Auto Billing System using IoT and object detection

1st Hetansh Shah
Institute of Technology
Nirma University
Ahmedabad, Gujarat
21bec045@nirmauni.ac.in

2nd Himangi Agrawal
Institute of Technology
Nirma University
Ahmedabad, Gujarat
21bec046@nirmauni.ac.in

***Abstract*—Using the power of artificial intelligence, computer vision, and machine learning, AutoBill is an inventive autonomous checkout system that transforms the retail shopping experience. AutoBill seeks to improve safety, convenience, and efficiency in retail settings. With accuracy and speed, the system uses sophisticated computer vision algorithms to visually detect and identify items placed on the checkout counter. Furthermore, AutoBill can quickly identify items and precisely calculate their prices thanks to integrated machine learning models, which expedites the checkout process and lessens the need for manual input. Reducing human contact during the checkout process is one of AutoBill's main goals in order to improve safety for customers and store staff. The checkout process is automated by the system, which also decreases wait times, lines, and physical contact points, all of which improve the hygienic conditions of the store. AutoBill instantly creates an itemized bill and automatically adds the products to the virtual shopping cart upon item identification. Objects are classified using machine learning algorithms and the model is loaded into Raspberry pi. Raspberry Pi, alongside a camera module, captures and categorizes items instantly, eliminating the possibility of manual errors. Machine learning algorithms ensure flexibility across different product categories. Moreover, the system provides features for inventory management and sales analysis, empowering retailers with valuable insights. Empirical testing confirms its effectiveness across various retail environments. Raspberry Pi, alongside a camera module, captures and categorizes items instantly, eliminating the possibility of manual errors. This advancement has the potential to transform retail operations, offering seamless checkout experiences and fostering business growth.**

For contemporary retail businesses looking to improve customer satisfaction and streamline their checkout procedures, AutoBill offers a cutting-edge option. AutoBill is setting the standard for future retail technology by providing a seamless, effective, and secure shopping experience through the use of AI, computer vision, and machine learning.

***Index Terms*—Raspberry pi, machine learning, object detection, User interface, HTML, JS, Edge impulse.**

I. Introduction

The checkout experience is a critical factor in determining customer satisfaction and loyalty in today's fast-paced retail environment. Conventional checkout procedures often involve lengthy lines, manual scanning, and interactions with cashiers, which can be inefficient and pose health risks, especially in light of the COVID-19 pandemic. In order to solve these issues and usher in a new era of retail innovation, AutoBill emerges as a ground-breaking solution that revolutionizes the checkout process. AutoBill enables retail environments to quickly and accurately detect and identify items placed on the checkout counter by integrating cutting-edge computer vision algorithms. This feature not only speeds up the checkout procedure but also reduces the amount of manual input required, which lowers errors and increases overall efficiency. IoT serves as the platform for connecting the software and hardware systems to the embedded system environment. IoT has been used in this project as a platform to connect devices and allow them to communicate with one another in order to achieve the intended result. Customers search and gather all the items they require in a basket or cart at a department store or shopping center. After gathering all the merchandise, it is imperative that customers wait in line for billing, which congests the area and wastes a significant amount of their time. This work uses an object detection algorithm to overcome this, allowing for automatic billing by the customer in the cart. AutoBill's design philosophy places a strong emphasis on hygiene and safety. The system lowers wait times, lines, and physical contact points by automating the checkout process and reducing human interaction. This makes the store environment more hygienic for both customers and staff. Apart from optimizing checkout procedures, AutoBill gives security and ease of use top priority when processing payments. Customers can transact quickly and contactlessly by using secure QR codes, guaranteeing a flawless end-to-end shopping experience. All things considered, AutoBill is a revolutionary approach to retail technology that gives merchants a cutting-edge way to boost customer happiness, maximize operational effectiveness, and meet the demands of the contemporary retail environment. AutoBill creates a new benchmark for autonomous checkout systems by combining AI-driven automation, computer vision capabilities, and machine learning algorithms. This opens the door to a safer, more intelligent, and enjoyable shopping experience.

II. RELATED WORK

The current system uses LCD to show the product details and RFID tags to read the product details. The current system can perform automated billing. To enable this feature, billing data is stored via Zigbee and subsequently retrieved by the cashier via a physical cable or Zigbee[5].

Numerous approaches have been investigated to make use of technologies such as RFID, Wi-Fi, Zigbee, and even robots in retail settings[3]. Expense, the need for an active RFID tag power source, and how well products are able to inform customers are among the obstacles. A trolley control system was built on an FPGA and designed with Simulink[3]. It makes use of a camera to recognize colors and is compared to current RFID-based systems. In order to lower overall system costs, it places a strong emphasis on wireless technologies, cost-effectiveness, and simplified components[3]. Advanced technology implementations in retail settings, such as RFID, Wi-Fi, and robots, can be challenging and necessitate extensive infrastructure setup. There might be difficulties and specialized knowledge involved in smoothly integrating these technologies into current retail systems. Smaller companies or retailers with tighter budgets may find it prohibitive to invest in and maintain these high-tech solutions. Systems like RFID, Wi-Fi, and robotic assistants[3] may need to be set up upfront with a significant financial commitment. In [6], an Arduino microcontroller is utilized to redact product details through RFID technology. By connecting the mobile device to the store's server via Bluetooth, product information is sent to the customer's phone and bill information is sent to the cashier. The mechanism fall short in that people still have to wait in line to make payments, even though billing is completed in the cart itself.

