Self-Checkout Kiosk

Table of Contents

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

In today's retail landscape, the demand for efficient and convenient checkout solutions continues to grow. Traditional cashier-assisted checkout systems face challenges such as long queues, human errors, and limited operational hours. To address these issues and enhance the shopping experience for customers, there is a need for innovative self-checkout solutions.

This project aims to develop a self-checkout kiosk specifically designed for fruits and vegetables. The self-checkout kiosk offers customers a convenient and seamless way to purchase fresh produce without the need for cashier assistance. By leveraging advanced technologies such as object detection, load cell integration, and user-friendly interface design, the self-checkout kiosk streamlines the checkout process, reduces waiting times, and improves overall customer satisfaction.

The key objectives of the project include:

- Training an object detection model capable of accurately identifying fruits and vegetables in real-time.

- Integrating a load cell with the self-checkout kiosk to measure the weight of produce and calculate pricing based on weight.

- Developing a user-friendly interface that guides customers through the checkout process smoothly and efficiently.

By achieving these objectives, the self-checkout kiosk aims to revolutionize the way customers purchase fruits and vegetables, offering a convenient, time-saving, and hygienic alternative to traditional checkout methods. Additionally, the project contributes to the advancement of retail technology and provides insights into the feasibility and effectiveness of self-checkout solutions in improving operational efficiency and customer experience.

CHAPTER 2
REVIEW OF LITERATURE

YOLO(You Only Look Once) is an emerging and very efficient model for object detection. Hence, a lot of research is being done on it and the idea of an automated smart billing system for supermarkets, vegetable stores etc.
Real Time Object Detection with Yolo by Geetha Priya S, N DuraiMurugan, SP chokkalingam(2019).In this paper, they have proposed a YOLO algorithm for the detection of objects using only one neural network. This paper also mentions the advantages that Yolo has over other Object detection algorithms and also concludes that YOLO is a much more efficient and fastest algorithm to use in real time.

Exploring the determinants of intention to use self-checkout systems in super market chain and its application by Ufuk Cebeci, Abdullah Ertug and Hulya Turkcan(2020) investigated the determinants of intention to use self-checkout systems in a supermarket chain. Their study employs the Technology Acceptance Model (TAM) to analyze factors such as technology anxiety, self-efficacy, compatibility, and knowledge, providing valuable insights into customer attitudes and behaviours towards self-checkout technology.

Intelligent Billing system using Object Detection by Neeraj Chidella, N Kalyan Reddy, N Sai Dheeraj Reddy, Maddi Mohan, Joydeep Sengupta(2022). The paper presents an automatic billing system for fruits and vegetables utilizing fine-tuned Convolutional Neural Networks (CNNs) and YOLO object detection. Their project emphasizes real-time object detection and billing, tackling the hurdles related to manually entering barcode data for perishable items. Designing a Self-Payment Cashier For Bakeries Using YOLO V4 by Henrique I. Wang(2021) .The paper presents an automation project tailored for bakery stores, employing the YOLOv4 object detector. A self-checkout cashier system was developed, wherein the customer places a tray with products under a webcam, and the system calculates the total purchase price. However, the system encounters challenges in accurately recognizing products when they are in close proximity. Kavan Patel proposed a self-checkout portal in supermarkets using the YOLO algorithm but he failed to complete the implementation of this model. Another paper written by Huimin Yuan and Ming Yan proposed an intelligent billing system using cascade R-CNN.

After all this literature survey, we found out that many authors attempted to develop smart checkout systems utilizing deep learning techniques. However, they encountered challenges, particularly in achieving satisfactory results for fresh produce. Therefore, in this paper, we propose a self-checkout system tailored for fresh produce retail, incorporating a load cell for accurate weight calculation to streamline the checkout process.

CHAPTER 3
METHODOLOGY

3.1 Data Collection and Annotation:

The initial phase of our project centred around gathering and annotating a comprehensive dataset of fruits and vegetables. However, sourcing such a dataset, especially for Indian fruits and vegetables in a top view perspective, can be challenging due to the specific requirements of the project. Here's how we addressed the dataset collection and annotation process:

3.1.1 Dataset Collection:

- Due to the unavailability of a suitable dataset that met our project requirements, we embarked on collecting our own dataset.

- We employed a systematic approach to capture images of Indian fruits and vegetables. This involved using high-resolution cameras to photograph fruits and vegetables from various angles and perspectives.

- 636 images of 14 fruits and vegetables in total were captured under different lighting conditions to ensure robustness and adaptability of the model to varying environments.

CLASS	NAME	NO OF IMAGES
0	Apple	60
1	Beetroot	41
2	Brinjal	60
3	Brinjal White	30
4	Capsicum	15
5	Chilly	17
6	Coconut	90
7	Guava	30
8	Lemon	39
9	Mosambi	60
10	Onion	40
11	Pomegranate	30
12	Potato	60
13	Tomato	64

Table 3.1: No of images in each class

3.1.2 Annotation Process:

- Annotation is a pivotal and meticulous process during the training of object detection models because it requires accurately labeling each individual object within the images with its specific class category and the precise coordinates of its bounding box, which are essential for the model to learn and accurately identify objects in new and unseen images.

- We utilized the MakeSense.ai annotation website for annotating our dataset. MakeSense provided user-friendly interface for annotating images, enabling efficient labeling of objects.

- Every single image included in the dataset underwent a meticulous annotation process with the primary goal of guaranteeing precise localization of various types of fruits and vegetables.

- Moreover, in our meticulous approach, we took great care to verify that the annotations remained uniform throughout the dataset, strictly following the specified format essential for preparing the YOLO model for effective training.

3.1.3 Dataset Splitting:

- After completing the annotation process, the dataset underwent a division to separate it into distinct training and testing sets. This division was essential in order to streamline the subsequent procedures of model training and evaluation effectively.

