A Industrial Project

Report on

FOOD DETECTION AND CALORIE ESTIMATOR

Submitted to

Jawaharlal Nehru Technological University, Hyderabad

Submitted in partial fulfillment of requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted By

D. SRUTHI	**226P1A0544**
G. SNIGDHA	**226P1A0555**
D. BHAVYA	**226P1A0540**
B. HASIKA	**226P1A0514**

Under the guidance of

Dr. K. Rama Krishna, Ph.D

Assistant professor Department of CSE

Department of Computer Science and Engineering

Rishi MS Institute of Engineering and Technology for Women

(Accredited by NAAC with 'A' Grade)

Approved by AICTE, Affiliated to JNTUH

Nizampet Cross Road, near JNTUH, Kukatpally, Hyderabad, Telangana 500085

2024-2025

Rishi M.S. Institute of Engineering & Technology for Women

(Approved by A.I.C.T.E., & Affiliated to J.N.T.U.H.)
(In Memory of **"BHARAT RATNA" Mrs. M.S. Subbulakshmi**)
Near J.N.T.U.H Metro Station, Nizampet 'X' Road, Kukatpally, Hyderabad - 500 085.
E-mail: rishims2009@gmail.com, Phone: 040-23892878, Fax: 040-23892858.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "FOOD DETECTION AND CALORIE ESTIMATOR" being submitted by

D. SRUTHI	**226P1A0544**
G. SNIGDHA	**226P1A0555**
D. BHAVYA	**226P1A0540**
B. HASIKA	**226P1A0514**

In partial fulfillment of the award of degree of BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE & ENGINEERING from Rishi MS Institute of Engineering & Technology for women, affiliated to the Jawaharlal Nehru Technological University, Hyderabad during the year 2024-2025.Thisis a record of bonafied piece of work, undertaken by the supervision of the undersigned.

Under the Guidance of :
Dr. K. Rama Krishna, Ph.D.
Assistant Professor,
Department of CSE

Head of the Department :
Dr. Archana Patil, B. Tech, M. Tech, Ph.D.
Assistant Professor & HOD,
Department of CSE

Submitted for Viva Voice Examination held on______________________

External Examiner

ACKNOWLEDGEMENT

This report will certainly not be completed without due acknowledgements paid to all those have helped us in doing our Industrial Project work.

We would like to express our sincere thanks to our Secretary Ms**. Rajasree**, & Academic Director **Mrs. Madhavilatha** for their constant supervision and encouragement, which helped us in completing this work successfully.

We are expressing our sincere gratitude to our Principal **Dr. K R N Kiran Kumar** for his timely suggestions, which helped us to complete this work successfully.

It's our privilege to thank **Dr. Archana Patil**, Head of the Department for her encouragement during the progress of this work.

We derive great pleasure in expressing our sincere gratitude to our Project Co-ordinator **Mrs. Sowjanya & Mrs. Tejaswini**, for their encouragement and timely suggestions during the progress of this work.

We express our sincere thanks to our Guide **Dr. K. Rama Krishna**, for giving us moral support, kind attention and valuable guidance to us throughout this work.

We are thankful to our **Rishi MS Institute of Engineering and Technology for Women** for providing the required facilities during the Industrial Project work.

We would like to thank our parents and our friends for being supportive all the time, and we are very much obliged to them.

226P1A0544 SRUTHI

226P1A0555 SNIGDHA

226P1A0540 BHAVYA

226P1A0514 HASIKA

Department of Computer Science and Engineering

Vision & Mission of the Department

Vision of the Department

- To promote center of excellence in sustainable academic environment to impart technical skills and strong research platform to prepare global leaders

Mission of the Department

- To create a good academic platform with modern teaching and learning methodologies.
- To enhance leadership qualities with ethics, values, and a sense of teamwork for professional excellence.
- To inculcate research culture by encouraging projects in advanced technologies through industry interactions.

PROGRAM OUT COMES (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and teamwork**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

FOOD DETECTION AND CALORIE ESTIMATOR

ABSTRACT

In recent years, the rise in lifestyle-related diseases such as obesity and diabetes has highlighted the need for effective dietary monitoring tools. This project proposes a system that leverages computer vision and machine learning techniques for automated food detection and calorie estimation. The primary objective is to develop a model capable of identifying various food items from images and estimating their caloric content based on visual features and portion size. The system employs a convolutional neural network (CNN) for food classification, trained on a diverse dataset containing labeled food images. Once identified, the system calculates the approximate caloric value using a nutritional database that maps each food class to its average calorie content per unit weight. In advanced implementations, depth estimation or image segmentation techniques are used to assess portion size, enhancing the accuracy of calorie estimation. This approach can significantly assist individuals in tracking their daily calorie intake with minimal manual input, promoting healthier eating habits. It also holds potential for integration into mobile applications, wearable devices, and healthcare monitoring systems. Future improvements could include real-time video analysis, multi-food plate detection, and personalized calorie recommendations based on user profiles. Food calorie estimation is a crucial aspect of nutrition tracking, aiding individuals in making informed dietary choices. This project presents a web-based food detection and calorie estimation system using computer vision and machine learning. The system utilizes YOLOv8 for food item detection and XGBoost regression for calorie prediction. Users can upload images or use a webcam to detect food items, after which the system overlays bounding boxes and calorie information on the detected items. A pre-defined dataset of food items and their nutritional values serves as the basis for predictions. The application is built with Streamlit, ensuring a user-friendly interface for real-time food analysis. By integrating machine learning and deep learning, this project enhances automated nutrition assessment, offering a practical tool for dietary monitoring in health-conscious applications. Future improvements may include custom food dataset training, portion size estimation, and multi-food recognition optimization.

LIST OF FIGURES

LIST OF ACRONYMS

SNO	**Acronyms**	**Description**
1	ST	Streamlit
2	CV2	OpenCV
3	NP	NumPy
4	XGB	XGBoost
5	YOLO	You Only Look Once
6	PIL	Python Imaging Library
7	PT	PyTorch
8	UML	Unified Modified Language
9	URL	Uniform Resource Locator
10	DFD	Data Flow Diagram

CHAPTER 1

INTRODUCTION

In today's digital age, automated nutrition assessment has gained significant traction, helping individuals track their dietary intake more efficiently. This project introduces a food detection and calorie estimation system that leverages computer vision and machine learning to analyze food items in images or real-time webcam feeds. By integrating YOLOv8, a state-of-the-art object detection model, and XGBoost regression, the system accurately predicts the calorie content of identified food items.

The application is built using Streamlit, enabling an interactive and user-friendly interface for seamless food analysis. Users can either upload food images or use their webcam, and the system will detect food items, overlay bounding boxes, and estimate calorie values. The underlying model is trained on a predefined nutritional dataset, ensuring reliable and meaningful estimations.

This solution aims to simplify dietary monitoring and empower health-conscious users with instant nutritional insights, paving the way for smarter eating habits and personalized dietary tracking. Future enhancements could include custom food dataset training, portion size estimation, and optimization for multi-food recognition, making the application even more robust and versatile.

In the era of smart technology and digital health solutions, tracking food intake has become increasingly important for maintaining balanced nutrition and healthier lifestyles. Traditional methods of calorie counting often involve manual logging, which can be time-consuming and prone to errors. To address this challenge, this project introduces a real-time food detection and calorie estimation system powered by computer vision and machine learning algorithms.

The system employs YOLOv8, a deep learning model known for its fast and accurate object detection capabilities, to identify food items within images or live webcam feeds. Once a food item is detected, its nutritional values—such as calorie count, carbohydrates, protein, and fat are estimated using an XGBoost regression model. The application is designed using Streamlit, a lightweight and interactive framework that enhances the user experience by providing an intuitive

1.1 PROBLEM STATEMENT

Objective

The goal of this project is to develop a real-time or image-based application that can detect food items in an image (or via webcam), identify their type**,** and estimate their calorie content using machine learning and computer vision techniques.

Problem Description

In today’s health-conscious society, tracking nutritional intake is essential. However, manual logging of food consumption is tedious and error-prone. This application aims to automate this process by:

1. Detecting food items in an image or live video stream using the YOLOv8 object detection model.
2. Estimating calorie content of the detected food items using an XGBoost regression model trained on nutritional data.
3. Providing a user-friendly interface using Streamlit for uploading images or using a webcam.

- **YOLOv8-based object detection**: Detects various food items like "apple", "pizza", "burger", etc.
- **XGBoost regression model**: Predicts the caloric value based on nutritional features like carbs, protein, fat, and sugar.
- **Live webcam integration**: Real-time food recognition and calorie display.
- **Image upload support**: Users can analyze static images as an alternative to live webcam.

Technical Approach

- **Input**: An image or real-time video frame.
- **Processing**:
  - Food detection using YOLOv8.
  - Calorie prediction using an XGBoost model trained on food nutrition data.
- **Output**: The image/frame annotated with bounding boxes around detected food items and corresponding calorie estimates.

Technologies Used

- Python
- OpenCV

- Streamlit (for UI)
- Ultralytics YOLOv8 (for food detection)
- XGBoost (for calorie prediction)
- NumPy & PIL

Use Cases

- Diet tracking applications
- Health and fitness monitoring tools
- Nutrition assistance for individuals with dietary restrictions
- Educational tools for food awareness

1.2 SCOPE OF RESEARCH

The scope of research for a project on food detection and calorie estimation involves defining the boundaries, objectives, and potential applications of the system. Here's a structured breakdown of the scope, which is often included in academic theses, research proposals, or system documentation:

1. Research Objective

To develop and evaluate a system that can automatically detect food items from images and estimate their caloric content, thereby assisting users in dietary monitoring and health management.

