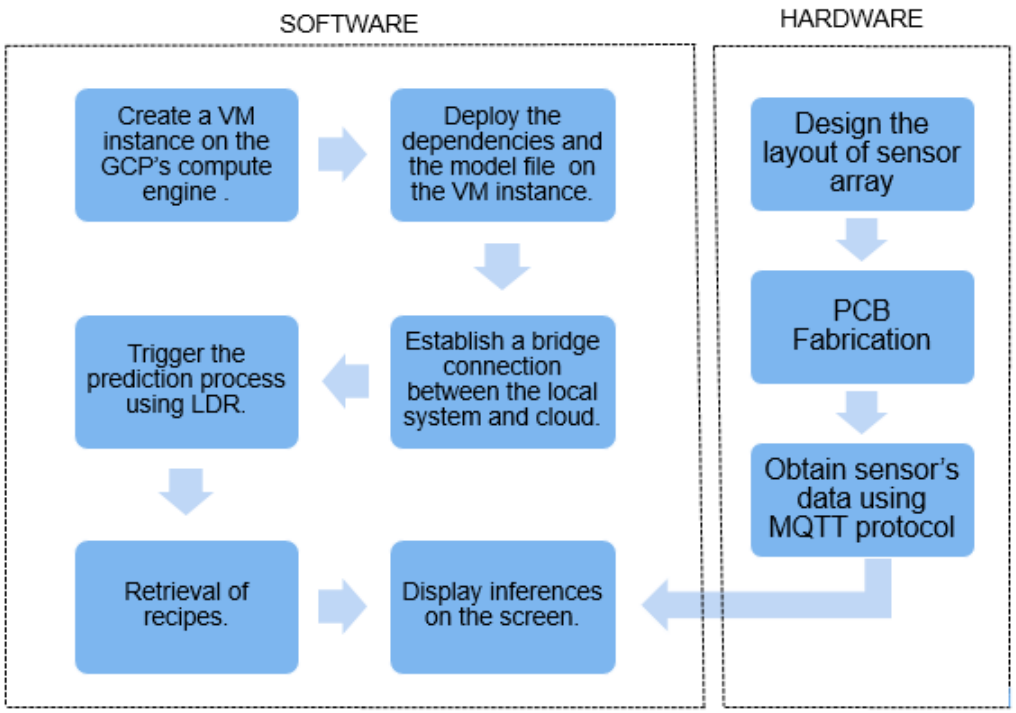


Figure 4.15: Workflow

centration reaches the threshold level, the sensor signals to the analog pin. The prototype board has an inbuilt ADC. All the sensor values keep sending to the cayenne server. The cayenne server stores the data and displays it on the dashboard.

Cayenne IoT : It is an online iot dashboard that is used to program a hardware related dashboard as shown in figure 5.6. It works with a raspberry pi and Arduino. It is a drag and drop tool that helps a connection with devices like sensors and motors to the online server. All we need to do is to install the cayenne agent and open an account and wire up the motors and sensors. Make sure that the sensors are mention on the cayenne server. In the software, you can get access to the GPIO lines. An Arduino board with a wifi module are to be connected to the sensors. An Arduino program that communicates between the sensors and dashboard is uploaded. In the dashboard, pick a suitable device and assign a GPIO

pin and select the display value. To get real-time data, the dashboard reads the sensor every 15 seconds. In this model, when the threshold of the mq3 sensor is met i.e more than 200, it triggers and sends a notification to the owner as shown in figure 5.8. The cayenne dashboard can be accessed on smartphones also as shown in figure 5.7.

CHAPTER 5

CONCLUSION

5.1 Experiments and results

SOFTWARE:

The training process for the YOLO model took over 5 hours with a final training set loss of 0.082. The model's efficiency was empirically observed to be **93.77 percent**. To test the real-time performance of our model, multiple objects were placed in the same picture and tested on. The model was successfully able to identify all the images. The metric used is confidence score as output by the model. The objects were kept at a distance of 1m.

IMAGE	No. of Objects	Classes of Objects	confidence score
1	1	Banana	0.45
2	1	Apple	0.63
3	1	Orange	0.7
4	2	Apple	0.5
		Orange	0.6
5	2	Banana	0.67
		Apple	0.43
6	3	Apple	0.49
		Banana	0.69
		Orange	0.57

Figure 5.1: Confidence score table

As observed, the confidence score is independent of the number of classes present as shown in figure 5.1. This is because the model attempts to draw bounding boxes around objects that it identifies and predicts the confidence of that box alone. It was observed that on occlusion, the confidence score of the occluded object decreased considerable, yet the model was still able to make a fairly confident prediction.