- To establish an effective training regimen, we implemented an 80-20 split ratio for our annotated images. This strategy involved allotting 80% of the images for the YOLO model's training phase, while reserving the remaining 20% specifically for evaluating and testing how well the model performed in various scenarios and conditions.

By collecting and annotating our dataset, we ensured the availability of high-quality training data tailored to the specific requirements of our self-checkout system. This comprehensive dataset serves as the foundation for training the YOLO model to accurately detect and classify Indian fruits and vegetables, thereby enabling seamless and efficient checkout experiences for customers.

3.2 Object Detection:

3.2.1 Working of YOLO:

The YOLO (You Only Look Once) algorithm represents a pivotal advancement in object detection within computer vision. At its core, YOLO operates by segmenting the input image into a grid, typically with dimensions \(n \times n\). The selection of \(n\) is contingent upon factors such as image complexity, aiming to strike a balance between granularity and computational efficiency. Each grid cell undergoes a dual process of classification and localization, wherein the algorithm endeavors to discern the presence of objects and accurately delineate their spatial extents. Through this approach, YOLO simultaneously predicts bounding boxes and assigns class probabilities for objects within each grid cell. This holistic

strategy streamlines the detection process, offering real-time performance ideal for diverse applications ranging from autonomous driving to surveillance systems.

Central to YOLO's functionality is the computation of an "objectness" score for each grid cell. This metric serves as a binary indicator of whether an object occupies the respective grid cell. When an object is absent, the objectness score assumes a value of zero, signifying the absence of significant content within the cell. Consequently, the associated bounding box coordinates are also nullified, reflecting the absence of any meaningful object localization. Conversely, in instances where an object is present within a grid cell, the objectness score is set to 1, denoting the presence of salient content. In this scenario, the algorithm proceeds to generate bounding box coordinates based on the object's spatial properties, facilitating precise localization. By effectively distinguishing between occupied and unoccupied grid cells, YOLO optimizes computational resources by focusing processing efforts exclusively on regions of interest. This targeted approach enhances efficiency while maintaining robust object detection performance across diverse scenarios.

The concept of anchor boxes further enhances YOLO's capacity to detect multiple objects within a single grid cell. These predetermined bounding box shapes aid in capturing variations in object sizes and aspect ratios, enabling the algorithm to accommodate a diverse array of objects within a unified framework. Through the utilization of anchor boxes, YOLO extends its applicability to scenarios characterized by the simultaneous presence of multiple objects, fortifying its versatility in real-world deployment.

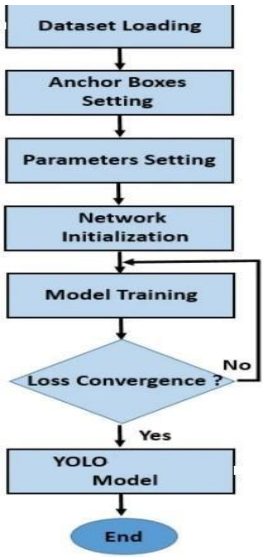

Fig 3.1 Model Training

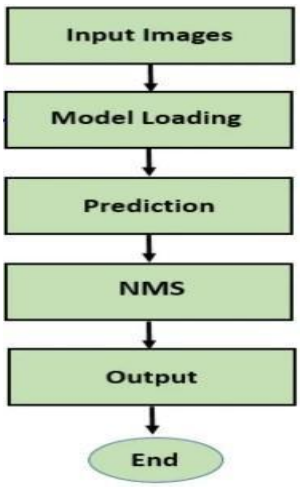

Fig 3.2 Detection

In scenarios where multiple grids contain the same object, ensuring accurate detection necessitates strategic measures such as employing IOU (Intersection over Union) and Non-Max Suppression techniques. IOU evaluation entails calculating the intersection and union areas of two bounding boxes, subsequently deriving the IOU value using the formula \(IOU = \frac{{\text{Area of Intersection}}}{{\text{Area of Union}}}\). This metric serves as a measure of overlap between bounding boxes, aiding in discerning the most accurate

representations of objects. By setting a threshold value, typically determined based on object characteristics and spatial proximity, bounding boxes with IOU values below this threshold are suppressed, refining the object detection process.

Non-Max Suppression serves as a complementary strategy to IOU evaluation, prioritizing bounding boxes with higher objectness scores while suppressing redundant or overlapping detections. This iterative process iteratively selects the most salient bounding boxes until only the most accurate representations remain. By iteratively refining bounding box selections based on both objectness scores and IOU values, Non-Max Suppression optimizes object detection accuracy in scenarios with multiple overlapping detections.

Moreover, to extend object detection capabilities beyond a single object per grid cell, anchor boxes are instrumental. These predefined bounding box shapes encapsulate information regarding object presence and spatial characteristics within grid cells. Each grid cell is associated with one or more anchor boxes, enabling the detection of multiple objects within a single grid cell. As the number of objects within a cell varies, the dimensions and properties of anchor boxes adjust accordingly, ensuring adaptability to diverse object configurations.

Incorporating anchor boxes into the detection pipeline enhances YOLO's versatility, enabling robust detection of multiple objects within a single grid cell. By leveraging anchor boxes in conjunction with IOU evaluation and Non-Max Suppression techniques, YOLO excels in accurately detecting and localizing objects across diverse scenarios, facilitating real-world deployment in applications ranging from autonomous vehicles to surveillance systems.

3.2.2 Training:

For our self-checkout system designed to streamline the purchasing process of fruits and vegetables, we employed the YOLOv5 specifically the version X architecture provided by Ultralytics. This selection was driven by the reputation of YOLOv5 for its superior performance in real-time object detection tasks, aligning seamlessly with our objective of streamlining the checkout process. Leveraging the computational prowess of Google Colab, we capitalized on its GPU resources to efficiently train our model without the need for extensive hardware investments. Through meticulous dataset curation and annotation, we cultivated a comprehensive collection of Indian fruits and vegetables images, meticulously labeled with bounding boxes and class labels. This meticulously curated dataset served as the bedrock for training the YOLOv5 version X model, enabling it to accurately detect and classify items during the checkout process. With a carefully chosen batch size of 16 and training over 40 epochs, we aimed to strike a balance between model accuracy and computational efficiency, ensuring optimal performance without compromising on speed.