In [7], the RFID reader (EM18) functions as a scanner, and the RFID tags function as a product. The lcd display (i2c) displays product details like the product name and price as soon as the tags are detected by the reader. Should the client choose to take the item out of the cart, they can do so by pressing the push button.After every product, the cost will be included. We have used object detection instead of RFID module. Numerous objects can be identified by object detection algorithms without the need for unique RFID tags for every item. This flexibility is especially helpful in situations where RFID tagging might be expensive or impractical, like large retail stores with constantly changing inventory. Deploying RFID tags may not be as economical as implementing object detection algorithms, particularly for companies that offer a wide range of products or regularly introduce new items. RFID tags can have substantial upfront costs because they require additional hardware and infrastructure for scanning and tagging. Object detection algorithms are useful in dynamic environments where objects are constantly moving or rearranging because they can identify objects in real-time without requiring physical contact or line of sight. On the other hand, RFID tags usually need to be in close proximity to a reader device in order to be detected, which could cause delays in object identification. Even in intricate and cluttered scenes, advanced object detection algorithms—especially those based on deep learning and neural networks—can identify objects with high precision and accuracy. This feature allows for accurate object identification in a variety of lighting, occlusion, and orientation scenarios—situations that could present difficulties for RFID-based systems.

III. PROPOSED WORK

We have utilized the Edge Impulse Platform for our object detection algorithm [2]. One of the top platforms for developing machine learning applications for edge devices is Edge Impulse, which is trusted by businesses and available for free to developers. The workflow of the process is described in fig[1]. Here, we are constructing a system that can identify the

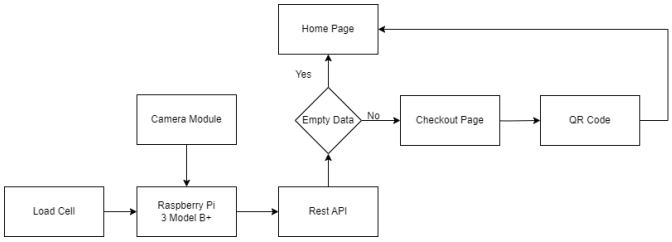

Fig. 1. Workflow

goods that are offered in the stores using machine learning. Next, we set up the Raspberry Pi 3B to run the system. Images are gathered to create the dataset 2. In order to create the machine learning model, a large number of product images are required. These product images are used in the model's training so that it can tell them apart. To improve detection accuracy, photos of the products that are sold in stores can be taken from a wide range of perspectives and magnifications. It utilizes the neural network concept to function. In order for neural networks to identify patterns in data sets, the more data available, the better. The dataset is uploaded and labelled. With this, the training dataset is established. For testing, the input image is taken. A image preprocessing block is used to manipulate the raw data and learning block to classify new data based on prior experiences. The image preprocessing block receives the color image, converts it to grayscale if desired, and then outputs an array of features from the data. Then a transfer learning block is employed, which processes all of the images and acquires the ability to discriminate between the classes. Each image's processed and raw features are produced. Upon initiating the process, features will be generated. The Feature explorer will then load, showing a plot of every feature in your dataset. Since images have many dimensions, a process on the dataset known as 'dimensionality reduction' is performed before visualizing the data. The features are reduced, and are then grouped according to similarities.

After the data processing, a neural network needs to be trained. Neural networks are a class of algorithms with pattern recognition capabilities, somewhat inspired by the structure of the human brain. The image data will be fed into the network that we are training, and it will attempt to map this to one of the three classes. Transfer learning is used to speed up and simplify the process of training. By only retraining the top layers of a neural network, it allows to build upon an already-trained model and produce far more dependable models that train much faster and require much smaller datasets. The base model and the network's learning rate can be adjusted. Following model completion, accuracy values will appear