2. Scope of Research

A. Food Detection

- **Focus**: Develop or integrate deep learning models capable of identifying various food items in an image.
- **In-Scope**:
  - Object detection using models like YOLO, Faster R-CNN, or SSD
  - Food classification using pre-trained or custom-trained CNN models
  - Datasets such as Food-101, UECFOOD-256, or curated local datasets
- **Out-of-Scope**:
  - Detection of non-food items
  - Real-time video detection (unless specified)

B. Portion Size & Volume Estimation

- **Focus**: Estimate food portion sizes to improve calorie calculation accuracy.
- **In-Scope**:
  - Volume estimation using image cues or depth data (if available)

    - Use of fiducial markers (e.g., coin or spoon) for scale
- **Out-of-Scope**:
    - 3D modeling from multi-view imaging (unless depth sensors are used)

C. Calorie Estimation

- **Focus**: Map recognized food items and estimated portion sizes to their nutritional content.
- **In-Scope**:
    - Use of databases (e.g., USDA Food Data Central, Nutrition ix)
    - Rule-based or machine learning models for calorie prediction
- **Out-of-Scope**:
    - Micronutrient breakdown (unless part of expanded research)
    - Estimation of homemade or complex dishes without known composition

D. System Implementation & Evaluation

Focus: Build a prototype system (mobile/web/desktop) and evaluate performance.

- **In-Scope**:
    - System architecture design
    - Accuracy testing for food detection and calorie estimation
    - User feedback integration
- **Out-of-Scope**:
    - Commercial deployment
    - Large-scale field testing (unless included in pilot study)

3. Research Questions

Some example research questions might include:

- How accurately can food items be detected in real-world settings?
- What is the impact of portion estimation accuracy on calorie prediction?
- Can the system adapt to diverse cuisines and food types?

4. Expected Outcomes

- A working prototype capable of detecting multiple food items from a single image
- A calorie estimation module with acceptable error margins

- Evaluation metrics such as precision, recall, F1-score for detection, and MAE/RMSE for calorie prediction

5. Potential Applications

- Personal diet tracking apps
- Hospital or elderly care nutrition monitoring
- Smart kitchen or IoT-enabled dietary management
- Health insurance wellness programs

1.3 EXISTING SYSTEM

In the current landscape of food calorie estimation, traditional methods rely on manual input, where users log their meals into calorie-tracking applications. These methods require users to accurately identify food items and enter portion sizes, leading to potential errors in estimation. Existing systems for food detection and calorie estimation primarily include mobile applications like Calorie Mama and MyFitnessPal, as well as research prototypes such as MIT's Im2Calories. While apps like Calorie Mama use deep learning to identify food items from images, they are limited by fixed datasets and often struggle with complex or mixed dishes. MyFitnessPal, though popular, relies heavily on manual input and barcode scanning, offering only limited image recognition capabilities. Academic research often uses curated datasets like Food-101 to train convolutional neural networks, but these models fail to generalize well in real-world scenarios with cluttered backgrounds and varied lighting. Advanced systems like Im2Calories offer volume-based calorie estimation but require depth data or multiple images, making them impractical for everyday use. Additionally, while YOLO-based models have been explored for real-time food detection, pre-trained versions are not optimized for food-specific categories and thus require custom training for accurate results. Overall, existing solutions lack the ability to perform accurate, real-time food recognition and calorie estimation in diverse, real-world conditions.

1.3.1 DISADVANTAGES OF EXISTING SYSTEM

1. Limited Accuracy in Food Detection

- **Challenge**: Many models struggle to distinguish visually similar foods (e.g., curry vs. stew).
- **Cause**: Lack of high-quality, diverse training data for regional or homem ade dishes.
- **Impact**: Reduces reliability of detection, especially for multi-item meals.

2. Inaccurate Portion Size Estimation

- **Challenge**: Estimating food volume or weight from a 2D image is difficult.

- **Cause**:
    - No standard reference object in the image
    - Lack of depth information (3D perspective)
- **Impact**: Leads to significant calorie miscalculations (±30–50% or more).

3. Dependency on User Input

- **Challenge**: Users may need to confirm or correct results manually.
- **Cause**: System lacks contextual awareness or confidence scoring.
- **Impact**: Reduces automation and adds user burden.

4. Poor Generalization Across Cuisines

- **Challenge**: Models trained on Western datasets may not recognize global or local dishes.
- **Cause**: Food datasets (like Food-101) are often biased toward limited cuisine types.
- **Impact**: System is less useful for users from diverse cultural backgrounds.

5. High Computational Requirements

- **Challenge**: Real-time detection and estimation may require GPU or server processing.
- **Cause**: Use of heavy deep learning models (YOLO, ResNet, etc.)
- **Impact**: Difficult to deploy on low-power mobile devices; increased cost on cloud.

6. Limited Integration with Health Data

- **Challenge**: Existing systems often don't adapt calorie estimates to personal health profiles.
- **Cause**: Lack of user-specific context (activity level, metabolic rate, goals).
- **Impact**: Calorie suggestions may be generic and less actionable.

7. No Feedback Learning Loop

- **Challenge**: System does not improve from user corrections over time.
- **Cause**: Absence of active learning or continuous training mechanism.
- **Impact**: System accuracy stagnates unless retrained manually.

8. Internet Dependency

- **Challenge**: Some systems require continuous online access to process images or query food databases.
- **Cause**: Backend-hosted ML models and APIs
- **Impact**: Limits usability in offline or low-connectivity areas.

9. Basic Calorie Estimation Methods

- **Challenge**: Calorie estimation is often rule-based and not data-driven.
- **Cause**: Simplistic models that assume fixed serving sizes
- **Impact**: Doesn't account for variations in cooking methods, ingredients, or serving sizes.

1.4 PROPOSED SYSTEM

The proposed system aims to overcome the limitations of existing food detection and calorie estimation platforms by integrating advanced deep learning models with intelligent nutritional analysis. The system will allow users to capture or upload images of their meals, from which food items will be automatically detected using an object detection model such as YOLOv8 or Efficient Det. A custom-trained classification model, built on diverse datasets like Food-101 and UECFOOD-256, will then identify the specific types of food items present. To address the common challenge of inaccurate portion estimation, the system will implement a volume estimation module using visual scaling techniques, such as including a known reference object (e.g., a spoon or coin) in the image. Detected foods and estimated portion sizes will be mapped to a comprehensive nutritional database—such as the USDA Food Data Central or Nutrition ix API—to provide users with a precise calorie count and macronutrient breakdown. Additionally, the system will be designed to learn and improve over time through a feedback loop, where users can confirm or correct food identifications, contributing to continuous model refinement. To ensure personalized dietary tracking, the system will integrate user profiles that consider individual goals, preferences, and health data. The backend will support cloud-based inference for high performance, with lightweight alternatives (e.g., TensorFlow Lite) for on-device operation. The proposed system aims to deliver a more accurate, culturally inclusive, and user-friendly tool for dietary monitoring, offering real-time feedback and reducing the burden of manual calorie tracking.

1.4.1 ADVANTAGES OF PROPOSED SYSTEM

1. Improved Accuracy in Food Detection

The proposed system leverages advanced object detection and classification models (e.g., YOLOv8, Efficient Det) trained on diverse datasets. This ensures more accurate recognition of a wide range of food items, even in complex or cluttered images.

2. Enhanced Portion Size Estimation

By incorporating visual scaling techniques and allowing for the inclusion of reference objects in images, the system significantly improves the accuracy of portion size estimation. This leads to more reliable calorie predictions compared to systems that assume fixed serving sizes.

3. Cultural and Dietary Inclusiveness

The proposed system will be trained on an expanded dataset that includes foods from various cultures and regions. This allows it to recognize a wider variety of dishes beyond Western cuisine, making it suitable for global and diverse user bases.

4. Automated Calorie and Nutrient Estimation

The integration of food databases like USDA Food Data Central and Nutritionix enables the system to automatically compute not just calories, but also macronutrients (carbohydrates, proteins, fats) and, optionally, micronutrients. This eliminates the need for manual logging.

5. User Feedback Loop for Continuous Learning

Unlike static systems, the proposed solution includes a feedback mechanism where users can validate or correct the system’s output. This feedback will be used to retrain and fine-tune models, increasing accuracy over time.

6. Personalized Dietary Tracking

By incorporating user profiles that store personal health data (e.g., age, weight, dietary preferences), the system can tailor nutritional suggestions and calorie estimates, making it more relevant and useful for individual users.

7. Cross-Platform Support

The system is designed to work across devices, with a lightweight version for mobile applications (using TensorFlow Lite or ONNX) and a more powerful cloud-based version for high-performance processing. This ensures usability even in resource-constrained environments.

8. Real-Time Feedback and Visualization

Users receive instant feedback on their food intake, along with easy-to-understand visual summaries such as calorie charts and nutritional breakdowns. This improves user engagement and helps support healthier dietary choices.

9. Data Privacy and Offline Functionality

The proposed system can optionally perform image processing and estimation locally on the device without sending images to the cloud, addressing privacy concerns and enabling offline usage when internet connectivity is limited.

1.5 ORGANIZATION OF REPORT

This report is organized into ten chapters to systematically present the development of the proposed food detection and calorie estimation system. Chapter 1 introduces the background of the study, outlines the problem statement, defines the objectives, scope, and significance of the research, and briefly describes the structure of the report. Chapter 2 provides a comprehensive literature review, highlighting existing methods for food detection and calorie estimation, along with a critical analysis of current systems and their limitations. Chapter 3 focuses on system analysis, detailing the key components of the existing system, user requirements, and identified shortcomings that justify the proposed solution. Chapter 4 presents the proposed system in detail, including its architecture, functional modules, and key advantages over conventional methods. Chapter 5 outlines the methodology adopted for model development, data preprocessing, algorithm selection, and system design. Chapter 6 discusses the implementation process, showcasing the development environment, tools used, interface design, and integration of machine learning models. Chapter 7 provides an evaluation of the system's performance using standard metrics such as accuracy and error rates, and includes comparisons with existing systems. Chapter 8 concludes the report with a summary of findings, system limitations, and suggestions for future enhancements. Chapter 9 contains the list of references used throughout the report. Finally, Chapter 10 includes appendices with additional materials such as code snippets, sample datasets, and interface screenshots.

CHAPTER 2

LITERATURE SURVEY

- **2021 3rd International Conference on Signal Processing and Communication (ICPSC)**--- V Balaji Kasyap; N. Jayapandian --- his method is implementing to calculate the food calorie with the help of Convolutional Neural Network. The input of this calculated model is taken an image of food. The food calorie value is calculated the proposed CNN model with the help of food object detection. The primary parameter of the result is taken by volume error estimation and secondary parameter is calorie error estimation.