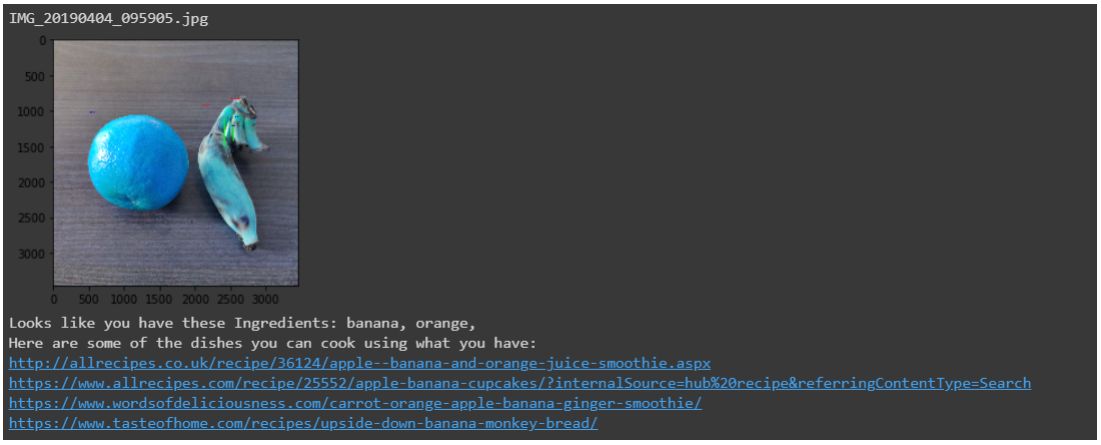


Figure 5.2: Output

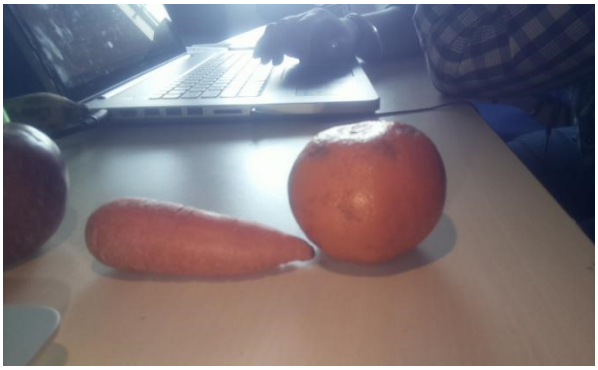


Figure 5.3: Image captured

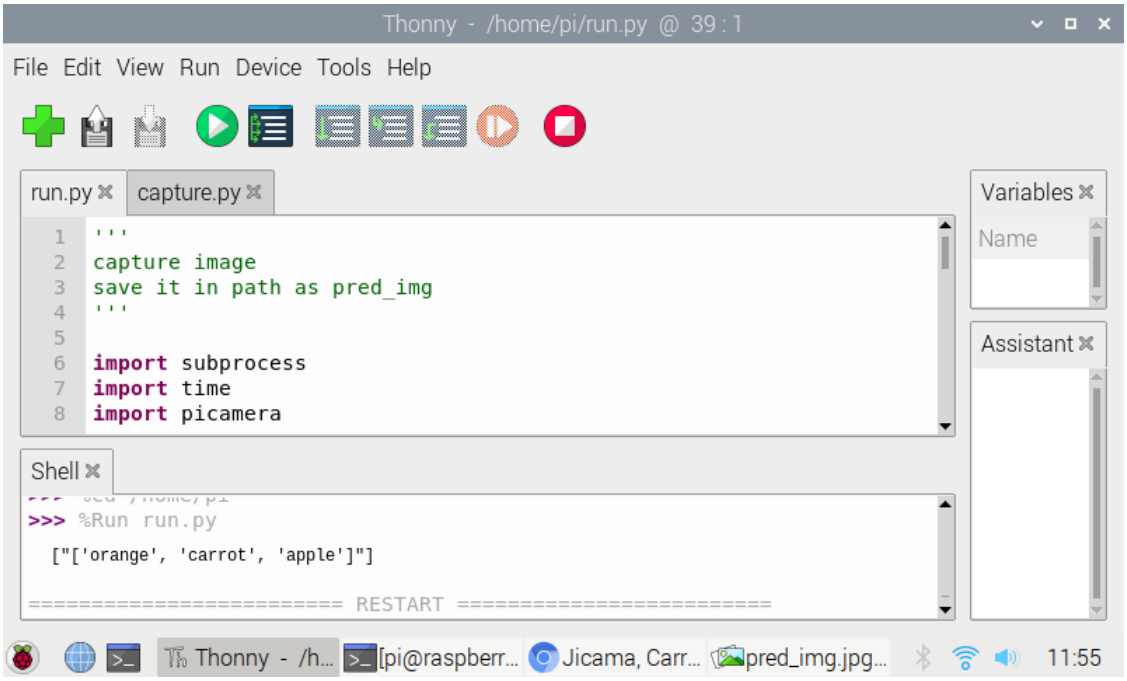


Figure 5.4: Items detected

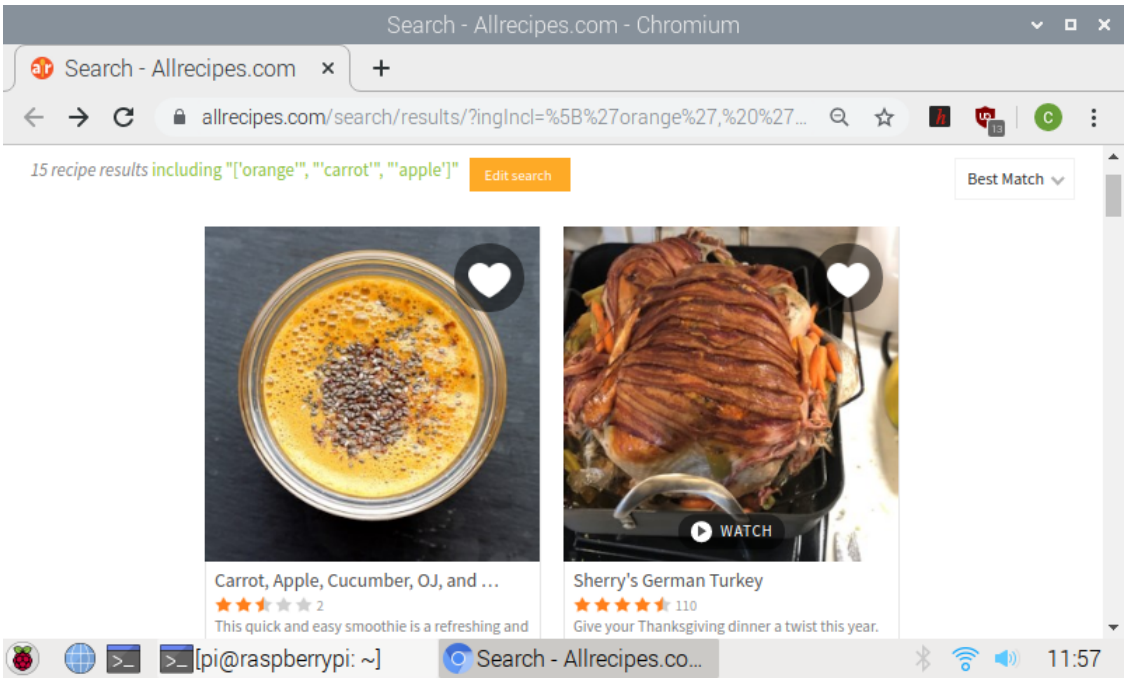


Figure 5.5: Recipes displayed

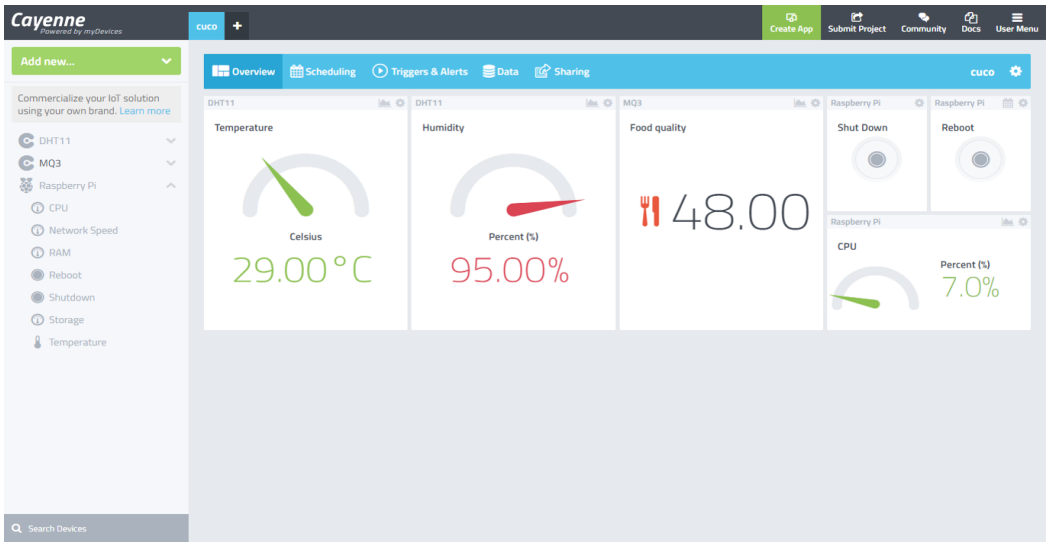


Figure 5.6: Dashboard on screen

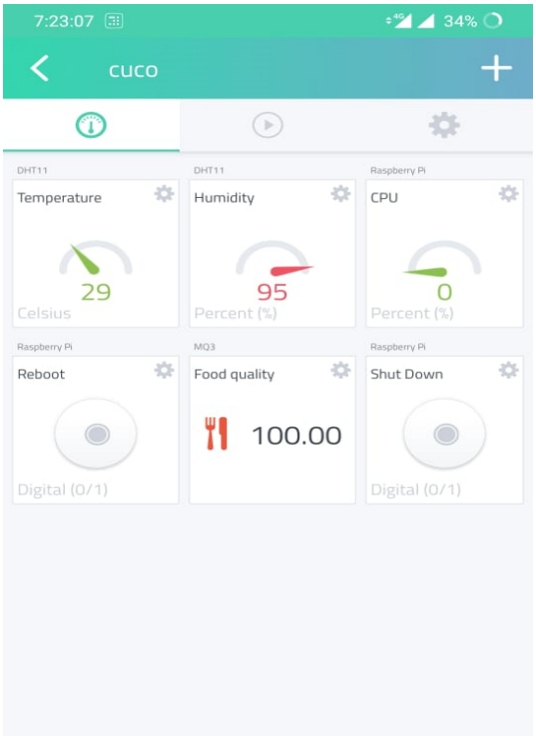


Figure 5.7: Dashboard on phone