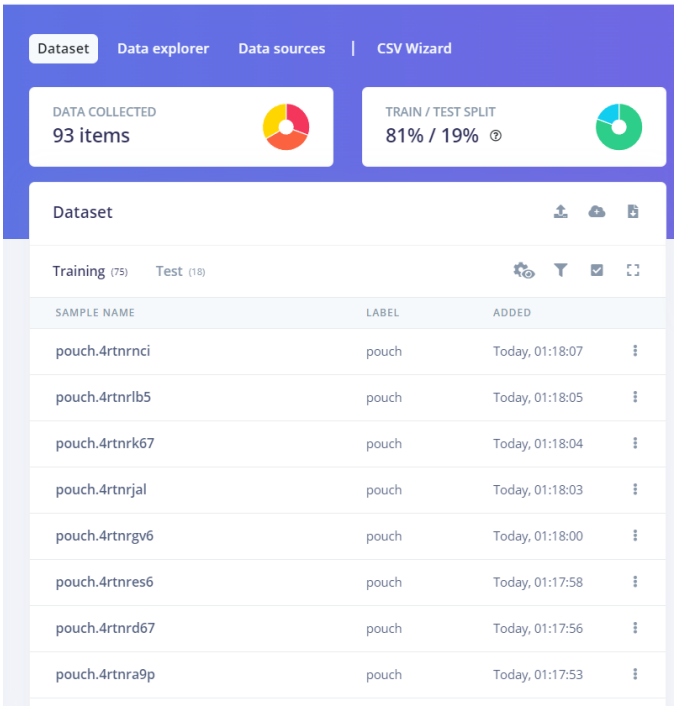

Fig. 2. Model Dataset

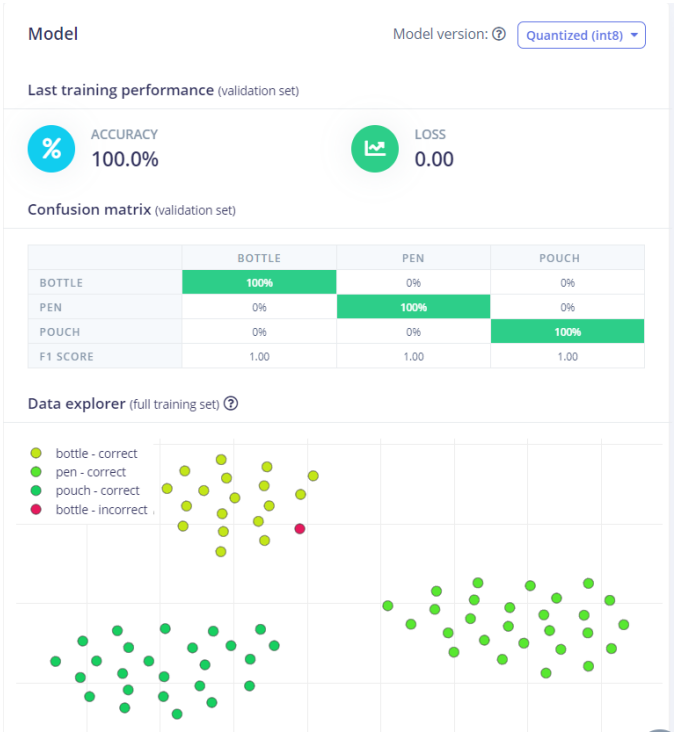

Fig. 3. Model training

Fig. 4. Model Testing

beneath the training output as shown in Fig3. The testing accuracy is shown in Fig4. The validity of the model can also be checked. The model is returned to the device. It reduces latency, maximizes battery life, and enables the model to operate without an internet connection. To integrate the model with the system, we have used the Linux Python SDK. We use the Raspberry-Pi camera to identify the items after loading the model. The objects' probabilities are displayed on the screen when we place them in front of the camera, as seen in Fig. 15. We have created a user interface for creating and paying bills. For frontend and backened APIs, it makes use of HTML, JS, and NodeJS respectively. The user is notified of any changes made by the front-end, which is always monitoring the back-end API for updates. An item appears on the front end as one that has been added to the cart after it has been added to the API. As the object is detected, the item name, its cost price and quantity is displayed on the screen. The user is then required to make the payment through UPI.

IV. BLOCK DIAGRAM

The block diagram of the implemented system is shown in Fig5.

V. HARDWARE AND SOFTWARE SETUP

A. *RASPBERRY PIE*

The Raspberry Pi 3 Model B+ is a versatile single-board computer renowned for its compact design, affordability, and impressive performance. Powered by a quad-core ARM Cortex-A53 processor running at 1.4 GHz, it offers significant processing power for a range of applications. With built-in dual-band wireless LAN, Bluetooth connectivity, Ethernet

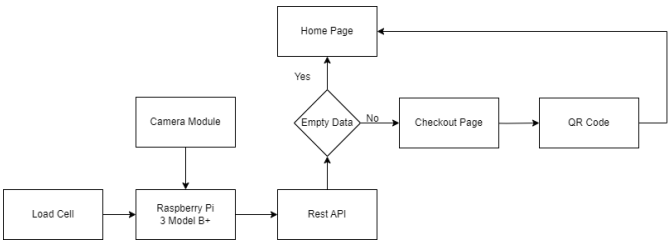

Fig. 5. Block Diagram

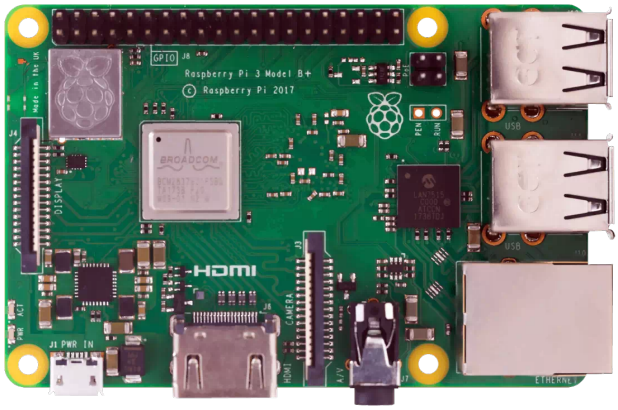

Fig. 6. Raspberry pi 3 Model B+ [2]

port, and four USB 2.0 ports, it provides ample connectivity options for various projects. Its 40-pin GPIO header enables interfacing with external devices and sensors, while the microSD card slot allows for easy storage expansion. Whether used for multimedia streaming, web browsing, gaming, or educational purposes, the Raspberry Pi 3 Model B+ remains a popular choice due to its low cost, small form factor, and extensive community support.