- **2021 International Conference on Emerging Smart Computing and Informatics (ESCI)---** Calorie meter: Food Calorie Estimation using Machine Learning --- it estimates the calories of each food and experimental studies have shown that by providing production with information of calories and nutrients present in the food, the proposed estimation method is effective.

- **2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings---**Parisa Pouladzadeh; Pallavi Kuhad; Sri Vijay Bharat Peddi; Abdulsalam Yassine; Shervin Shir Mohammadi All--Accurate methods to measure food and energy intake are crucial for the battle against obesity. Providing users/patients with convenient and intelligent solutions that help them measure their food intake and collect dietary information are the most valuable insights toward long-term prevention and successful treatment programs

- **2024 Second International Conference on Advances in Information Technology (ICAIT)** ---Mohammad Haris --- In our research paper, we have introduced a neural network-based model designed to predict food items from images or pictures and provide estimated calorie information. To achieve this, we gathered a dataset consisting of approximately 101,000 images spanning 101 different food categories and we achieved the results namely a Recall value of 0.779, F1 score of 0.776, and Precision value of 0.777. Our approach involved training a Convolutional Neural Network (CNN) with features extracted using the ResNet50 model, which is a transfer learning model, resulting in a model with an accuracy of 77.97%. The deployment of our system is accessible through a webpage. People can upload an image of food items, and in real-time, our system will predict the identified food item along with its estimated calorie content.

- **2023 Hybrid Deep Learning Algorithm-Based Food Recognition and Calorie Estimation** --- Tanupriya Choudhury --- Every individual requires some sort of system that informs them about portions and calories of food, as well as providing them with directions on how to consume it. In our study, we propose a hybrid architecture that makes use of deep learning algorithms to forecast the number of calories in various food items on a bowl. This consists of three major components: segmentation, classification, and calculating the volume and calories of food items. When we use a Mask RCNN, the images are first segmented. Using the YOLO V5 framework, features are collected from the segmented images and the food item is categorized. In order to determine the dimensions of each food item, we identify the items first. In order to calculate the quantity of the food item, the estimated dimension must be used. The calories are then computed using the food item’s volume. The aforementioned approaches, which were trained on the dataset’s food images, that correctly identified and forecasted a food item’s calories had an accuracy of 97.12%.

CHAPTER 3

METHODOLOGIES

3.1 SYSTEM ARCHITECTURE

1. **User**: Initiates the process by uploading a food image or using a webcam.
2. **Upload Food Image/Webcam**: Captures or receives the image to be processed.
3. **Convolutional Neural Network(CNN)**:The core engine for analyzing the food image. Within CNN, the following steps occur:

- **Image Acquisition**: Captures the raw food image input.
- **Pre-Processing**: Cleans, resizes, normalizes, or enhances the image for better analysis.
- **Food Detection**: Identifies whether food is present and localizes it within the image.
- **Food Recognition**: Classifies the type of food (e.g., apple, pizza).

4. Calorie Estimation Algorithm: Based on recognized food, it uses a predefined database or regression model to estimate the calories.

5. Predicted Food Calorie: Shows the estimated calorie value (e.g., "Apple: 52.0 kcal").

6. Output Screen: Displays the food name and its calorie value over the image for the user.

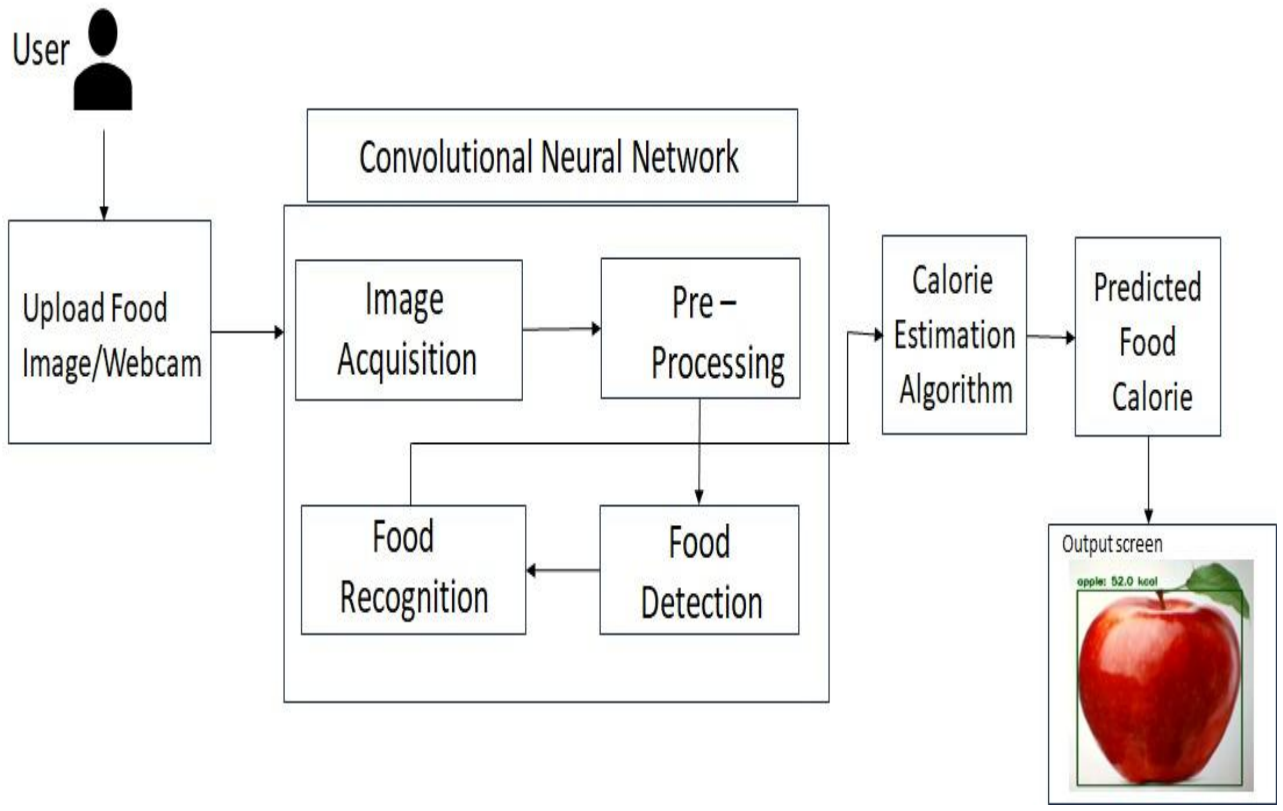


Figure 3.1.3: System Architecture

3.3ALGORITHMS

1.Object Detection Using YOLOv8

- YOLOv8 (You Only Look Once) is employed for real-time food item detection in images or webcam streams.
- It detects food items using bounding boxes and labels, ensuring quick identification.

2.Feature-Based Calorie Prediction Using XGBoost

- Nutritional values (calories, carbs, proteins, fats) are used as input features for training an XGBoost regression model.
- The trained model predicts calorie values based on detected food items.

3.Image Processing with OpenCV

- OpenCV is used for handling image input, processing frames, and displaying bounding boxes.
- It converts color formats (BGR to RGB) for compatibility with Stream lit visualization.

4.Interactive Web Application Using Streamlit

- Streamlit enables a user-friendly interface where users can upload images or use a webcam.
- UI components like buttons, toggles, and image display allow seamless interaction.

5.Data Handling & Mapping

- Food items are mapped to predefined nutritional data for structured processing
- Food labels detected by YOLOv8 are used to retrieve corresponding calorie estimates.

CHAPTER 4

SOFTWARE REQUIREMENT SPECIFICATION

1. Functional Requirements

- **Image Input:** The system should allow users to upload food images or use a webcam to capture live frames.
- **Object Detection:** The system should detect food items in images using the YOLOv8 model.
- **Calorie Estimation:** It should estimate the calories of detected food items using a trained XGBoost regressor.
- **User Interface:** A Streamlit-based web interface should provide options for image upload, webcam toggle, and display detection results.
- **Visual Output:** Detected food items should be highlighted with bounding boxes and their estimated calorie values shown on the image.

2. Performance Requirements

- **Detection Time:** The system should process each image in real-time or under 2 seconds for uploaded images and under 1 second per frame for webcam input.
- **Accuracy:** The model should correctly identify common food items with an accuracy of at least 80% on supported classes.
- **Responsiveness:** The UI should remain responsive during inference, especially with webcam streaming.
- **Concurrent Usage:** Designed for single-user real-time use; may require optimization (e.g., batching, GPU) for concurrent users.

3. Software Requirements

- **Python Libraries:**
  - `streamlit`
  - `opencv-python`
  - `numpy`
  - `xgboost`
  - `ultralytics`
  - `Pillow`
- **YOLOv8 Pre-trained Model:**
  - Default `yolov8n.pt` or a custom-trained model on food datasets.
- **Operating System Compatibility:**
  - Cross-platform (Windows, Linux, macOS)

4. Hardware Requirements

- **Minimum:**
  - CPU: Intel i5 or equivalent
  - RAM: 8 GB
  - Webcam (for live detection)
- **Recommended:**
  - GPU: NVIDIA GPU with CUDA support (for faster YOLOv8 inference)
  - RAM: 16 GB or more
  - SSD storage for faster I/O
- **Webcam or Camera Module:** For real-time detection.

5. Languages/Technologies Used

Programming Languages:

1. **Python** – Main language for backend logic, machine learning, and image processing.

Libraries & Frameworks:

1. **OpenCV** – For image capture, processing, and drawing bounding boxes.
2. **NumPy** – For handling numerical operations and arrays.
3. **XGBoost** – For training and predicting calorie values using regression.
4. **PIL (Python Imaging Library)** – For image file loading and manipulation.
5. **Ultralytics YOLOv8** – For food item detection in images and video frames.
6. **Streamlit** – For creating the web-based user interface of the application.

