1. **Define Objectives and Requirements**

* **Goal**: Develop a system to detect food from images, estimate portion sizes, calculate calorie content, and provide dietary suggestions.
* **Key Features**:

1. Upload images via a user-friendly frontend.
2. Detect and classify food items.
3. Estimate portion sizes and compute calories.
4. Provide suggestions for portion control or meal balancing.
5. **Data Collection and Preparation**
   1. **Source Images**
      * Use platforms like Flickr, Instagram, or food blogs for diverse, high-quality images.
      * APIs: Unsplash API, Google Images Scraper for automating image collection.
   2. **Ensure Diversity**
      * Include Indian dishes, covering regional specialties, snacks, curries, breads, and sweets.
      * Account for different portion sizes and variations (e.g., butter naan vs. garlic naan).
   3. **Annotate Data**
      * Use tools like LabelImg or RectLabel.
      * Label the dataset for:
        + **Classification:** Each image is labeled with the food item's name (e.g., "biryani").
        + **Object Detection:** Annotate bounding boxes if food items need localization in an image
   4. **Organize Dataset**
      * **Split data into:**
        + Training (70%)
        + Validation (20%)
        + Testing (10%)
   5. **Augment Dataset**
      * Use libraries like imgaug or Albumentations for:
        + Rotations, flips, cropping.
        + Adjust brightness, contrast, and color variations.
   6. **Create a Food-Calorie Database**
      * Design a CSV or database table

|  |  |
| --- | --- |
| **food\_item** | **calories\_per\_100g** |
| biryani | 300 |
| idli | 60 |
| paneer\_butter\_masala | 400 |

* + - * + Use trusted sources like the **National Institute of Nutrition, India**, or APIs like **USDA FoodData Central** to populate data.
        + <https://www.kaggle.com/datasets/syedkhalid076/indian-food-nutrition>
        + <https://www.nutritionix.com/food/indian-foods>

1. **Model Training and Development**
   1. **Select Pre-trained Models**
      * + **MobileNet:**

- Lightweight, efficient for deployment.

- Good for mobile/web applications.

* + - * **ResNet:**
* Highly accurate but computationally heavier.
* Use for backends with sufficient resources.
  + - * **YOLO (You Only Look Once):**
* For object detection (multi-food recognition or portion estimation).
  1. **Fine-Tune Models**
     + Use **Food-101** as a base dataset.
     + Supplement with custom Indian food data for domain specificity.
     + Frameworks: **TensorFlow/Keras** or **PyTorch**.
       - Export model in formats like **SavedModel (.pb)** or **TorchScript (.pt)**.

1. **Portion Estimation Model**
   1. **Depth and Shape-Based Estimation**
      * Use MonoDepth or similar models for depth estimation.
      * For standard foods
        + Assume geometric shapes (e.g., spheres for apples, cylinders for soda cans).
   2. **Reference Object Scaling**
      * Detect reference objects (e.g., plates, coins, spoons) using YOLO or Faster R-CNN.
      * Estimate food dimensions relative to these objects.
   3. **Object Detection**
      * Train YOLO/Detectron on annotated data for multi-food detection.
2. **Backend Development**
   1. **Choose Framework**
      * Use FastAPI for high-performance APIs.
        + Advantages:
          - Type hinting for easier development.
          - Automatic Swagger documentation.
          - Async processing for scalability.
   2. **Environment Setup**
      * Python Environment
      * Database
        + Use SQLite for small-scale applications.
        + Switch to PostgreSQL for production scalability.
   3. **Build API Endpoints**
      * **Upload Image:**
        + Endpoint: /upload
        + Accept user-uploaded images.
      * **Process Image:**
        + Endpoint: /process
        + Integrate food detection model.
        + Estimate portion size.
      * **Fetch Calorie Info:**
        + Endpoint: /calories
        + Query database for calorie values.
      * **Suggestions:**
        + Endpoint: /suggestions
        + Return portion control or dietary suggestions.
3. **Frontend Development**
   1. **Choose Framework**
      * Use React.js for an interactive frontend.
   2. **Implement Features**
      * **Image Upload**
        + Integrate with the backend via HTTP requests.
        + Libraries: axios or fetch API.
      * **Results Display**
        + Show detected food items, portion size, and calorie count.
      * **Suggestions Section**
        + Provide tips for healthier eating or portion control.
4. **Deployment**
   1. **Cloud Services**
      * AWS EC2, Google Cloud, or Heroku for hosting.
      * Use Nginx as a reverse proxy for scalability.
5. **Testing and Optimization**
   1. **Image Inputs**
      * Test under various conditions (lighting, angles, food arrangements).
      * Check performance for Indian dishes and portion estimation accuracy.
   2. **Scalability**
      * Use tools like Apache JMeter to stress-test API endpoints.
6. **Continuous Improvement**
   1. **Enhancements**
      * Add multi-language support to make the app accessible.
      * Train on additional datasets for global cuisine.
      * Enable user feedback loops for correcting misclassifications.
7. **Final Deliverables**
   * + - **Backend API:** FastAPI-based, integrated with ML models and database.
       - **Frontend Application:** React.js-based interface for user interaction.
       - **Deployed System:** Cloud-hosted API and frontend for end-to-end calorie estimation.

**python -m venv yolov8-env**

**yolov8-env\Scripts\activate**

**train the model : python train\_yolov8.py --mode train**

**Evaluate : python train\_yolov8.py --mode evaluate**

**Predict: python train\_yolov8.py --mode predict**

[**https://skalefitness.com/indian-food-calorie-chart/**](https://skalefitness.com/indian-food-calorie-chart/)

[**https://www.nutritionix.com/food/rice**](https://www.nutritionix.com/food/rice)