3.2.3 Interface development:

To create an intuitive and user-friendly self-checkout system for fruits and vegetables, we developed a Tkinter-based graphical user interface (GUI) that seamlessly guides users

through the checkout process. The GUI comprises three main windows: the Start Window, the Main Window, and the Checkout Window.

Start Window:

Upon launching the application, users are greeted with the Start Window, a meticulously designed interface that immediately captures their attention. The Start Window presents a visually appealing layout featuring vibrant colors and intuitive navigation elements, including a prominently displayed "Start" button that beckons users to take the next step. Upon eagerly pressing the "Start" button, users are seamlessly transitioned to the Main Window, marking the beginning of a streamlined and user-friendly checkout process that ensures a seamless and engaging user experience.

Fig 3.3 Start Window

Main Window:

The Main Window serves as the central hub of the self-checkout system. Here, users can place the desired fresh produce in the tray under the camera by pressing the "Add Items to the Kiosk" button. Upon activation, the system initiates the capture of a picture, facilitating real-time object detection of fruits and vegetables. The detected label is then matched with the corresponding entry in a CSV file containing the names of fresh produce and their respective prices per kilogram. Simultaneously, the system fetches the weight of fresh produce from the load cell, allowing for accurate pricing calculation based on weight. The identified items, along with their quantity and price, are dynamically displayed in the interface, providing users with a clear overview of their selections. Furthermore, users can effortlessly remove items by selecting the item and pressing the "Remove" button. The total price of the purchase is prominently displayed. Once users have completed their shopping, they can proceed to the checkout process by pressing the "Checkout" button.

No	name	cost
1	Apple	240
2	Beetroot	50
3	Brinjal	40
4	Brinjal_w	40
5	Capsicum	120
6	Chilly	34
7	Coconut	80
8	Guava	80
9	Lemon	90
10	Mosambi	140
11	Onion	25
12	Pomegranate	210
13	Potato	60
14	Tomato	35

Table 3.2 Cost per kg table in CSV file

Fig 3.4 Main Window

Checkout Window:

The Checkout Window marks the culmination of the self-checkout process, presenting users with a seamless and efficient checkout experience. Upon navigating to the Checkout Window, users are greeted with a QR code, simplifying the payment process. By scanning the QR code with their preferred payment method, users can swiftly complete their transaction, thus concluding their shopping journey with ease and convenience.

Fig 3.5 Checkout window

3.2.4 Hardware Integration:

To build self-checkout kiosk for fruits and vegetables, we seamlessly integrated various hardware components to enhance the functionality and efficiency of the system. The core hardware components utilized in our setup include a 5kg load cell, HX711 amplifier, and ESP8266 microcontroller and laptop.

We built the structure of the kiosk using MDF boards and we mounter a Logitech C270 webcam on top of the kiosk.

Fig 3.6 Kiosk

Load Cell and HX711 Amplifier:

The heart of our system's weight measurement capability lies in the utilization of a 5kg load cell in conjunction with an HX711 amplifier. The load cell serves as the sensor responsible for detecting weight variations, while the HX711 amplifier facilitates accurate analog-to-digital conversion, enabling precise weight measurements to be captured by the system. This combination of components ensures reliable and accurate weight sensing, critical for determining the quantity and pricing of fruits and vegetables during the checkout process.

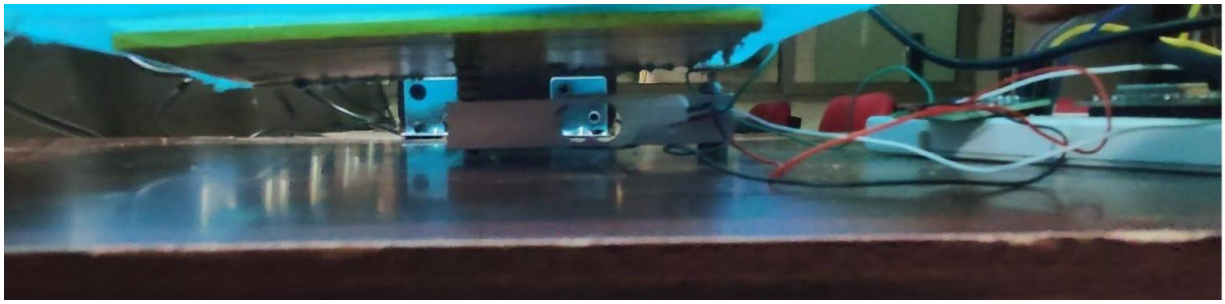

Fig 3.7 Load cell

ESP8266 Microcontroller:

To enable seamless communication between the hardware components and our software system, we leveraged the ESP8266 microcontroller. The ESP8266 is a versatile and cost-effective microcontroller equipped with built-in Wi-Fi capabilities, making it an ideal choice for IoT (Internet of Things) applications. In our setup, the ESP8266 connects to our Wi-Fi network and creates a local host server, awaiting requests from the Python script running on our system. Upon receiving a request, the ESP8266 executes the desired code to measure the weight using the load cell and HX711 amplifier. Subsequently, it returns the weight measurement as an HTTP request, allowing the Python script to retrieve and process the data seamlessly. This integration streamlines the communication between the hardware and software components, facilitating efficient operation of the self-checkout system.