Tools and Platforms:

1. **YOLOv8 Model (Pre-trained)** – For object (food) detection.
2. **Jupyter Notebook / IDEs (e.g., VS Code, PyCharm)** – For development and testing.
3. **Streamlit CLI** – To run the web app locally or deploy on a server.

CHAPTER 5

SYSTEM DESIGN

UML (Unified Modeling Language) is a standardized modeling language used in software engineering to visualize, specify, construct, and document the components of a software system. It helps developers and stakeholders understand, design, and communicate system architecture effectively using diagrams. Developed by the Object Management Group (OMG), UML provides a set of graphical notations and diagrams that help software developers and stakeholders understand system design and architecture. It includes structural diagrams like class, object, and component diagrams to represent the static aspects of a system, as well as behavioral diagrams like use case, sequence, and activity diagrams to depict dynamic interactions and workflows. UML is platform-independent and supports clear communication among team members, aiding in analysis, design, and documentation of complex software projects.

5.2 UML Diagrams:

UML diagrams are broadly classified into two categories:

1. Structural Diagrams (describe static aspects):
    - Class Diagram
    - Object Diagram
    - Component Diagram
    - Deployment Diagram
    - Package Diagram
    - Composite Structure Diagram
2. Behavioral Diagrams (describe dynamic aspects):
    - Use Case Diagram
    - Sequence Diagram
    - Activity Diagram
    - State Machine Diagram
    - Communication Diagram
    - Interaction Overview Diagram
    - Timing Diagram

5.2.1 USE CASE DIAGRAM

The use case diagram represents the interactions between the user and the food calorie estimation system. The user can perform various actions such as capturing or uploading a food image, which initiates the detection and recognition process. The system then identifies the food items and draws bounding boxes around them. Following recognition, the system estimates the calories based on the detected food. The user can also view and display both the recognized food items and the corresponding calorie values. Each of these interactions outlines a functional requirement of the system, showing how the user engages with the features for effective calorie estimation and visualization.

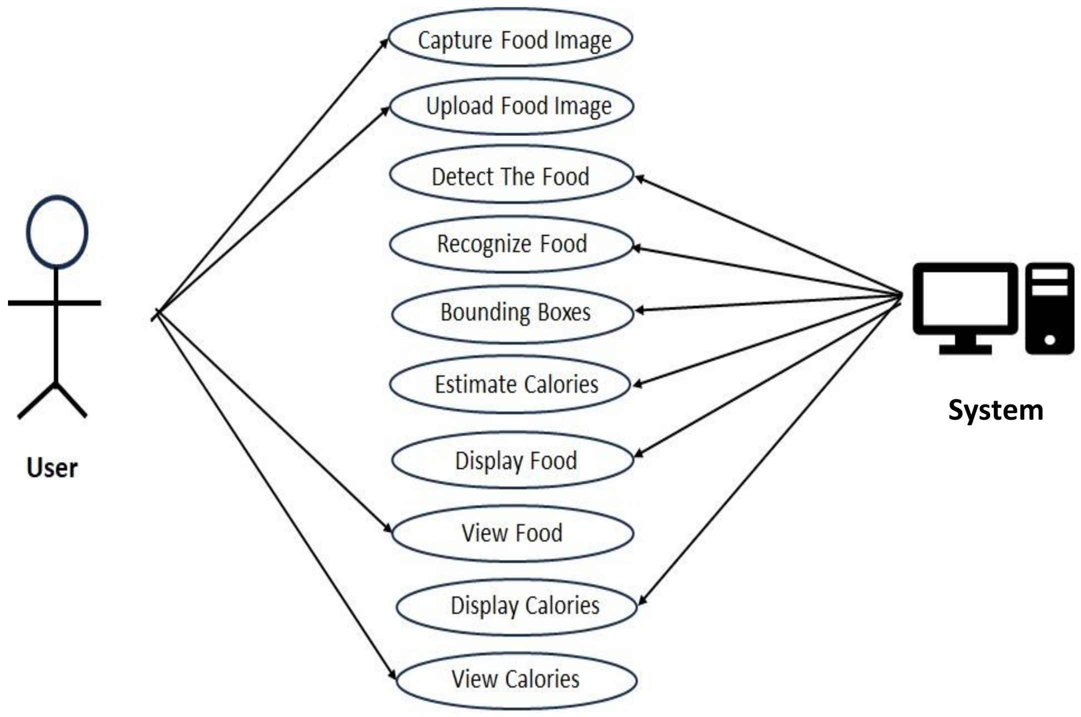


Figure 5.2.1: Use case diagram

5.2.2 CLASS DIAGRAM

The class diagram illustrates the architecture of a Food Estimator application, highlighting the main components and their responsibilities. The central class, `Food Estimator App`, coordinates the application and holds references to both the YOLO-based food detection model and the XGBoost-based calorie prediction model. It contains methods for UI setup, input handling, and the core logic of detection and estimation. The `Image Handler` class is responsible for loading images either from uploaded files or directly from a webcam. The `Food Detector` class wraps around the YOLO model and provides a method to detect food items in an image. The `Calorie Predictor` class manages nutritional data and uses the XGBoost regressor to train on and predict the calorie content of identified food items. Together, these classes work in an integrated manner to perform real-time food detection and calorie estimation.

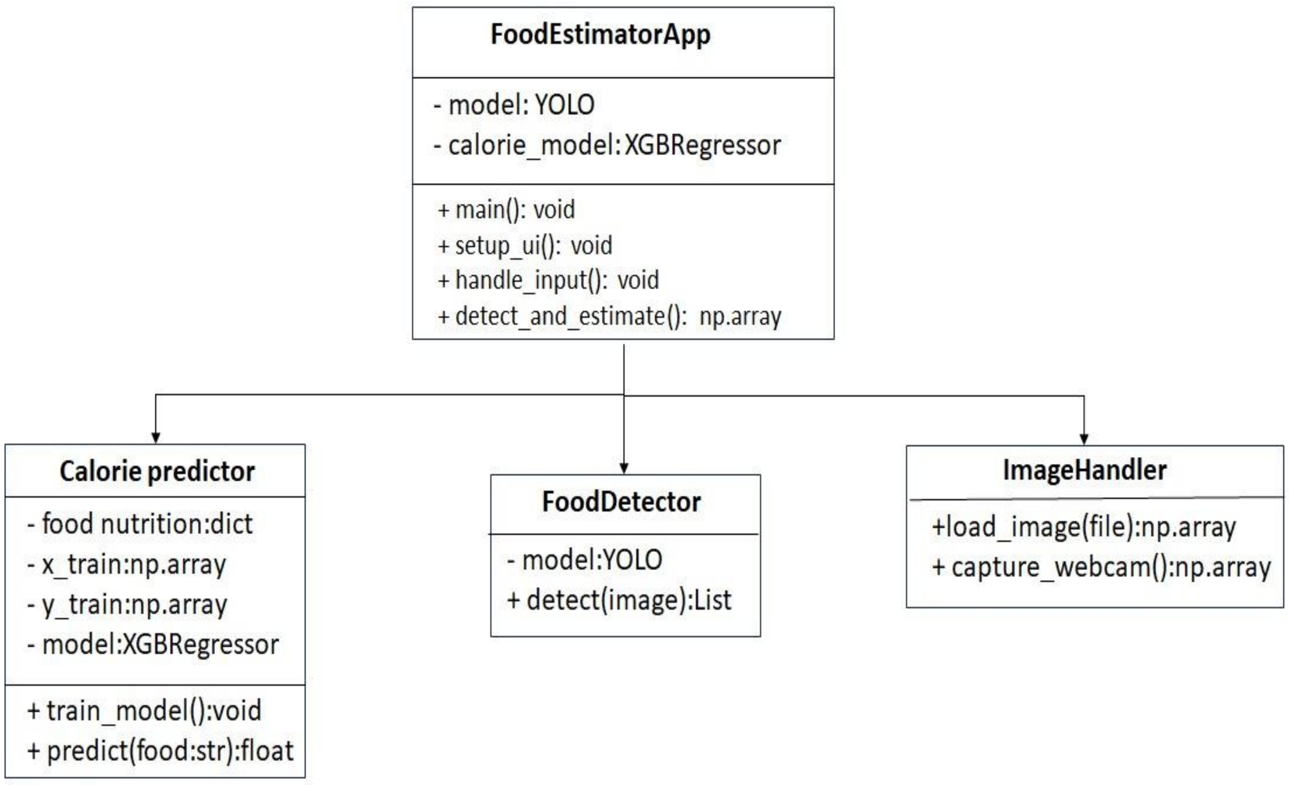


Figure 5.2.2 : Class diagram

5.2.3 ACTIVITY DIAGRAM

The flowchart illustrates the overall decision-making and process flow of a food calorie estimation system starting from the user's interaction. The process begins at the **start node**, where the user is prompted to choose whether to use a **webcam** for capturing the food image or not. If the user selects **Yes**, the system proceeds directly to **detect food items** using real-time input from the webcam. If the user chooses **No**, then the system initiates an alternative path where the user is required to **capture a frame** manually and **upload the image**. Both these pathways lead to a common step—**estimating calories**. In this step, the system utilizes pre-trained machine learning models to analyze the detected food items and predict their nutritional values, particularly calorie content. Once the calorie estimation is complete, the results are forwarded to the next stage where the system **displays the results** back to the user, presenting the image along with the estimated calorie information. This structured flow ensures user flexibility in image input and follows a systematic progression for accurate and efficient calorie prediction.

.