B. 5MP OV5647 Camera Module

The 5MP OV5647 Camera Module, crafted by OmniVision Technologies, is a compact imaging solution prized for its adaptability across diverse fields like robotics, surveillance, and embedded systems. Renowned for its high-resolution 5-megapixel sensor, this module excels in capturing sharp images and videos. Its petite size and efficient power usage make it a go-to choice for projects with limited space. Offering customizable image settings including exposure and color balance, it accommodates various lighting conditions effortlessly. Furthermore, its compatibility with popular platforms like Raspberry Pi democratizes access to advanced imaging

Fig. 7. 5MP OV5647 Camera Module[2]

capabilities, fostering creativity and advancement in computer vision applications.

C. RaspberryPi Imager

Fig. 8. Raspberry Pi Imager

Raspberry Pi Imager is a straightforward software utility created by the Raspberry Pi Foundation. Its purpose is to simplify the installation of operating systems onto SD cards or USB drives for Raspberry Pi computers. This tool streamlines the setup process by offering a simple interface where users can select their preferred operating system, pick the storage device, and begin the installation with minimal effort. It's especially beneficial for novices and hobbyists who wish to

embark on Raspberry Pi projects without dealing with the complexities of manual SD card flashing.

D. Raspberry Pi OS (Legacy, 32-bit) Full

Raspberry Pi OS, formerly Raspbian, serves as the official operating system for Raspberry Pi single-board computers. The Legacy, 32-bit edition of Raspberry Pi OS delivers a comprehensive computing environment tailored to Raspberry Pi hardware. It encompasses essential productivity tools like web browsers, office suites, programming aids, and multimedia software. Designed for older Raspberry Pi models lacking 64-bit support, this version ensures user-friendly navigation and wide compatibility within the Raspberry Pi community. It caters to diverse projects, spanning education, home automation, and media hubs.

E. PuTTY software

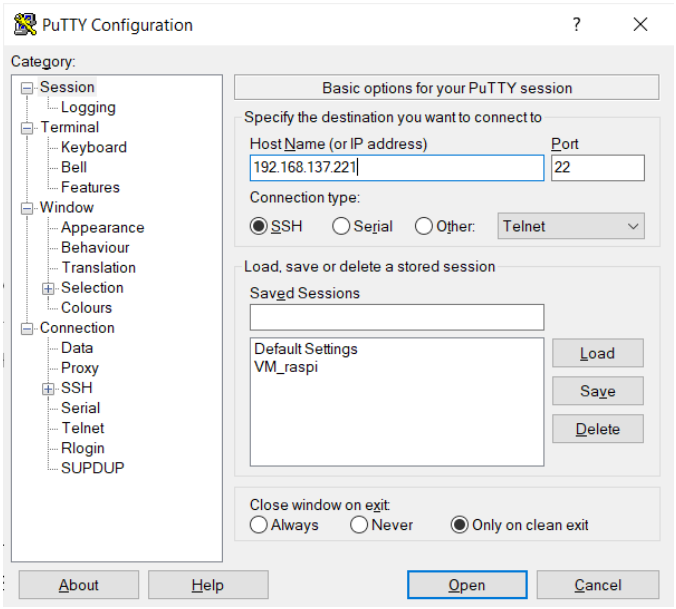

Fig. 9. PuTTY home page

PuTTY has played a significant role in enabling headless interaction with Raspberry Pi devices. Through its SSH (Secure Shell) support, PuTTY permits users to remotely access and administer Raspberry Pi systems without necessitating a graphical interface or direct physical link. This functionality proves invaluable for tasks like configuration adjustments, command execution, and issue resolution, particularly in scenarios where Raspberry Pi devices are situated in distant or inaccessible locales. By harnessing PuTTY's streamlined and effective interface, individuals can establish secure connections to Raspberry Pi units, facilitating smooth management and oversight of these adaptable computing devices.

F. RealVNC Viewer

Because it provides simple remote access and control, the remote desktop program RealVNC Viewer has been essential in helping Raspberry Pi users realize their full potential. Users may easily supervise and manage their Raspberry Pi projects by connecting to their devices from any location with RealVNC Viewer. This software makes it easier to do things like setup, monitoring, and diagnostics for Raspberry Pi projects without requiring you to be physically close to the device. It is an essential tool for both enthusiasts and pros as it offers a graphical interface for viewing and controlling the desktop environment of distant computers, including Raspberry Pi devices.

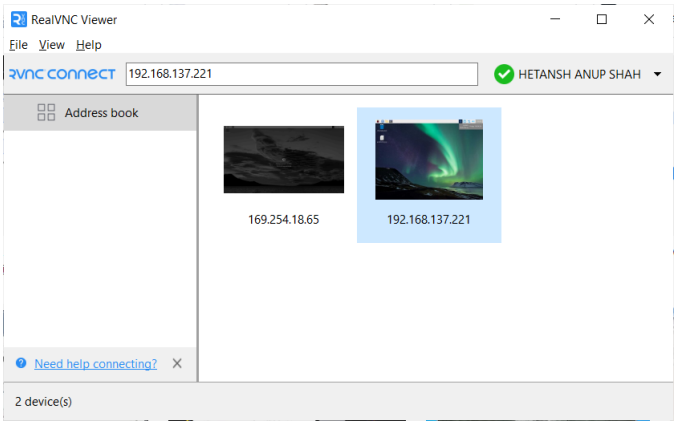

Fig. 10. RealVNC Viewer

G. Edge Impulse

Edge Impulse is a robust software platform tailored for developing machine learning models specifically for edge devices. In the context of an auto billing system, Edge Impulse is instrumental in object classification. Through its user-friendly interface and extensive feature set, developers can train models to precisely identify and categorize objects, such as products on store shelves or items moving through a checkout line. This functionality empowers the auto billing system to automatically detect items, streamline the checkout process, and ensure precise billing. With Edge Impulse, businesses can harness the efficiency and accuracy of machine learning to optimize their billing systems and enhance the overall customer experience.