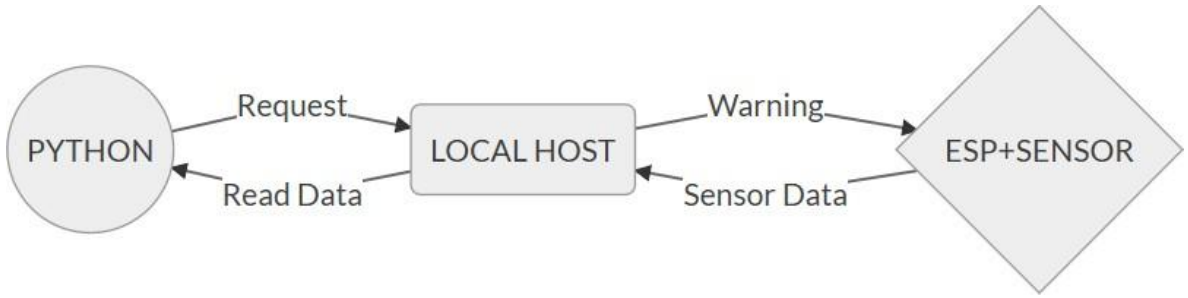

Fig 3.8 Communication between ESP8266 and Python script
*Source:*https://www.instructables.com/ESP8266-and-Python-Communication-ForNoobs/

By integrating the 5kg load cell, HX711 amplifier, and ESP8266 microcontroller into our hardware setup, we have enhanced the functionality and capabilities of our self-checkout system. This robust integration ensures accurate weight measurement and efficient communication between hardware and software components, laying the foundation for a seamless and reliable checkout experience for users.

CHAPTER 4
CODE AND EXPLANATION

Code creates a functional GUI for a self-checkout kiosk system, allowing users to add items to their cart, view the total cost, and complete the payment process. The script is made with three classes:

Start Class: This class represents the start page of the GUI. It creates a window with a canvas and displays a text message ("SELF CHECKOUT KIOSK"). It also includes buttons for navigation to the main and payment pages.

Main_window Class: This class represents the main page of the GUI where users can add items to their cart. It captures images from a camera, processes them using the YOLOv5 model, identifies the objects, fetches their prices from a CSV file, and displays them in a treeview widget along with their quantities and prices. Users can remove items from the cart and proceed to the payment page.