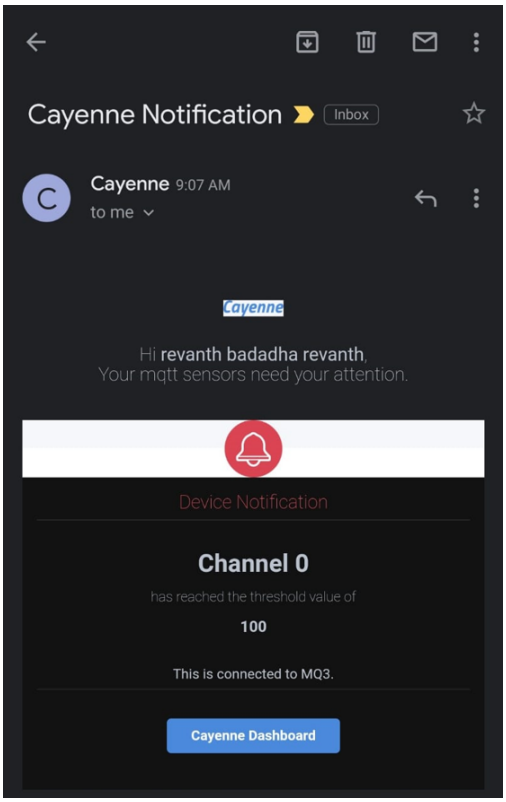


Figure 5.8: Notification email

5.2 Applications

• Smart home applications for inventory.

• Recipe recommendations.

• Monitoring of several factors like freshness and age of the components.

• User interactive intelligent refrigerator.

5.3 Realistic constraints

• YOLO imposes strong spatial constraints .

• YOLO struggles with tiny objects that are in groups.

• Despite this limitation it still surpasses the image classifier as it efficiently recognizes multiple occluded objects.

• All the sensors should be compatible with lower temperatures.

• Hazing of the images, hence Dehazing is required.

• User has to organise and arrange the items inside the refrigerator properly.

• Requirement of network connectivity all the time.

5.4 Conclusion

A regular refrigerator which turns SMART by adding a single module that :

• Recommends recipes automatically.