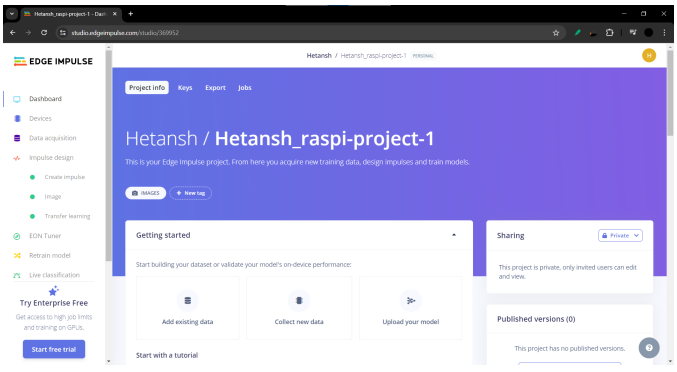

Fig. 11. Edge Impulse Home screen

H. Hardware

The hardware implementation of the implemented system is shown in Fig12.

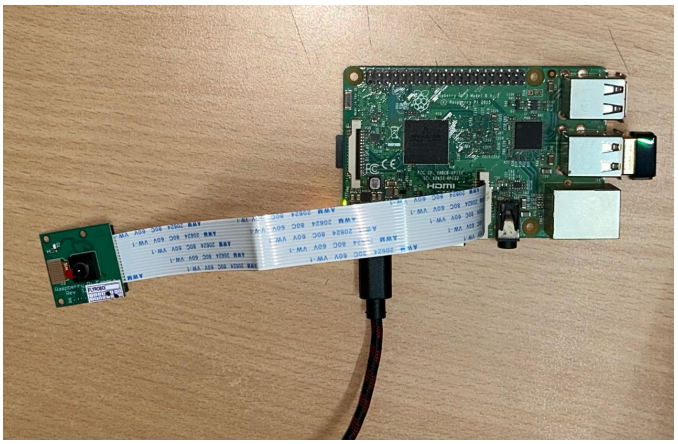

Fig. 12. Hardware

VI. SIMULATION AND RESULTS

Fig13 shows the initial setup on raspberry pi. Objects are identified through raspberry-pi camera and the probabilities of different objects in the dataset are displayed on the screen as shown in Fig14. Fig15 shows the result when pen is placed

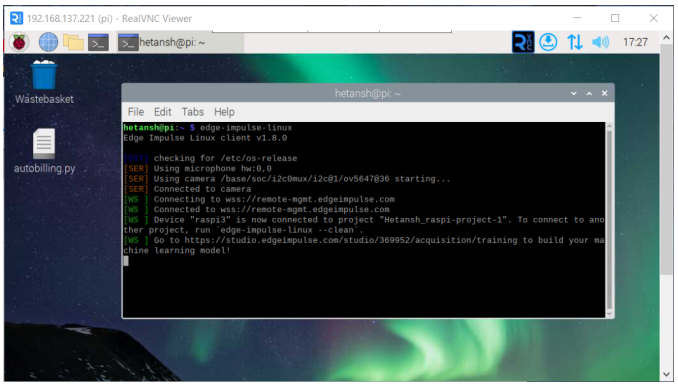

Fig. 13. Rpi- Edge Impulse

in front of camera. The home screen of the user interface is shown in Fig16. When an object is identified, it is added on the screen of user interface with name of the object, quantity of the object, and price of the object as shown in Fig17 and Fig18. When it is processed, a bill of items is produced with the total cost and a UPI payment option as shown in Fig19 and Fig20.

VII. CONCLUSION

The use of the Edge Impulse Platform for deep learning and neural network-based object detection algorithms is an important step toward creating a system that can accurately identify objects in a variety of complex and challenging situations where conventional RFID-based systems might not