Payment Class: This class represents the payment page of the GUI. It displays a QR code for payment and includes buttons for returning to the start page or completing the payment and returning to the start page.

```
import tkinter as tk
from pathlib import Path
from tkinter import *
from tkinter import ttk
import cv2
from PIL import Image, ImageTk
from tkinter import messagebox
import subprocess
import csv
import torch
from contextlib import contextmanager
import pathlib
import urllib.request
import numpy as np



OUTPUT_PATH = Path(__file__).parent
Start_ASSETS_PATH = OUTPUT_PATH / Path(r"E:/Project/GUI TWO
FRAME/build/start_page/assets")
Main_ASSETS_PATH = OUTPUT_PATH / Path(r"E:/Project/GUI TWO
FRAME/build/main_page/assets")
Payment_ASSETS_PATH = OUTPUT_PATH / Path(r"E:/Project/GUI TWO
FRAME/build/payment_page/assets")

save_location = "E:/Project/Major project/Results/takpic/"

def start_relative_to_assets(path: str) -> Path:
```

```
    return Start_ASSETS_PATH / Path(path)
def main_relative_to_assets(path: str) -> Path:
    return Main_ASSETS_PATH / Path(path)
def payment_relative_to_assets(path: str) -> Path:
    return Payment_ASSETS_PATH / Path(path)

count = 0

camera = cv2.VideoCapture(0)
image_counter = 0
l1=[]
url = "http://192.168.43.92/"
EXPORT_PATH = pathlib.Path('C:/Users/gowth/Downloads/Yolov5x/best.pt')
model = torch.hub.load('E:/Project/Major project/yolov5', 'custom',path =
EXPORT_PATH,source='local')


root = tk.Tk()
root.withdraw()

def mainWindow():
    Main_window()

def paymentWindow():
    Payment()

total_cost = 0
class Start(Toplevel):
    def __init__(self, *args, **kwargs):
        Toplevel.__init__(self, *args, **kwargs)

        self.geometry("1200x600+100+100")
        self.configure(bg = "#FFFFFF")

        self.canvas =
            Canvas(self,
            bg = "#FFFFFF",
             height = 600,
            width = 1200,
            bd = 0,
            highlightthickness = 0,
            relief = "ridge"
        )

        self.canvas.place(x = 0, y = 0)

        self.canvas.create_text(
            205.0,
            206.0,
```

```
            anchor="nw",
            text="SELF CHECKOUT KIOSK",
            fill="#0C22E7",
            font=("MontserratRoman Black", 64 * -1)
        )

        button_image_1 =
PhotoImage(file=start_relative_to_assets("button_1.png"))
        self.start_button =
            Button(self.canvas,
            image=button_image_1,
            borderwidth=0,
            highlightthickness=0,
            command=mainWindow,
            relief="flat"
        )
        self.start_button.place(x=430.0,y=332.0,width=354.0,height=79.48387145
99609)

        self.resizable(False, False)
        self.mainloop()

class Main_window(Toplevel):
    def __init__(self, *args, **kwargs):
        Toplevel.__init__(self, *args, **kwargs)

        self.geometry("1200x600+100+100")
        self.configure(bg = "#FFFFFF")

        self.canvas = Canvas(self,bg = "#FFFFFF",height = 601,width = 1200,bd
= 0,highlightthickness = 0,relief = "ridge")
        self.canvas.place(x = 0, y = 0)
        self.canvas.create_text(376.0000114440918,11.999996606200284,anchor="n
w",text="PLACE THE ITEM IN KIOSK",fill="#5E95FF",font=("Montserrat Bold", 26 *
-1))
        button_image_1 =
            PhotoImage( file=main_relative_to_assets("button_1.png"))
        self.add_button =
            Button( self.canvas,
            image=button_image_1,
            borderwidth=0,
            highlightthickness=0,
            command=self.capture_image,
            relief="flat"
        )
        self.add_button.place( x=
            333.00000190734863,
            y=62.99998812696492,
```

```
            width=460.1075148164746,
            height=48.59711299145067
        )
        button_image_2 =
            PhotoImage( file=main_relative_to_assets("button_2.png"))
        self.checkout_button =
            Button(self.canvas,
            image=button_image_2,
            borderwidth=0,
            highlightthickness=0,
            command=paymentWindow,
            relief="flat"
        )
        self.checkout_button.place(
            x=897.0,
            y=459.97490410971204,
            width=193.3779095123282,
            height=48.25088963565668
        )

        self.my_tree = ttk.Treeview(self)

        self.my_tree['columns'] = ("Product", "QNTY (in grams)", "Price (in
Rs)")
        self.my_tree.column("#0", width=0, stretch=NO)
        self.my_tree.column("Product", anchor=W, width=120, minwidth=25)
        self.my_tree.column("QNTY (in grams)", anchor=CENTER, width=80,
minwidth=25)
        self.my_tree.column("Price (in Rs)", anchor=CENTER, width=120,
minwidth=25)

        self.my_tree.heading("#0", text="", anchor=W)
        self.my_tree.heading("Product", text="Product", anchor=W)
        self.my_tree.heading("QNTY (in grams)", text="QNTY", anchor=CENTER)
        self.my_tree.heading("Price (in Rs)", text="Price", anchor=CENTER)

        self.my_tree.place(x=90.0, y=131.0, width=1000, height=300.0)
        #self.data1 = [["Tomato", "1 kg", "30rs"], ["Mosambi", "2 kg",
"30rs"], ["Apple", "3 kg", "30rs"]]
        self.data = []
        self.data_cpy = []
        button_image_3 =
            PhotoImage( file=main_relative_to_assets("button_3.png"))
        self.remove_button =
            Button(self.canvas,
            image=button_image_3,
            borderwidth=0,
            highlightthickness=0,
```

```
            command=self.remove,
            relief="flat"
        )
        self.remove_button.place( x=5
            0.0, y=495.97653283280226,
            width=229.3640397614563,
            height=48.297600756316456
        )
        button_image_4 =
            PhotoImage( file=main_relative_to_assets("button_4.png"))
        self.cancel_button =
            Button(self.canvas,
            image=button_image_4,
            borderwidth=0,
            highlightthickness=0,
            command=self.destroy1,
            relief="flat"
        )

        self.cancel_button.place( x=9
            04.0, y=533.9844326365036,
            width=186.03715636059133,
            height=48.241361108865135
        )

        self.count = 0
        self.resizable(False, False)
        self.mainloop()

    def destroy1(self):
            global total_cost
            total_cost = 0
            self.destroy()

    def remove(self):
        x = self.my_tree.selection()
        #print(int(x[0]))
        self.data_cpy = [item for item in self.data_cpy if item[0] !=
int(x[0])]
        #print(self.data_cpy)
        global total_cost
        total_cost = 0
        for cost in self.data_cpy:
            total_cost += cost[3]
            #print(total_cost)
        #print(total_cost)
```

```
        self.update_total_cost()
        for record in x:
            self.my_tree.delete(record)

    def get_data(self):
        self.n = urllib.request.urlopen(url).read()
        self.n = self.n.decode("utf-8")
        return self.n

    def  update_total_cost(self):
        self.canvas.delete("total_cost_text")
        self.canvas.create_text(398.99999237060547, 508.0000271237784,
anchor="nw",
                                text="TOTAL AMOUNT : " + str(total_cost) +"
Rs", fill="#5E95FF",
                                font=("Montserrat Bold", 26 * -1),
tag="total_cost_text")

    def capture_image(self):

        ret, frame = camera.read()
        self.image_name = f"{save_location}image.jpg"
        self.im = f"{save_location}image1.jpg"

        cv2.imwrite(self.image_name, frame)
        im = self.image_name
        results = model(im)
        results.save()

        df=results.pandas().xyxy[0]
        count_df = df['name'].value_counts().reset_index()
        count_df.columns = ['name', 'Count']
        max_index = count_df['Count'].idxmax()
        max_name = count_df.loc[max_index, 'name']


        l=[]
        l1 = []
        st=max_name
        l1.append(self.count)
        l.append(max_name)
        l1.append(max_name)
        ld=self.get_data()

        l.append(ld)
        l1.append(ld)
        filename="C:/Users/gowth/Downloads/my_table4.csv”
```

```
            r=csv.reader(file)
            for i in r:
                if i[1]==st:

                    cost = int(ld)*int(i[2])*0.001
                    cost=round(cost,2)
                    global total_cost
                    total_cost += cost
                    total_cost=round(total_cost,2)

                    l.append(str(cost))
                    l1.append(cost)

        self.data.append(l)
        self.data_cpy.append(l1)

        self.update_total_cost()
        self.my_tree.insert(parent='',index='end',iid=self.count,text="Parent"
,
                            values=self.data[self.count])

        self.count +=1

class Payment(Toplevel):
    def __init__(self, *args, **kwargs):
        Toplevel.__init__(self, *args, **kwargs)

        self.geometry("1200x600+100+100")
        self.configure(bg = "#FFFFFF")

        self.canvas =
            Canvas(self,
            bg = "#FFFFFF",
             height = 600,
            width = 1200,
            bd = 0,
            highlightthickness = 0,
            relief = "ridge"
        )

        self.canvas.place(x = 0, y = 0)

        self.canvas.create_text(
            357.0,
            9.000013355083865,
            anchor="nw",
            text="PAY HERE",
            fill="#5E95FF",
```

```
            font=("Montserrat Bold", 96 * -1)
        )

        qr_image_1 =
PhotoImage(file=payment_relative_to_assets("image_1.png"))
        image_1 = self.canvas.create_image(
                            599.0,
                            323.0,
                            image=qr_image_1
                        )
        button_image_1 =
PhotoImage(file=payment_relative_to_assets("button_1.png"))
        self.back_button =
            Button(self.canvas,
            image=button_image_1,
            borderwidth=0,
            highlightthickness=0,
            command=self.destroy,
            relief="flat"
        )
        self.back_button.place( x=83.
            0, y=536.000017621813,
            width=211.26927952970442,
            height=48.83606246040631
        )

        button_image_2 =
PhotoImage(file=payment_relative_to_assets("button_2.png"))
        self.finish_button =
            Button(self.canvas,
            image=button_image_2,
            borderwidth=0,
            highlightthickness=0,
            command=self.destroyed,
            relief="flat"
        )
        self.finish_button.place( x=8
            69.0, y=532.000017621813,
            width=211.26855010077998,
            height=48.27411224574075
        )

        self.resizable(False, False)
        self.mainloop()

    def destroyed(self):
```

```
        global total_cost
        total_cost = 0
        self.destroy()
        Start()

Start()
```