• Informs about food expiry .

• Provides information about food items.

• Comprises of a display unit with advanced and user friendly UI.

5.5 Direction for further research

• A voice assistant can be integrated with the module.

• Automation of inventory restocking.

• Mobile application can be developed for user to access anywhere.

• Adds food items to the inventory if needed.

5.6 Multi-disciplinary aspects

1) Utilization of several machine learning techniques in training of the neural network.

2) Use of microprocessor like raspberry Pi or interfacing and execution.

3) Design of a User Interface dashboard for real time data display.

5.7 Engineering Standards used

1) IMAGE :JPEG FORMAT.

2) MQTT - ISO/IEC 20922

3) IEEE 802.11 – WI FI

5.8 Statement of Contribution

This report and the project that it has entailed, is a collaborative effort of all the group members.

Vyshnavi Muppirala :

1. Building and training the neural network.
2. Deploying the model on cloud platform.
3. Creating an API and connecting with the Pi.

C.V.Pranav :

1.Building and training the neural network.

2. Deploying the model on cloud platform.
3. Creating an API and connecting with the Pi.

Revanth B :

1.Design of Sensor Array

2.Monitoring and analysing the data from the MCU.

3.Recipe Retrieval.

APPENDIX A

CODING

A.1 Dashboard

```
define CAYENNEDEBUG
define CAYENNEPRINT Serial
include <CayenneMQTTESP8266.h>
define MQ3PIN A0
// WiFi network info.
char ssid[] = "revanth";
char wifiPassword[] = "1234567890";
// Cayenne authentication info. This should be obtained from the Cayenne
Dashboard.
char username[] = "c1006df0-5deb-11ea-84bb-8f71124cfdfb";
char password[] = "b27cccd67627e515375ce15a9ee1df3066beea9c";
char clientID[] = "97731f60-6100-11ea-ba7c-716e7f5ba423";
unsigned long lastMillis = 0;
int reading = 0;
void setup()
Serial.begin(9600);
Cayenne.begin(username, password, clientID, ssid, wifiPassword);


void loop()
Cayenne.loop();  // Default function for sending sensor data at intervals
```

to Cayenne.

// You can also use functions for specific channels, e.g CAYENNEOUT(1) for sending channel 1 data.

CAYENNEOUTDEFAULT()

CAYENNEOUT(0)

reading = analogRead(A0);

// Write data to Cayenne here. This example just sends the current uptime in milliseconds on virtual channel 0.

Cayenne.virtualWrite(0, millis());

// Some examples of other functions you can use to send data.

Cayenne.celsiusWrite(1, 22.0);

Cayenne.luxWrite(2, 700);

Cayenne.virtualWrite(3, 50, TYPEPROXIMITY, UNITCENTIMETER);

Cayenne.virtualWrite(0, reading);

A.2 Raspberry Pi display

```
import RPi.GPIO as GPIO
from time import sleep
from subprocess import call
import os
LimitSwitchOpen = 4
GPIO.setmode(GPIO.BCM)
GPIO.setup(LimitSwitchOpen, GPIO.IN)
print('monitoring fridge')
poll interval = 0.001
while(True):
DoorOpen0 = GPIO.input(LimitSwitchOpen)
DoorOpen1 = 0
while DoorOpen0==0 and DoorOpen1==0:
sleep(poll interval)
DoorOpen1=GPIO.input(LimitSwitchOpen)
if DoorOpen0==0 and DoorOpen1==1:
call(['cd',' '])
x = call(['python','run.py'])
print(x)

print('image caped')
while DoorOpen1==1:
sleep(1)
DoorOpen1 = GPIO.input(LimitSwitchOpen)
```

A.3 Recipe retrival code

```
”’
run.py
capture image
save it in path as pred img
”’
import subprocess
import time
import picamera
camera = picamera.PiCamera()
camera.rotation = 180
camera.capture(’/home/pi/BruceLee/pred images/pred img.jpg’)
x = 1
r=1
img path = ’/home/pi/BruceLee/pred images/pred img.jpg’
VM dest = ’cucofridge@bruce-lee:/home/cucofridge/fire/’
VM res = ’bruce-lee:/home/cucofridge/cuco/results.txt’
pi res = ’/home/pi/BruceLee/’
subprocess.call([’gcloud’,’compute’,’scp’, img path, VM dest])
while (x != 0) and (r==’0’):
time.sleep(10)
subprocess.call([’gcloud’,’compute’,’scp’,VM res,pi res])
with open(’/home/pi/BruceLee/results.txt’,’r’) as res:
r = res.read()
r = [line.rstrip(’,’) for line in
```

```
open('/home/pi/BruceLee/results.txt')];
print(r)
n= len(r)
import webbrowser
if n==1:
webbrowser.open('https://www.allrecipes.com/search/results/?ingIncl='
+r[0]+'sort=re')
elif n==2:
webbrowser.open('https://www.allrecipes.com/search/results/?ingIncl=
'+r[0]+','+r[1]+'sort=re')
elif n==3:
webbrowser.open('https://www.allrecipes.com/search/results/?ingIncl=
'+r[0]+','+r[1]+','+r[2]+'sort=re')
exit()
```