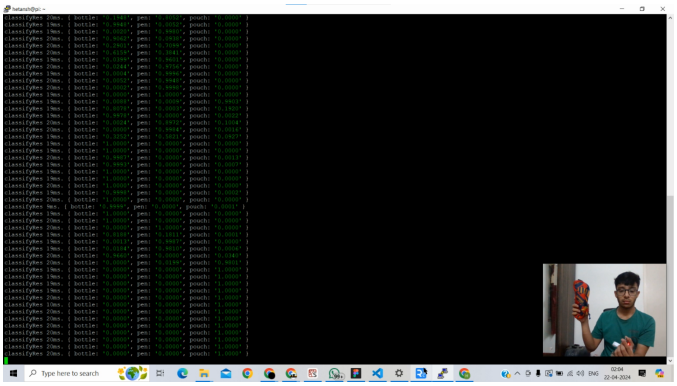

Fig. 14. Object identification

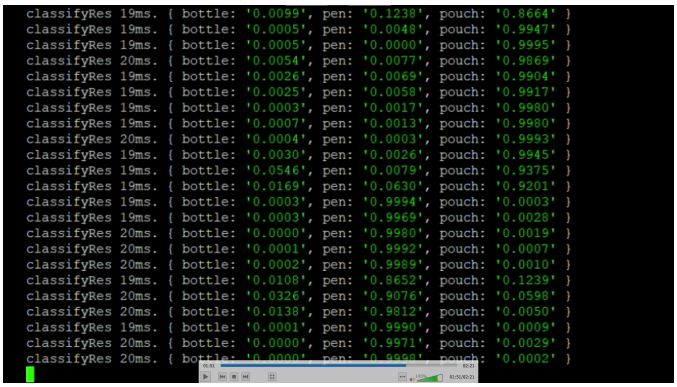

Fig. 15. Pen is detected

be able to perform. This research highlights the potential for machine learning applications to completely transform the shopping experience, emphasizing the necessity of cutting-edge technological solutions in retail contexts. The model after training is loaded in Raspberry pi 3B+. Automated billing and secure QR transactions have advanced with the use of Raspberry Pi 3Bs and machine learning models. The method enables thorough training, optimizing detection accuracy and enhancing the system's capacity to distinguish objects from different angles and magnifications by compiling a varied

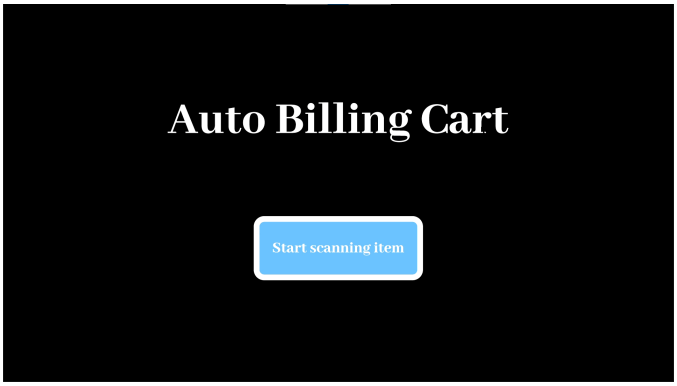

Fig. 16. Home Screen of UI

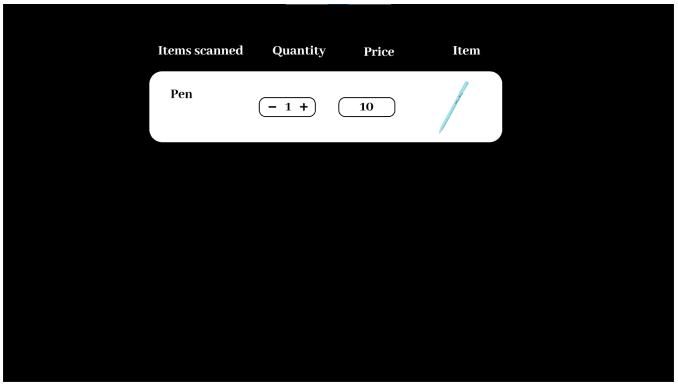

Fig. 17. First object added

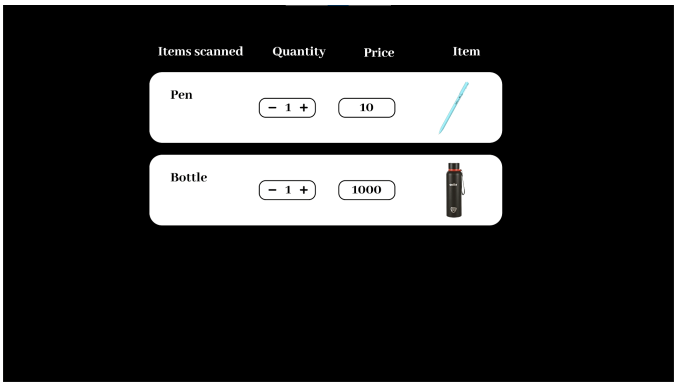

Fig. 18. Second object added

dataset of product photos. This guarantees a flawless end-to-end purchasing experience for customers in addition to improving operational efficiency.

The review of relevant work emphasizes the difficulties and constraints that come with using current technology in retail environments, like RFID, Wi-Fi, and robotic systems. It also underlines the potential obstacles that smaller businesses and merchants with tighter budgets may encounter. This study suggests a more accessible and economical way to improve the retail experience by creating a system that abandons typical

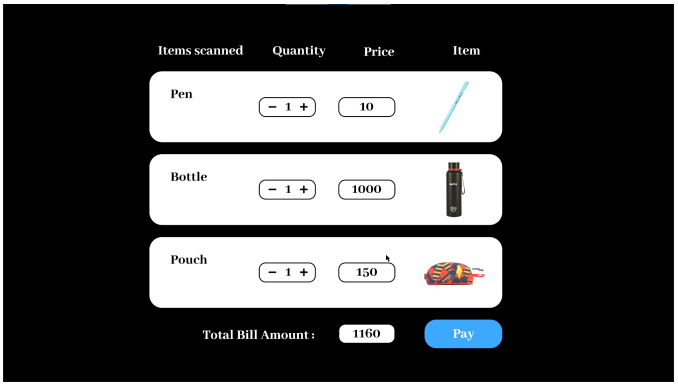

Fig. 19. Third object added

Fig. 20. Payment screen

RFID modules in favor of more flexible object detection algorithms. The User Interface developed shows the bill that is generated along with the total amount and payment option of UPI. The Edge Impulse Platform, Raspberry Pi 3B, object detection algorithms, and user interface working together could revolutionize retail systems by providing shoppers and employees with a smarter, safer, and more effective shopping experience.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to all our batchmates who have contributed to the successful completion of this project, Auto-billing system using Raspberry pi. This endeavor would not have been possible without the unwavering support and guidance of Prof. Viranchi Pandya, whose expertise and encouragement played a pivotal role in shaping the project.