4.1 Functions

Main_window.capture_image():

- This method is called when the user clicks the "Add" button on the main window interface.
- It captures an image from the camera, processes it using the YOLOv5 model to detect objects, retrieves their names, and fetches their prices from a CSV file.
- The detected objects along with their quantities and prices are added to the cart displayed in the treeview widget on the main window interface.

Main_window.remove():

- This method is called when the user clicks the "Remove" button on the main window interface.
- It removes the selected item from the cart displayed in the treeview widget and updates the total cost accordingly.

Main_window.update_total_cost():

- This method updates the total cost displayed on the main window interface after adding or removing items from the cart.

Payment.destroyed():

- This method is called when the user clicks the "Finish" button on the payment window interface.
- It resets the total cost to zero and closes the payment window, returning the user to the start window.

These functions encapsulate various functionalities such as image processing, cart management, and interface navigation in the self-checkout kiosk system. They collectively enable the system to capture, process, and display information to the user effectively.

CHAPTER 5

RESULT

5.1 YOLOv5

After training the YOLOv5 model for 40 epochs, the following results were achieved: a box loss of 0.022737, an object loss of 0.029617, and a class loss of 0.0015752 on the training dataset. The model demonstrated a precision of 91.997%, a recall of 91.9%, and anmAP of 92.08% at a confidence threshold of 0.5. The YOLOv5 model achieved high precision and recall values, indicating strong performance in both detecting and classifying objects. Moreover, the mean average precision (mAP) at an IoU threshold of 0.5 reached 0.9208, showcasing the model's accuracy in localizing objects with a moderate overlap. It is worth noting that the learning rate during training remained constant at 0.000595, ensuring the model's stability and consistent learning trajectory. Overall, these results underscore the effectiveness and reliability of the YOLOv5 model in accurately detecting, classifying, and localizing objects in diverse scenarios.