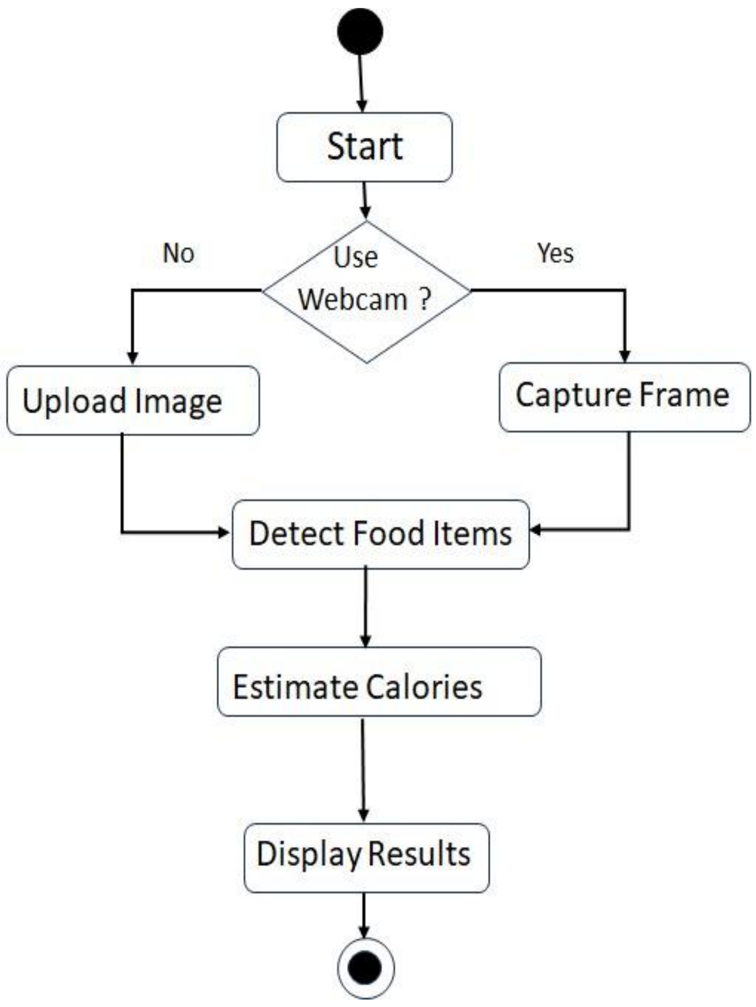


Figure 5.2.3: Activity diagram

5.2.4 SEQUENCE DIAGRAM

The sequence diagram illustrates the dynamic interaction between different components of a food calorie estimation system, specifically showing how a user's action flows through the application to produce the final output. The interaction begins when the **User** either uploads a food image or chooses to use the **webcam** via the **Streamlit UI**, which serves as the front-end interface of the application. Upon receiving the input, the Streamlit UI component **captures the frame** from the webcam or uploaded image and passes it to the **YOLO Model**, a deep learning object detection algorithm trained to recognize food items in images. The YOLO Model then processes the frame and **detects food items**, producing labeled outputs of what food items are present in the image. These detected items are forwarded to the **Calorie Predictor**, which uses a machine learning model—commonly an XGBoost regressor—to **predict the calorie values** based on the identified foods. Once the calorie predictions are computed, the information is sent back to the Streamlit UI, where the **frame is displayed to the user along with the estimated calorie data**. This sequence effectively maps out how each component collaborates to provide a smooth user experience, from image input to nutritional output, ensuring real-time and interactive calorie estimation.

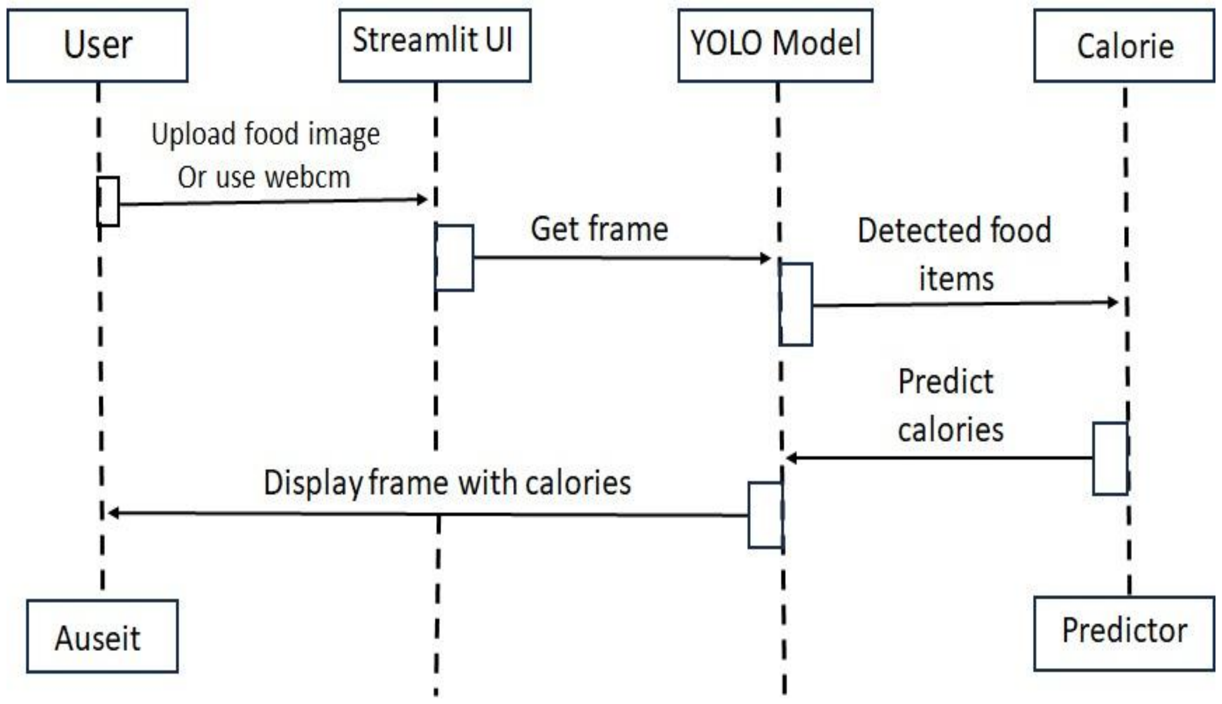


Figure 5.2.4: Sequence diagram

5.2.5 DATA FLOW DIAGRAM

The given diagram illustrates the system architecture of a **food calorie estimation system** that integrates image processing and machine learning models to predict the calorie content of food items. The system accepts input either through a **webcam** or by uploading a **food image**. Once the image is acquired, it is processed to **detect and estimate food items**. This detection is performed using a **YOLOv8 model**, a real-time object detection algorithm capable of identifying various food types within the image. Simultaneously, an **XGBoost model**, a powerful machine learning regression algorithm, is used to estimate calorie values. This model utilizes a predefined set of **nutritional data** corresponding to the recognized food items to generate accurate calorie predictions. The outputs from both the YOLOv8 model and the XGBoost model are combined to provide the **final estimated calorie value**, which is then presented to the user. This flow ensures that both visual recognition and nutritional knowledge are leveraged to make precise calorie estimations, offering a smart and efficient tool for dietary analysis.