REFERENCES

[1] ”Electronic bill auto-creation system and its visual development software,” in Tsinghua Science and Technology, vol. 6, no. 2, pp. 167-167, June 2001. keywords: Software;Visualization;Java;Consumer electronics;Portable document format;Optimization,

[2] https://studio.edgeimpulse.com/studio/366915

[3] Athauda, T., Marin, J. C. L., Lee, J., Karmakar, N. (2018). Robust low-cost passive UHF RFID based smart shopping trolley. IEEE Journal of Radio Frequency Identification, 1–1. doi:10.1109/jrfid.2018.2866087 10.1109/jrfid.2018.2866087

[4] S. R. Rupanagudi et al., ”A novel video processing based cost effective smart trolley system for supermarkets using FPGA,” 2015 International Conference on Communication, Information Computing Technology (ICCICT), Mumbai, India, 2015, pp. 1-6, doi: 10.1109/ICCICT.2015.7045723. keywords: Image color analysis;Cameras;Radiofrequency identification;Prototypes;Computers;Color;Robots;basket;FPGA;RFID;robotic assistant;shopping assistant;shopping cart;smart shopping system;smart trolley;supermarket;trolley;video processing,

[5] IoT application on secure Smart Shopping system by Ruinian Li , Tianyi Song , Nicholas Capurso , Jiguo Yu, Jason Couture , and Xiuzhen Cheng - 2017.

[6] Anandakumar, H., Umamaheswari, K. (2017). A bio-inspired swarm intelligence technique for social aware cognitive radio handovers. Computers Electrical Engineering. doi:10.1016/j.compeleceng.2017.09.016 10.1016/j.compeleceng.2017.09.016

[7] D. M. Mary, G. S. Jasmine, V. A. Prakash, R. Charan, N. A. kumar and P. E. Surya, "Social Distance Shopping Using Embedded Based Auto Cart and Android App," 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2021, pp. 1140-1143, doi: 10.1109/ICACCS51430.2021.9441847. keywords: Communication systems;EPROM;Human factors;Social factors;Servers;RFID tags;Diseases;Arduino;RFID Tag and Reader;GSM;Android Application;EEPROM,

[8] https://circuitdigest.com/microcontroller-projects/raspberry-pi-based-autonomous-checkout-system-for-retail-stores

Appendix

Source Code

Main Python Code

```
import cv2
import os
import sys
import signal
import time
from edge_impulse_linux.image import ImageImpulseRunner

import requests
import json
from requests.structures import CaseInsensitiveDict

runner = None
show_camera = True

global id_product
id_product = 1
list_label = []
count = 0
taken = 0

a = ‘Pouch’
b = ‘Bottle’
c = ‘Pen’


def now():
    return round(time.time() * 1000)

def sigint_handler(sig, frame):
    print('Interrupted')
    if (runner):
        runner.stop()
    sys.exit(0)

signal.signal(signal.SIGINT, sigint_handler)

def help():
    print('python classify.py <path_to_model.eim> <Camera port ID, only required when more than 1 camera
is present>')

def post(label, price, final_rate, taken):
    global id_product
    url = "https://automaticbilling.herokuapp.com/product"
    headers = CaseInsensitiveDict()
    headers["Content-Type"] = "application/json"
    data_dict = {"id": id_product, "name": label, "price": price, "units": "units", "taken": taken,
"payable": final_rate}
    data = json.dumps(data_dict)
    resp = requests.post(url, headers=headers, data=data)
    print(resp.status_code)
    id_product += 1
    time.sleep(1)

def list_com(label):
    global count
    global taken
    count += 1
    print('count is', count)
    time.sleep(1)
    if count > 1:
        if list_label[-1] != label:
            print("New Item detected")
            print("Final weight is", list_weight[-1])
            rate(list_label[-2], taken)

def rate(label, taken):
    print("Calculating rate")
    if label == a:
        print("Calculating rate of", label)
        price = 200
        post(label, price, 1, taken)
```

```
    elif label == b:
        print("Calculating rate of", label)
        price = 1000
        post(label, price, 1000, taken)
    else:
        print("Calculating rate of", label)
        price = 10
        post(label, price, 10, taken)

def main(argv):
    try:
        opts, args = getopt.getopt(argv, "h", ["--help"])
    except getopt.GetoptError:
        help()
        sys.exit(2)
    for opt, arg in opts:
        if opt in ('-h', '--help'):
            help()
            sys.exit()

    if len(args) == 0:
        help()
        sys.exit(2)

    model = args[0]

    dir_path = os.path.dirname(os.path.realpath(__file__))
    modelfile = os.path.join(dir_path, model)

    print('MODEL: ' + modelfile)

    with ImageImpulseRunner(modelfile) as runner:
        try:
            model_info = runner.init()
            print('Loaded runner for "' + model_info['project']['owner'] + ' / ' +
model_info['project']['name'] + '"')
            labels = model_info['model_parameters']['labels']
            if len(args) >= 2:
                videoCaptureDeviceId = int(args[1])
            else:
                port_ids = get_webcams()
                if len(port_ids) == 0:
                    raise Exception('Cannot find any webcams')
                if len(args) <= 1 and len(port_ids) > 1:
                    raise Exception("Multiple cameras found. Add the camera port ID as a second argument
to use to this script")
                videoCaptureDeviceId = int(port_ids[0])

            camera = cv2.VideoCapture(videoCaptureDeviceId)
            ret = camera.read()[0]
            if ret:
                backendName = camera.getBackendName()
                w = camera.get(3)
                h = camera.get(4)
                print("Camera %s (%s x %s) in port %s selected." % (backendName, h, w,
videoCaptureDeviceId))
                camera.release()
            else:
                raise Exception("Couldn't initialize selected camera.")

            next_frame = 0  # limit to ~10 fps here

            for res, img in runner.classifier(videoCaptureDeviceId):
                if (next_frame > now()):
                    time.sleep((next_frame - now()) / 1000)

                # print('classification runner response', res)

                if "classification" in res["result"].keys():
                    print('Result (%d ms.) ' % (res['timing']['dsp'] + res['timing']['classification']),
end='')
                    for label in labels:
                        score = res['result']['classification'][label]
```

```
                        if score > 0.9:
                            list_com(label)
                            if label == a:
                                print('Apple detected')
                            elif label == b:
                                print('Banana detected')
                            elif label == l:
                                print('Lays detected')
                            else:
                                print('Coke detected')
                    print('', flush=True)
                next_frame = now() + 100
        finally:
            if (runner):
                runner.stop()

if __name__ == "__main__":
    main(sys.argv[1:])
```