A.4 YOLO Code

!apt-get update

!pip3 install numpy

!apt-get install python-opencv -y

!pip install cython

Commented out IPython magic to ensure Python compatibility.

Clean up the directory

Clone the darkflow repository

!git clone https://github.com/thtrieu/darkflow

Change into the darkflow dir and install darkflow with pip

!zip -r /content/darkflow.zip /content/darkflow/

Commented out IPython magic to ensure Python compatibility.

!python setup.py build ext inplace

from google.colab import drive

drive.mount('/content/gdrive',force remount = True)

import pandas as pd

def done(identified):

root = 'Looks like you have these Ingredients: '

for i in identified:

root=root+i+', '

print (root)

df = pd.read csv('/content/gdrive/My Drive/recipes/recipes.csv')

df.RECIPE = df.RECIPE.astype(str)

df.index = df.INGREDIENT

del df['INGREDIENT']

```
recipes = []
for i in identified:
templist=[]
temp = df.loc[i,'RECIPE']
temp = temp.split(',')
for j in temp:
recipes.append(j)
recipes=list(set(recipes))
print ('Here are some of the dishes you can cook using what you have: ')
notneeded = list(set(labels)-set(identified))
for i in recipes:
print (i)
import os
os.getcwd()
import tensorflow
import cv2
from darkflow.net.build import TFNet
import matplotlib.pyplot as plt
options = 'model':'cfg/yolo.cfg','load':'/content/gdrive/My Drive/weights/yolov2.we
1.0
tfnet = TFNet(options)
import numpy as np
import matplotlib.pyplot as plt
cucoTestPath = '/content/gdrive/My Drive/CUCO/Find my recipe/'
labels = ['apple','banana','orange','carrot']
file = os.listdir(cucoTestPath)[-1]
```

```
print (file)
img = cv2.imread(cucoTestPath+file,cv2.IMREAD COLOR)
print(type(img))
results = tfnet.return predict(img)
print(type(results))
identified = []
for i in range(len(results)):
if results[i]['label'] in labels:
identified.append(results[i]['label'])
tl = (results[i]['topleft']['x'],results[i]['topleft']['y'])
br = (results[i]['bottomright']['x'],results[i]['bottomright']['y'])
label = results[i]['label']
img = cv2.rectangle(img,tl,br,(0,255,0),2)
img = cv2.putText(img,label,tl,cv2.FONT HERSHEY COMPLEX,1,(255,0,0),1)
img = cv2.cvtColor(img,cv2.COLOR BGR2RGB)
plt.imshow(img)
plt.show()
identified = list(set(identified))
done(identified)
```

APPENDIX B

SENSOR ARRAY

B.1 PCB layout

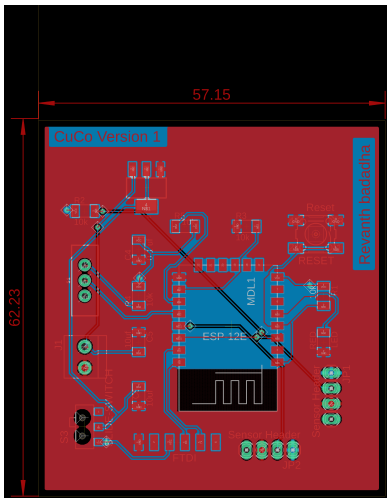


Figure B.1: Pcb layout

B.2 Freeboard dashboard

Figure B.2: Dashboard- freeboard.io

REFERENCES

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

2.**You Only Look Once**: *Unified, Real-Time Object Detection* Joseph Redmon , Santosh Divvala†, Ross Girshick¶ , Ali Farhadi† University of Washington , Allen Institute for AI† , Facebook AI Research¶