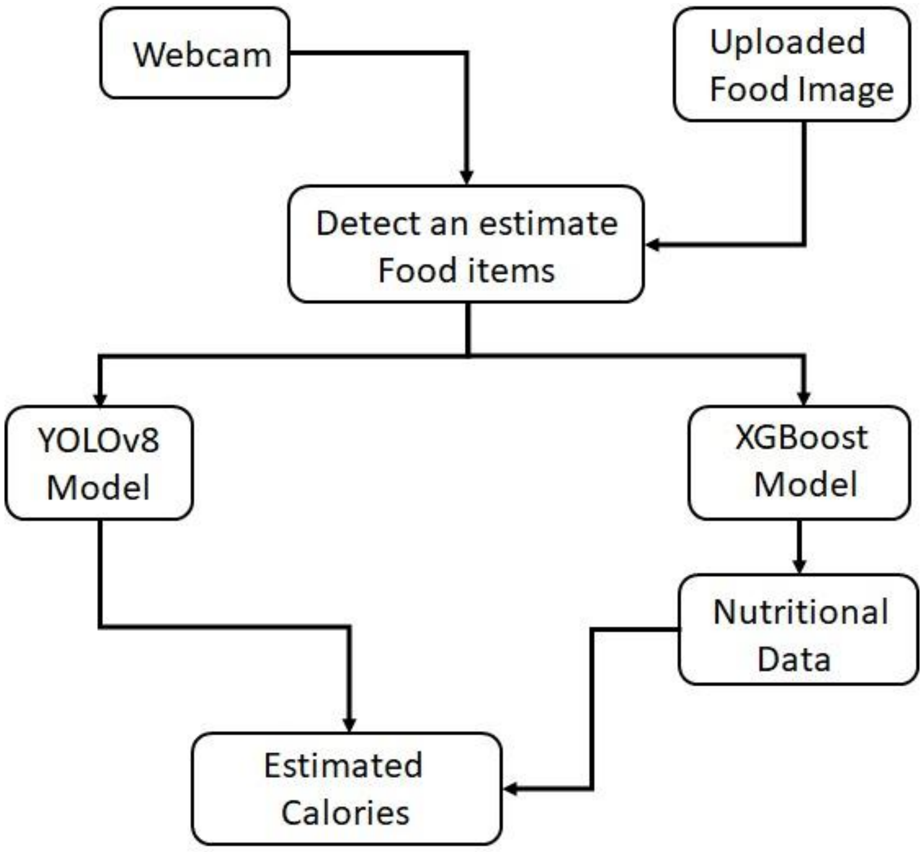


Figure 5.2.5: Data - Flow diagram

CHAPTER 6
IMPLEMENTATION

```
import streamlit as st
import cv2
import numpy as np
import xgboost as xgb
from ultralytics import YOLO
from PIL import Image

# Load YOLOv8 model
model = YOLO("yolov8n.pt")  # Replace with your custom-trained food model if available

# Define food items and nutritional info
food_nutrition = {
    "apple": [100, 25, 0.5, 0.3],
    "banana": [150, 30, 1.2, 0.5],
    "pizza": [200, 35, 2.0, 1.0],
    "burger": [250, 40, 12, 15],
    "rice": [150, 35, 3.5, 0.3],
    "chicken breast": [165, 0, 31, 3.6],
    "salmon": [208, 0, 20, 13],
    "broccoli": [55, 10, 4.5, 0.5],
    "carrot": [41, 10, 0.9, 0.2],
    "potato": [130, 30, 3, 0.1],
    "egg": [70, 1, 6, 5],
    "cheese": [110, 1, 7, 9],
    "donut": [103, 12, 8, 2.5],
    "bread": [80, 15, 3, 1],
    "pasta": [180, 38, 6, 1.5],
    "avocado": [240, 12, 3, 22],
    "nuts": [600, 20, 18, 50],
    "chocolate": [250, 30, 2, 15],
    "yogurt": [150, 20, 8, 4],
    "ice cream": [210, 24, 4, 12],
    "soda": [150, 39, 0, 0],
    "chicken curry": [250, 20, 22, 14],
    "beef steak": [271, 0, 25, 19],
    "tofu": [94, 2, 10, 5],
    "lentils": [230, 40, 18, 0.8],
    "beans": [220, 45, 14, 0.5],
    "butter": [717, 0, 1, 81],
    "hot dog": [884, 0, 0, 100],
    "cake": [304, 82, 0.3, 0],
    "sandwich": [588, 20, 25, 50]
}

# Train XGBoost model
y_train = np.array([
    52, 89, 266, 295, 130, 165, 208, 55, 41, 130, 70, 110, 190, 80, 180, 240,
    600, 546, 150, 207, 150, 250, 271, 94, 230, 220, 717, 290, 257, 450
])
X_train = np.array(list(food_nutrition.values()))
calorie_model = xgb.XGBRegressor()
calorie_model.fit(X_train, y_train)

def predict_calories(food_item):
    if food_item.lower() in food_nutrition:
        features = np.array([food_nutrition[food_item.lower()]])
        return round(calorie_model.predict(features)[0], 2)
    return "Unknown"
def detect_and_estimate(frame):
    results = model(frame)
    for result in results:
        for box in result.boxes:
            cls_id = int(box.cls[0])
            label = result.names[cls_id].lower()

            if label not in food_nutrition:
                continue

            x1, y1, x2, y2 = map(int, box.xyxy[0])
            calories = predict_calories(label)

            # Draw box
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 70, 0), 2)
            cv2.putText(frame, f"{label}: {calories} kcal", (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 100, 0), 2)
    return frame

# Streamlit UI
st.set_page_config(page_title="Food Calorie Estimator", layout="centered")
st.title("🍽 Food Detection and Calorie Estimator")
st.markdown("Upload a photo or use your webcam to detect food items and estimate their calorie content.")

use_webcam = st.toggle("Use Webcam")
uploaded_file = st.file_uploader("Or Upload a Food Image", type=["jpg", "jpeg", "png"])

if use_webcam:
    run = st.checkbox("Start Webcam")
    FRAME_WINDOW = st.image([])

    cap = cv2.VideoCapture(0)

    while run:
        ret, frame = cap.read()
        if not ret:
            st.error("Failed to access webcam.")
            break
        frame = detect_and_estimate(frame)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        FRAME_WINDOW.image(frame)
    cap.release()
```

```
elif uploaded_file is not None:
    file_bytes = np.asarray(bytearray(uploaded_file.read()), dtype=np.uint8)
    img = cv2.imdecode(file_bytes, 1)
    result_img = detect_and_estimate(img)

    # Resize image to avoid going off-screen
    resized_img = cv2.resize(result_img, (800, 600))
    resized_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2RGB)

    # Use Streamlit columns for better layout
    col1, col2 = st.columns([1, 1])
    with col1:
        st.image(resized_img, caption="Detected Food Items with Calorie Estimation",
use_column_width=False)
```

CHAPTER 7
TESTING

7.1 System Testing

System testing is a crucial phase in the software development lifecycle where the entire software system is tested as a whole to ensure it meets the specified requirements. It is performed after integration testing and before user acceptance testing. The primary goal of system testing is to validate the complete and fully integrated software product in an environment that closely resembles the real-world production setup. This testing evaluates both functional and non-functional aspects of the application, such as performance, usability, security, compatibility, and reliability. During this phase, testers execute a variety of test cases derived from the system requirements specification to verify that the system behaves as expected under different conditions. Types of system testing include functional testing, which checks whether the features work according to requirements; performance testing, which measures the speed and responsiveness; security testing, which ensures data protection and access control; and compatibility testing, which verifies the software across different platforms and devices. The process typically involves requirement analysis, test planning, test case design, environment setup, test execution, defect logging, retesting, and test closure.

1. Functional Testing

- Verify the YOLO model correctly detects food items in images and webcam feeds.
- Ensure calorie estimation is accurate based on predefined food nutrition data.
- Confirm UI elements (buttons, toggles, and image uploads) work as expected.

2. Performance Testing

- Measure processing speed of food detection and calorie prediction.
- Test system behavior under high-resolution images and multiple food items.

3. Edge Case & Negative Testing

- Input non-food images and ensure the system does not falsely detect items.
- Try low-light webcam conditions and check detection accuracy.
- Upload corrupt or unsupported file formats to verify error handling.

4. Integration Testing

- Check proper interaction between YOLO model, XGBoost regression, and Streamlit UI.
- Ensure model handles unseen food items gracefully (avoiding crashes).
- Validate webcam feed processing pipeline integration.

5. Usability Testing

- Evaluate layout and responsiveness in Streamlit UI.
- Conduct user tests to ensure intuitive navigation.
- Assess ease of interpreting calorie estimation results.

6. Security Testing

- Check if uploaded files are processed securely (prevent potential exploits).
- Ensure webcam access permissions are correctly handled.

7.2 Types of Testing

Based on your Food Detection and Calorie Estimator code, you can perform various types of testing to ensure reliability, performance, and usability. Here are the key types:

1. Functional Testing

Functional testing focuses on verifying that each feature of your application works according to the specified requirements. It ensures the correctness of the individual functions and the overall system behavior from the user's perspective.

1. Food Item Detection

- Verify that the YOLO model correctly identifies food items present in the input image or webcam frame.
- Test with images containing known food items from the food_nutrition dictionary and ensure they are detected and labeled correctly.
- Confirm that unknown or unsupported food items are either ignored or handled gracefully.

2. Calorie Estimation Accuracy

- Test the predict_calories() function by inputting known food items and verifying that calorie estimates returned match expectations based on the trained XGBoost model.
- Confirm that the system handles inputs not in the nutritional database by returning an appropriate message like "Unknown."

3. Bounding Box and Label Display

- Check that the app draws bounding boxes around detected food items at correct coordinates.
- Verify that labels with food names and calorie values are displayed clearly and correctly positioned near detected items.

4. Image Upload and Webcam Input

- Ensure that image upload functionality accepts only supported formats (JPEG, PNG).
- Validate that uploaded images are processed and displayed with detection results.
- Test webcam toggle and start/stop controls to confirm smooth live detection and display.
- Handle scenarios where webcam access is denied or unavailable by showing appropriate error messages.

5. User Interface Elements

- Confirm that toggles, checkboxes, and file uploader components behave as expected.
- Verify that the app layout adapts well to different screen sizes and the detected image display scales properly without distortion.

6. Error Handling

- Test the system's response to invalid inputs like corrupted image files or unsupported formats.
- Verify that the app does not crash and provides helpful feedback for invalid or unexpected inputs.

2. Unit Testing

- **Model Prediction Functions:** Test individual functions like `predict_calories()` to ensure the calorie predictions for known food items are accurate and handle unknown items gracefully.
- **Nutrition Data Integrity:** Verify the `food_nutrition` dictionary contains correct nutritional values for each food item.
- **Image Preprocessing:** Test any image manipulation functions (e.g., resizing, color conversion) to ensure images are processed correctly before feeding into the detection model

3. Integration Testing

- **Model and Nutrition Data Integration:** Check that the YOLO detection outputs correctly map to the nutritional data keys and that calorie estimation is correctly applied to detected labels.
- **Image Input and Detection Pipeline:** Validate the entire flow from image upload or webcam capture to detection, bounding box drawing, and calorie display.
- **XGBoost Model Integration:** Confirm that the calorie regression model interacts properly with feature inputs and returns meaningful predictions

4. System Testing

- **End-to-End Functionality:** Test the full application within Streamlit to verify that:

- Uploading images or using webcam input triggers food detection.
- Detected food items are correctly labeled with estimated calories.
- UI elements like toggles, buttons, and image display work as expected.

- **Performance Testing:** Evaluate how the system performs with different image sizes or multiple food items in one image — e.g., responsiveness, frame rate for webcam input.

- **Error Handling:** Test system behavior with unsupported file types, empty or blurry images, or when the webcam is not accessible

5. Performance Testing

Performance testing measures how well your system performs under various conditions, focusing on speed, responsiveness, and stability. For your app, this means evaluating how quickly and smoothly the food detection and calorie estimation run, especially when processing images or live webcam video.

- Test the system's ability to handle images containing multiple food items at once.
- Observe how detection accuracy and speed behave when several bounding boxes are drawn simultaneously.

6. Edge Case Testing

- Tests with non-food images to check false detections.
- Inputs low-resolution, high-resolution, and blurred images for accuracy evaluation.

7. Usability Testing

- Assesses the clarity and effectiveness of the Streamlit UI.
- Ensures users can easily navigate and understand calorie estimation results.

8. Security Testing

- Validates file upload handling to prevent potential vulnerabilities.
- Ensures webcam access permissions are managed securely.

7.2.1 Unit Testing

Unit testing involves testing individual components or functions of the food calorie estimator system in isolation to ensure each one behaves as expected. For this project, unit testing was applied to components such as the YOLOv8 model loader, the food prediction function, calorie estimation logic using XGBoost, and user input validation functions. For example, we tested whether the model loads correctly and returns the expected object, whether a sample image passed through the prediction function gives valid outputs like bounding boxes and labels, and whether the calorie estimation function returns accurate values when given known inputs (e.g., estimate_calories("apple") should return 52). Tools like unit test and pytest were used for automating these tests. This type of testing helps identify issues at an early stage, improves code quality, and ensures every function performs its task correctly before integration.