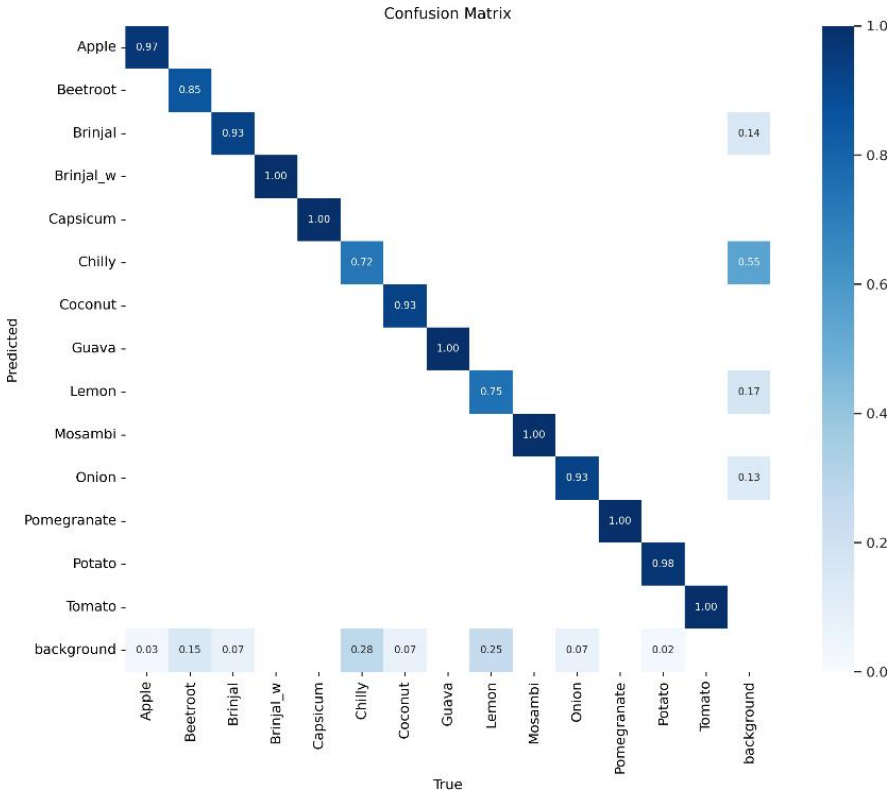

Fig 5.1 Confusion matrix

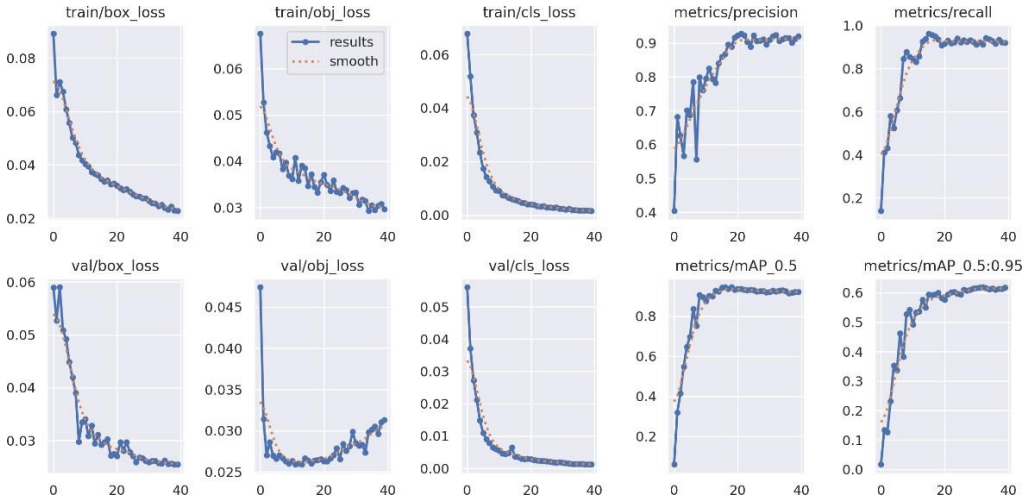

Fig 5.2 Result Graphs

5.2 Detection output

The detection output from the YOLOv5 model showcases its effectiveness in accurately identifying and classifying fruits and vegetables within the provided dataset. With a precision of 91.997% and a recall of 91.9%, the model demonstrates its robustness in detecting objects across various classes. The mean average precision (mAP) of 92.08% at a confidence threshold of 0.5 further underscores the model's ability to reliably detect objects with high confidence. While the mAP drops slightly to 61.736% for a broader confidence range of 0.5 to 0.95, the model still maintains a considerable level of accuracy. This detection output reaffirms the suitability of the YOLOv5 model for integration into the self-checkout kiosk system, ensuring efficient and reliable identification of items for seamless checkout experiences.