UI Code:

Html code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Shopping Cart</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Simple Shopping Cart</h1>
        <div class="items" id="items">
            <div class="item">
                <img src="Bottle.jpg" alt="Item 1">
                <p>Bottle - $1000</p>
                <button class="add-to-cart" onclick="addToCart(1, 'Bottle', 1000)">Add to Cart</button>
            </div>
            <div class="item">
                <img src="Pen.jpg" alt="Item 2">
                <p>Pen - $10</p>
                <button class="add-to-cart" onclick="addToCart(2, 'Pen', 10)">Add to Cart</button>
            </div>
            <div class="item">
                <img src="Pouch.jpg" alt="Item 3">
                <p>Pouch - $200</p>
                <button class="add-to-cart" onclick="addToCart(3, 'Pouch', 200)">Add to Cart</button>
            </div>
        </div>
        <div class="cart" id="cart">
            <h2>Shopping Cart</h2>
            <div id="cart-items"></div>
            <h3>Total: <span id="total-price">$0</span></h3>
            <button onclick="showQR()">Continue</button>
        </div>
        <div class="qr" id="qr" style="display: none;">
            <h2>Scan QR Code</h2>
            <img src="QrCode.jpg" alt="QR Code" id="qr-code">
            <p>Thank you for shopping!</p>
        </div>
        <div class="thank-you" id="thank-you" style="display: none;">
            <h2>Thank you for shopping!</h2>
        </div>
    </div>
```

```
    <script src="script.js"></script>
</body>
</html>
```

JS code:

```
let cartItems = [];
let total = 0;

function addToCart(id, name, price) {
    let existingItem = cartItems.find(item => item.id === id);
    if (existingItem) {
        existingItem.quantity++;
    } else {
        cartItems.push({ id, name, price, quantity: 1 });
    }
    total += price;
    updateCart();
}

function updateCart() {
    let cartItemsDiv = document.getElementById("cart-items");
    cartItemsDiv.innerHTML = "";
    for (let item of cartItems) {
        let itemDiv = document.createElement("div");
        itemDiv.innerText = `${item.name} - $${item.price} x ${item.quantity}`;
        cartItemsDiv.appendChild(itemDiv);
    }
    document.getElementById("total-price").innerText = "$" + total;
}

function showQR() {
    document.getElementById("cart").style.display = "none";
    document.getElementById("qr").style.display = "block";
    document.getElementById("items").style.display = "none";
    setTimeout(() => {
        document.getElementById("qr").style.display = "none";
        document.getElementById("thank-you").style.display = "block";
    }, 15000); // Hide QR code after 15 seconds
}
```

Css code:

```
/* styles.css */

@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(-20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

.container {
    max-width: 800px;
    margin: 0 auto;
    animation: fadeIn 0.5s ease-out; /* Faster fadeIn animation for .container */
```

```
}

.items {
    display: flex;
    justify-content: space-around;
    margin-bottom: 20px;
    animation: fadeIn 0.5s ease-out; /* Faster fadeIn animation for .items */
}

.item {
    text-align: center;
    animation: fadeIn 0.5s ease-out; /* Faster fadeIn animation for .item */
}

.item img {
    max-width: 100px;
    margin-bottom: 10px;
}

.cart, .bill, .qr {
    border: 1px solid #ccc;
    padding: 10px;
    margin-bottom: 20px;
    animation: fadeIn 0.5s ease-out; /* Faster fadeIn animation for .cart, .bill, .qr */
}

.cart, .bill, .qr button {
    display: block;
    margin: 10px auto;
    padding: 10px 20px;
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    transition: background-color 0.3s ease; /* Add transition effect to button hover */
}

.cart button:hover, .bill button:hover, .qr button:hover {
    background-color: #0056b3;
}
```