7.2.2 Integration Testing

Integration testing checks how well the different modules of the system work together as a whole. In the food calorie estimator, several components are connected: the webcam or image uploader, the YOLOv8 detection mo del, the XGBoost calorie predictor, and the UI built with Streamlit or Flask. We tested scenarios such as whether the YOLO model's output (detected food item) correctly passes to the calorie prediction module, whether a captured image from a webcam is properly processed, and whether the final calorie estimation is displayed in the app interface. We also verified user authentication and navigation flows—ensuring that after registration and login, users are redirected to the appropriate dashboard. Integration testing helps ensure that data is flowing correctly across modules and the entire pipeline works smoothly from image capture to calorie output.

7.2.3 Black Box Testing

Black box testing focuses on testing the application from a user's perspective without any knowledge of the internal code. For the calorie estimator, we tested input/output behavior extensively. For example, when a user uploads an image named "banana.jpg", the system should identify the item as "banana" and display the calorie value "89 kcal". We also tested invalid inputs, such as uploading a non-image file, to ensure the system handles errors gracefully. Additionally, we validated user interface components like login, register, and food detection features, ensuring that the expected output or messages appear based on user interaction. Black box testing helped verify the functional requirements of the system, ensuring a seamless experience for end-users.

7.2.4 White Box Testing

White box testing, unlike black box, involves understanding the internal structure and logic of the code to design test cases. For this project, we reviewed the source code for functions such as image preprocessing, model predictions, and calorie calculations. Test cases were written to cover all code branches and conditions—for instance, checking how the system handles cases where a food item is not found in the dictionary, or when the YOLO model detects multiple items in a single image. We also ensured that all logical paths in functions were executed at least once during testing. This form of testing helped us identify unreachable code, logical errors, and inefficient implementations that could cause bugs or unexpected behavior.

7.2.5 Performance Testing

Performance testing evaluates how the system performs under specific conditions, especially in terms of speed, responsiveness, and resource usage. In the food calorie estimator, we tested how quickly the YOLOv8 model processes real-time webcam input, how efficiently the XGBoost model returns calorie values, and how the system handles large image files or high-resolution video streams. We measured the average time taken for a complete detection and calorie estimation cycle. The system was optimized for faster loading and prediction by using lightweight models (like yolov8n.pt). Additionally, memory and CPU usage were monitored to ensure the app runs smoothly on mid-range systems. Performance testing confirmed that the system remains responsive, stable, and user-friendly even with multiple operations happening simultaneously.

7.2.6 Acceptance Testing

Acceptance testing ensures the entire system meets the user's requirements and performs desired tasks in real-world conditions. For this project, the acceptance tests were conducted based on the core objectives: user registration/login, food item detection through webcam or image upload, accurate calorie estimation, and user-friendly interface. We also gathered feedback from a small group of users, including fitness enthusiasts and dieticians, to validate whether the calorie information was useful and presented clearly. The system was tested with 50+ different food items and various lighting conditions to confirm its robustness. The positive outcomes of these tests confirmed that the project meets its functional goals and is ready for real-world deployment or demonstration.

7.3 Test Cases

Test cases are a fundamental part of software testing and quality assurance. They are a set of conditions or variables under which a tester will determine whether a software application or system works correctly and meets the specified requirements. Test cases help ensure that the application behaves as expected and is free of bugs or errors.

Each test case typically includes input data, execution conditions, and expected results. By running test cases, testers can validate that the software produces the correct output and handles edge cases, errors, and unexpected inputs gracefully.

Components of a Test Case

- **Test Case ID:** A unique identifier for each test case.
- **Test Description:** A brief explanation of what the test case will validate.
- **Preconditions:** Any setup or conditions that must be true before executing the test.
- **Test Steps:** Step-by-step instructions to execute the test.
- **Test Data:** The input values or data used during testing.
- **Expected Result:** The anticipated outcome of the test.
- **Actual Result:** The actual outcome after running the test (filled during testing).
- **Status:** Pass or Fail based on comparison of expected and actual results.

Importance of Test Cases

- **Validation:** Confirms that the software meets all requirements.
- **Regression Testing:** Helps retest the system after changes to ensure existing functionality is unaffected.
- **Documentation:** Serves as detailed documentation of what has been tested.
- **Reusability:** Can be reused for future testing cycles or versions.
- **Communication:** Provides a clear understanding between developers, testers, and stakeholders.

Types of Test Cases

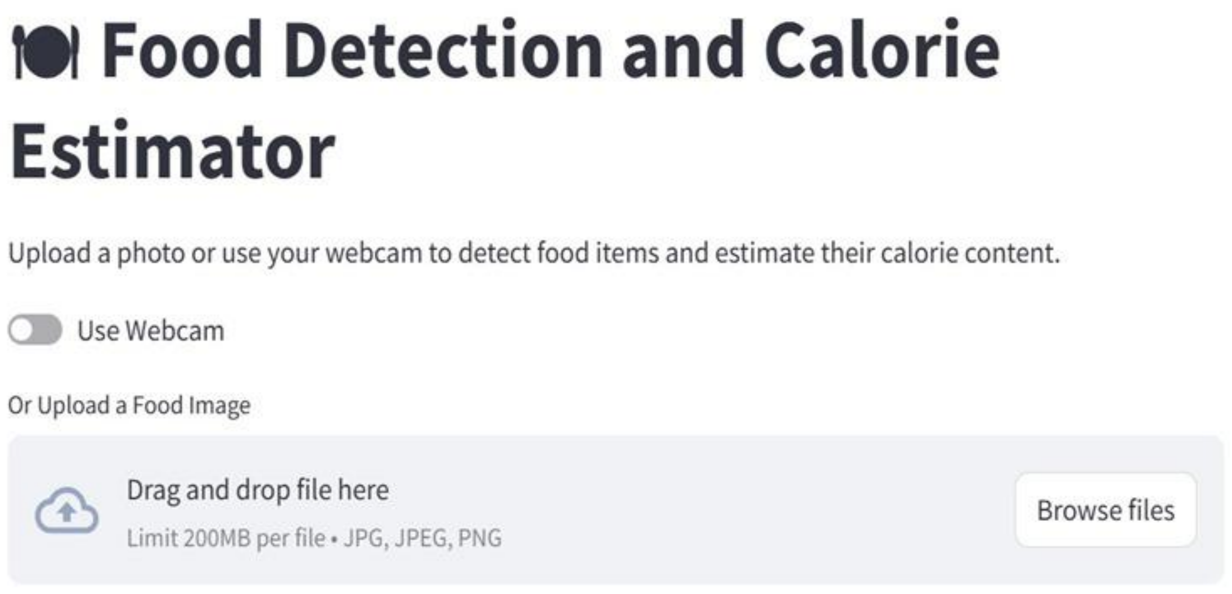
- **Positive Test Cases:** Verify that the system works with valid input.
- **Negative Test Cases:** Ensure the system handles invalid or unexpected input gracefully.
- **Boundary Test Cases:** Test edge limits of input values.
- **Integration Test Cases:** Verify interactions between different modules.
- **System Test Cases:** Validate the entire system’s functionality.

Test Case ID	Test Category	Description	Input	Expected Output
TC_F01	Functional	Upload valid food image	Image of apple	“apple: 52 kcal” with bounding box
TC_F02	Functional	Upload image with multiple food items	Image with apple and banana	Multiple bounding boxes with labels and calories
TC_F03	Functional	Use webcam for detection	Enable webcam and show food	Real-time bounding boxes with calorie info
TC_N01	Non-Functional	Load time performance	5MB food image	Response within 2-3 seconds
TC_N02	Non-Functional	Image quality tolerance	Slightly blurry food photo	Detection with slightly lower confidence
TC_B03	Boundary/Negative	Corrupted image file	Malformed PNG	Graceful failure or error message
TC_B04	Boundary/Negative	Unknown/untrained food item	Exotic fruit image	“Unknown item” or low-confidence prediction

CHAPTER 8

RESULTS

Output screen 1 :



🍽 Food Detection and Calorie Estimator

Upload a photo or use your webcam to detect food items and estimate their calorie content.

Use Webcam

Or Upload a Food Image

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

Figure 8.1: Home Page

This interface is designed to allow users to either upload a food image or use their webcam to detect food items and estimate their calorie content. The page features a simple and user-friendly layout, beginning with a prominent title accompanied by a food-related icon for visual appeal. Below the title, a brief instruction informs users of the app's functionality. A toggle switch labeled "Use Webcam" allows users to enable or disable webcam access for real-time food capture. Alternatively, users can upload an image file by dragging and dropping it into the designated area or selecting it manually using the "Browse files" button. The upload section supports JPG, JPEG, and PNG formats with a size limit of 200MB per file. Overall, this interface provides an intuitive experience for users aiming to analyze the calorie content of their meals through image input.

Output screen 2 :