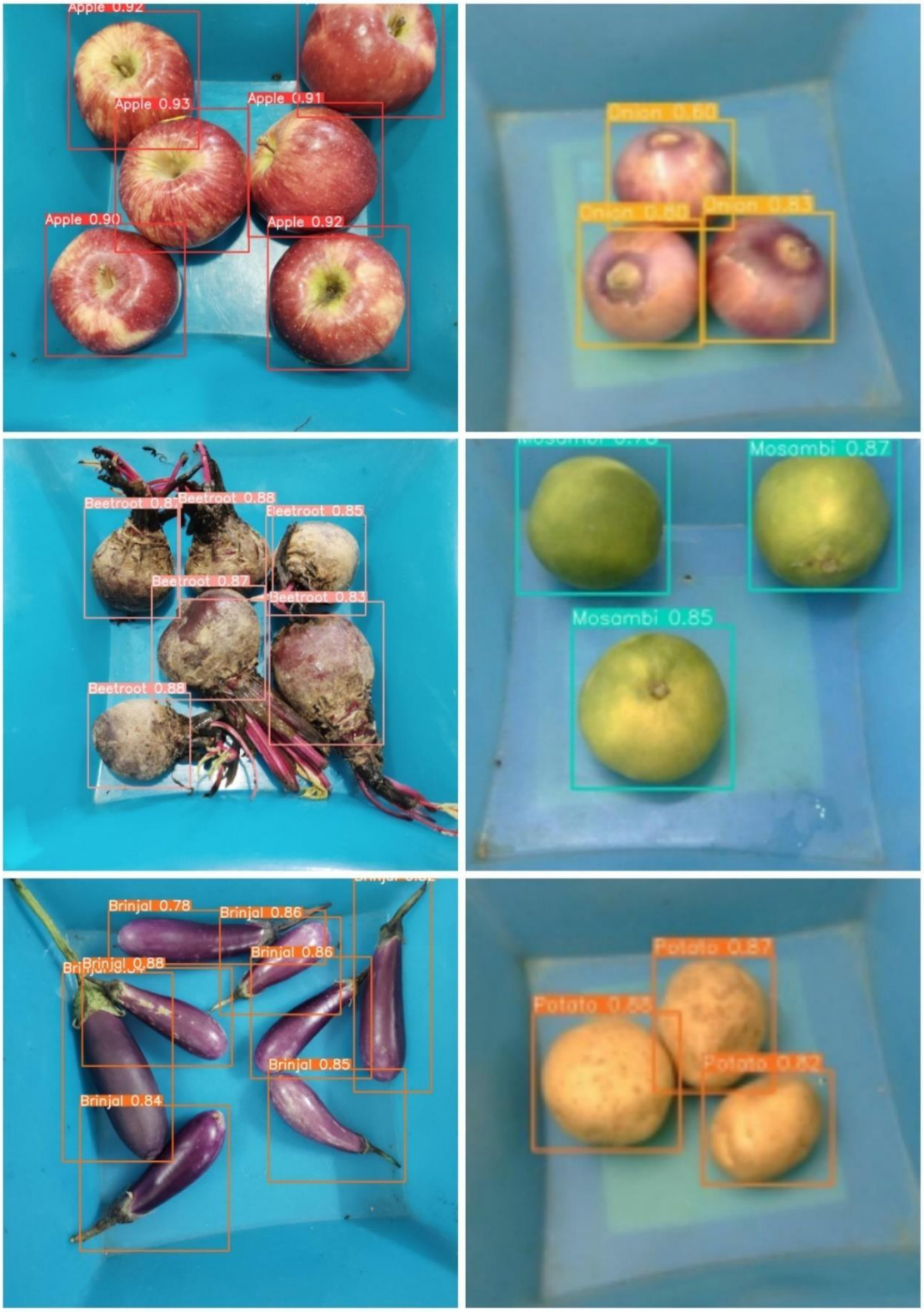

Fig 5.3 Detection Outputs

5.3 Interface

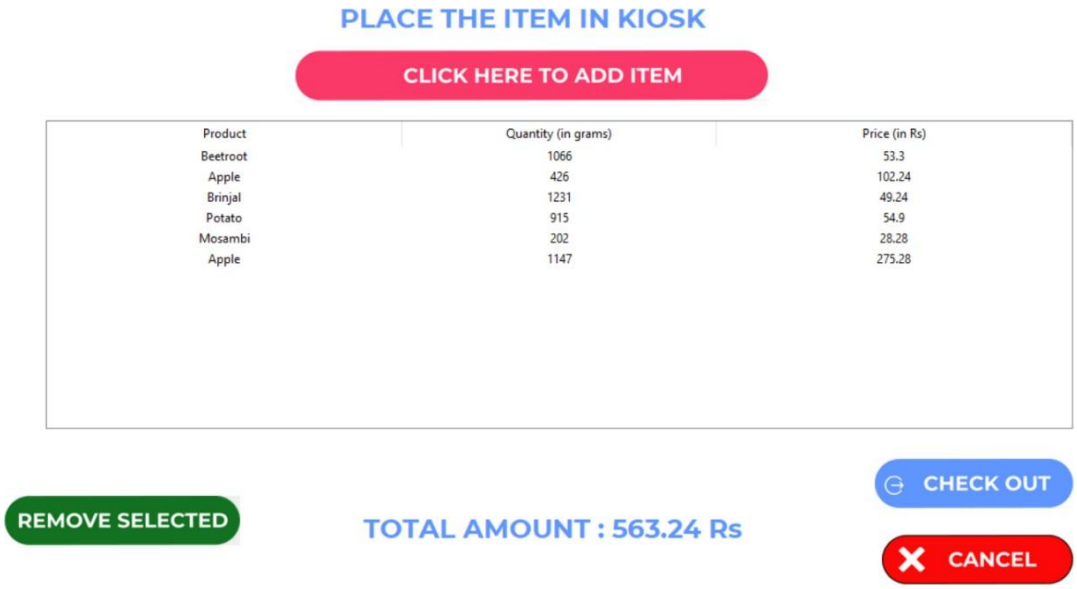

Fig 5.4 Checkout Overview

The displayed interface in the figure illustrates the main window of the system, showcasing the fruits and vegetables added by the user one by one. Each item is accurately weighed using the load cell, displaying the quantity in grams. The corresponding price of each item and the total checkout amount are also presented, providing a comprehensive overview of the items to be processed.

CHAPTER 6

CONCLUSION

The project successfully addresses the inefficiencies of traditional checkout systems by implementing an intelligent self-checkout kiosk for fruits and vegetables. Leveraging YOLO object detection and Tkinter interface, the system efficiently identifies items with a commendable accuracy rate of 93%, reducing reliance on manual scanning. Integration of a calibrated load cell with an ESP for weight measurement further enhances the checkout process, incorporating weight-based pricing seamlessly. Despite challenges such as dataset scarcity and hardware limitations, the project underscores a significant stride towards automating retail checkout processes, promising enhanced efficiency and convenience for customers while demonstrating the viability of advanced technologies in retail automation.

6.1 Future work

In future developments, our focus will revolve around training our model to detect a broader range of Indian fruits and vegetables, ensuring adaptability to diverse regional preferences. Transitioning our self-checkout system to run on Raspberry Pi will enhance accessibility, particularly in resource-constrained environments. Integration of a secure payment gateway directly into the interface aims to streamline transactions, while implementing a feedback loop mechanism allows for continuous improvement based on customer insights. These enhancements collectively aim to elevate the functionality, accessibility, and user experience of our self-checkout system, setting new standards for efficiency and convenience in retail environments.

CHAPTER 7

REFERENCES

1) Geethapriya. S, N. Duraimurugan, S.P. Chokkalingam. “Real-Time object detection with yolo”. International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249– 8958, Volume-8, Issue 3S, February 2019.
2) N. Chidella, N. K. Reddy, N. S. D. Reddy, M. Mohan and J. Sengupta, "Intelligent Billing system using Object Detection," 2022 1st International Conference on the Paradigm Shifts in Communication, Embedded Systems, Machine Learning and Signal Processing (PCEMS), Nagpur, India, 2022.
3) Chengji Liu,Yufan Tao, Jiawei Liang, Kai Li ,Yihang Chen. “Object detection based on YOLO network”. 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC 2018).
4) Kavan Patel. “Fruits and vegetable detection for POS with Deep Learning”.
5) ParthJadhav “Tkinter-Designer”.
6) N. Chidella, N. K. Reddy, N. S. D. Reddy, M. Mohan and J. Sengupta, "Intelligent Billing system using Object Detection," 2022 1st International Conference on the Paradigm Shifts in Communication, Embedded Systems, Machine Learning and Signal Processing (PCEMS), Nagpur, India, 2022.
7) B. -F. Wu, W. -J. Tseng, Y. -S. Chen, S. -J. Yao and P. -J. Chang, "An intelligent self-checkout system for smart retail," *2016 International Conference on System Science and Engineering (ICSSE)*, Puli, Taiwan, 2016.
8) V. P. Chaudhari, R. V. Chaudhari, A. S. Burgul and S. A. Jain, "Smart Self-Checkout System for Supermarkets," *2023 IEEE 3rd International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET)*, Mysuru, India, 2023.
9) H. I. Wang, L. K. Miyazaki, M. S. Falheiro and M. S. G. Tsuzuki, "Designing a Self-Payment Cashier For Bakeries Using YOLO V4," *2021 14th IEEE International Conference on Industry Applications (INDUSCON)*, São Paulo, Brazil, 2021
10) N. M. Krishna, R. Y. Reddy, M. S. C. Reddy, K. P. Madhav and G. Sudham, "Object Detection and Tracking Using Yolo," *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*, Coimbatore, India, 2021