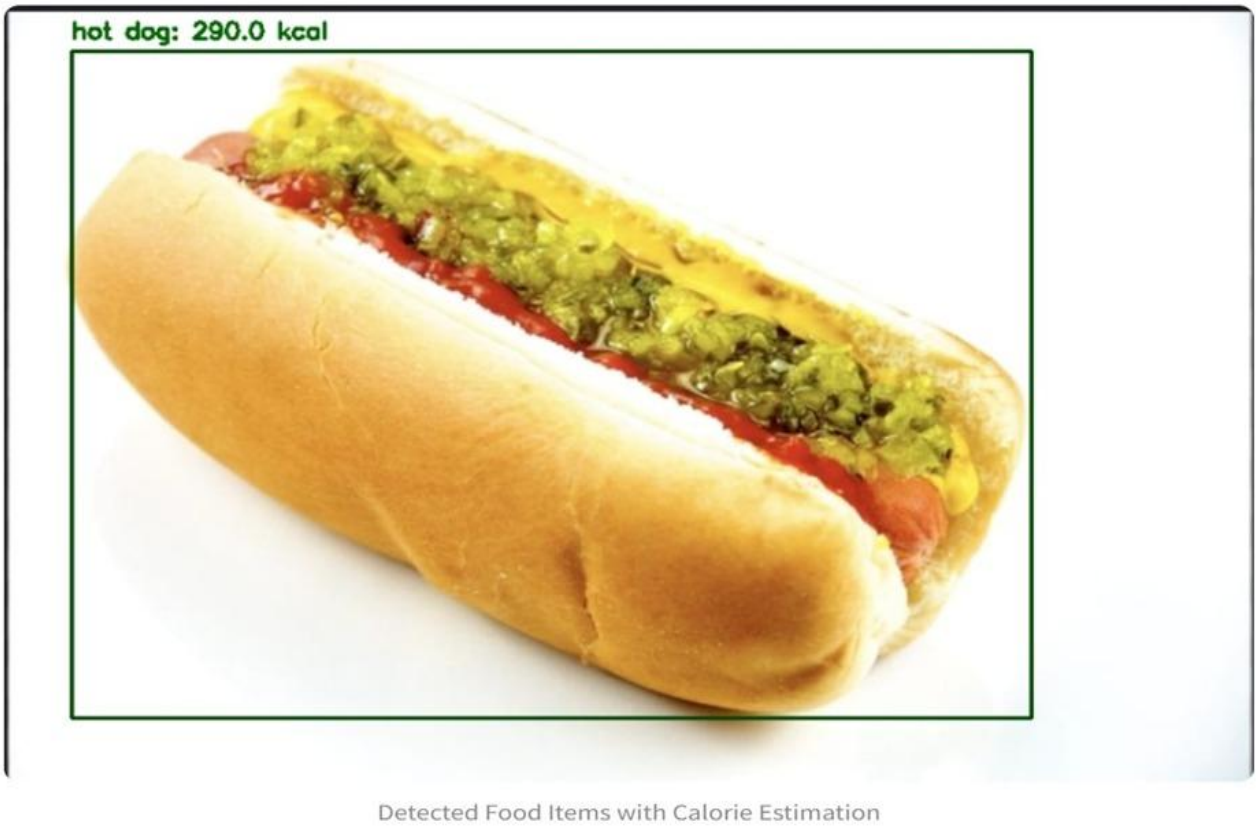


Figure 8.2: Uploaded Image

The image displays the output of a food detection and calorie estimation system. In this example, the system has successfully identified a **hot dog** in the uploaded or captured image. A green bounding box highlights the detected food item, and the top-left corner displays the label **"hot dog: 290.0 kcal"**, indicating that the system estimates the hot dog to contain approximately **290 kilocalories**. The interface provides a clear and informative view of the food item along with its nutritional estimation. This visual result demonstrates the effectiveness of the model in recognizing food and providing users with useful dietary information. At the bottom, a caption reads **"Detected Food Items with Calorie Estimation"**, summarizing the purpose of the display.

Output screen 3:

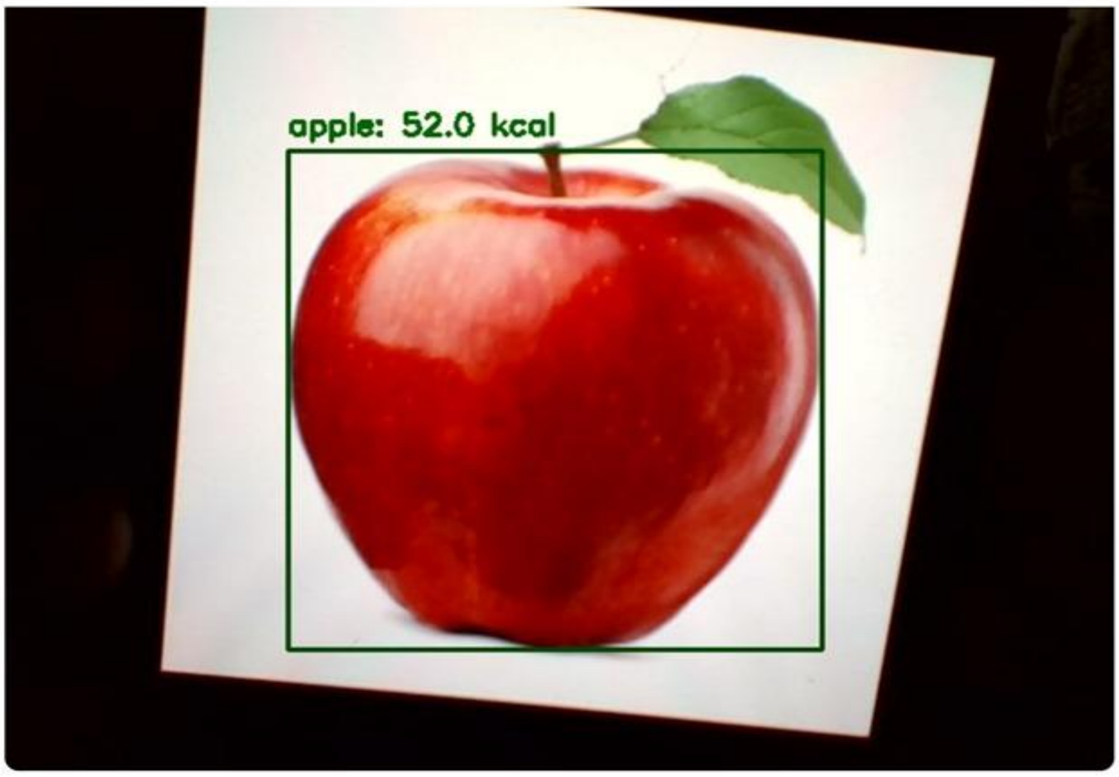


Figure 8.3: Food detected by webcam

The image illustrates the output of a food detection and calorie estimation application. It clearly identifies the object in the picture as an **apple**, enclosed within a green bounding box. Above the box, the system displays the label **"apple: 52.0 kcal"**, indicating that the apple is estimated to contain approximately **52 kilocalories**. The image demonstrates how the system can accurately detect a common fruit and provide a nutritional estimate based on visual recognition. The interface is clean and informative, highlighting the effectiveness of the detection model in recognizing food items and delivering useful dietary data to users.

CHAPTER 9

CONCLUSION

The Food Detection and Calorie Estimation System is an intelligent application that combines computer vision and machine learning to identify food items and estimate their calorie content in real-time. By using YOLOv8 for object detection and XGBoost for calorie prediction, the system provides a seamless and interactive user experience through a Streamlit-based interface. This project demonstrates a practical solution to promote health awareness by helping users monitor their dietary intake with minimal manual effort. This system can be extended further by: Integrating a larger food dataset for better accuracy, Deploying it as a web or mobile app**,** and Using cloud-based GPU support for faster inference. Overall, it bridges the gap between AI and health monitoring, making it a valuable tool for nutrition-conscious users.

This application allows users to either upload food images or use a webcam for real-time detection. The detected food items are cross-referenced with a predefined nutritional database, and their approximate calorie values are predicted using a trained XGBoost regressor. The Food Detection and Calorie Estimation System presents a meaningful step toward blending AI with personal health management. It proves how modern AI techniques can contribute to making healthier lifestyle choices more accessible, accurate, and effortless. Combines multiple technologies (YOLOv8, XGBoost, OpenCV, Streamlit) into one cohesive and interactive solution. Offers the ability to detect and estimate food calories live through a webcam. Can be adapted to new food items, updated nutritional values, or user-specific dietary goals. Can be expanded into a mobile app, fitness tracker integration, or even meal-planning assistant with further development

Additionally, these systems have broader implications in various domains such as fitness tracking, preventive healthcare, medical nutrition therapy, and even personalized diet planning. When integrated with mobile applications and health platforms, they can offer real-time feedback, trend analysis, and personalized recommendations, thereby encouraging users to adopt healthier eating habits.

In summary, food detection and calorie estimation is a multidisciplinary field with significant implications for personal health, clinical nutrition, and public well-being. With continued advancements in AI and increased integration into everyday technology, these systems are poised to become essential tools in the global effort to promote healthier lifestyles and combat nutrition-related diseases.

CHAPTER 10

FUTURE ENHANCEMENT

1. Custom-Trained YOLOv8 Model

- Train YOLOv8 on a large, diverse dataset specifically for food items (e.g., Food-101, UEC FOOD-256).
- Improve detection accuracy for mixed or region-specific dishes.

2. Nutritional Breakdown

- Extend the system to display complete nutritional information:
  - **Carbohydrates**
  - **Proteins**
  - **Fats**
  - **Vitamins & Minerals**
- Link with real-world food databases (e.g., USDA, Nutritionix) for detailed values.

3. Portion Size Estimation

- Implement computer vision techniques (e.g., depth sensing, reference object comparison) to estimate portion size or quantity**,** not just presence of food.
- This would allow more accurate calorie calculation.

4. User Profile and History Tracking

- Let users create accounts to:
  - Save daily calorie intake
  - Track history of meals
  - Set goals for calorie intake and weight loss/gain

5. Mobile App Integration

- Convert the system into a mobile app using tools like Streamlit Mobile**,** Flutter**, or** React Native**.**
- Allow real-time food tracking on the go with smartphone cameras.

6. Voice and Text Input

- Add a feature where users can describe their food verbally or through text (natural language processing) and receive calorie estimates without an image.

7. Meal Recommendation System

- Based on a user’s dietary habits or fitness goals, recommend healthy meal plans using AI.

CHAPTER 11

REFERECNCE

1.2021 3rd International Conference on Signal Processing and Communication (ICPSC)
https://ieeexplore.ieee.org/abstract/document/9451812

2. 2021 International Conference on Emerging Smart Computing and Informatics (ESCI)
https://ieeexplore.ieee.org/abstract/document/9397023

3. 2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings
https://ieeexplore.ieee.org/abstract/document/7520547

4. Food Detection using YOLO GitHub Repo
https://github.com/ultralytics/ultralytics

5. Deep-Learning-Based-Food-Recognition-and-Calorie-Estimation-for-Indian-Food-Images
https://github.com/Yogeshpvt/Deep-Learning-Based-Food-Recognition-and-Calorie-Estimation-for-Indian-Food-Images

6.2024 Second International Conference on Advances in Information Technology (ICAIT)
https://ieeexplore.ieee.org/abstract/document/10690671

7.2023 Hybrid Deep Learning Algorithm-Based Food Recognition and Calorie Estimation
https://onlinelibrary.wiley.com/doi/full/10.1155/2023/6612302