FOOD NUTRITION DETECTOR PROJECT

- Developed by: Saijasi Sinha
- Registration number : 25BSA10021

Introduction

- The Food Nutrition Detector is a machine-learning-based system that identifies a food item from an image and predicts its nutritional values, such as calories, proteins, fats, and carbohydrates.

- The project applies deep learning, image processing, and backend integration to build an end-to-end functional solution.

Problem Statement

- Many individuals struggle to track their nutritional intake manually. Existing methods often require manual searches or estimations, leading to inaccuracies.

- This project solves the issue by allowing users to simply upload a food image and instantly receive nutritional estimates.

Functional Requirements

- User should be able to upload an image.
- System detects food category using ML.
- Predict nutritional values from dataset.
- Display results visually in UI.
- Provide calorie, protein, fat, and carbohydrate breakdown.

Non-functional Requirements

- Accuracy: ML model should maintain good prediction accuracy.
- Scalability: System should handle multiple food types.
- Efficiency: Quick processing and prediction.
- Usability: Simple and user-friendly interface.
- Maintainability: Easy to update dataset or model.

System Architecture

- Architecture Overview:
- Frontend (HTML/CSS/JS) → Image Upload → Flask Backend → Preprocessing → ML Model → Nutrient Predictor → Output Display
- Components:
- - Web UI
- - Flask Server
- - CNN-based Food Recognition Model
- - Nutrition Database
- - Output Renderer

Use Case Diagram

- Actors: User, System
- Use Cases:
- - Upload Image
- - Detect Food
- - Retrieve Nutrition
- - Display Output

Workflow Diagram

- User uploads food image.
- Image is sent to backend.
- Preprocessing and model prediction.
- Nutrition lookup from dataset.
- Display results to user.

```
+---------------+
|     User      |
+---------------+
        |
------------------------------
 |         |            |
Upload Image  Capture Image
 View Nutrition
 |         |            |
------------------------------
        |
     Detect Food
        |
     Lookup Nutrition
        |
     Display Results
        |
     Save Daily Nutrition
```

Sequence Diagram

- User → UI: Upload Image
- UI → Backend: Send Image
- Backend → Model: Predict Food
- Model → Backend: Return Label
- Backend → Database: Fetch Nutrition
- Backend → UI: Display Results

```
User        Web UI / CLI      Detect Module       Lookup
Module        CSV Storage
  |               |               |               |               |
  |  Upload Image    |               |               |               |
  | ---------------> |               |               |               |
  |               |  detect_food()   |               |               |
  |               | ---------------> |               |               |
  |               |               |  Identify label  |               |
  |               |               | ---------------> |               |
  |               |               |               |  Query CSV        |
  |               |               |               | ---------------> |
  |               |               |               |  Return Nutrition  |
  |               |               | <--------------- |               |
  |               | <--------------- |               |               |
  |  Display Result  |               |               |               |
```

Class/Component Diagram

- Components:
- - ImageProcessor
- - FoodClassifier (ML Model)
- - NutritionDatabase
- - FlaskController
- - UI Renderer

```
+--------------------+
| detect.py          |
|--------------------|
| +detect_food()     |
+--------------------+

+--------------------+
| lookup.py          |
|--------------------|
| +get_nutrition()   |
+--------------------+

+--------------------+
| food_utils.py      |
|--------------------|
| +FoodUtil          |
| +get_nutrition()   |
+--------------------+
```

```
+--------------------+
| data_loader.py     |
|--------------------|
| +load_dataset()    |
+--------------------+

+--------------------+
| detect_and_lookup.py|
|--------------------|
| +index()           |
| +find_candidates()|
+--------------------+

+--------------------+
| main.py            |
|--------------------|
| +CLI workflow      |
+--------------------+
```

ER Diagram

Entities:

- - Food
- - Nutrition

Relationships:

- Food (id, name) → Nutrition (calories, proteins, fats, carbs)

```
+---------------------------+
| daily_food_nutrition.csv  |
|---------------------------|
| id (int)                  |
| food_name (string)        |
| calories (float)          |
| protein (float)           |
| fat (float)               |
| carbs (float)             |
| date (date)               |
+---------------------------+

+---------------------------+
| nutrition_dataset.csv     |
|---------------------------|
| food_name (string)        |
| calories (float)          |
| protein (float)           |
| fat (float)               |
| carbs (float)             |
+---------------------------+
```

Design Decisions & Rationale

- Flask chosen for lightweight backend.
- CNN model selected for high accuracy in image recognition.
- Kaggle dataset used for nutrition mapping.
- Frontend kept simple for clarity.
- Modular design for easier updates.

Implementation Details

Technologies Used:

- - Python, Flask
- - OpenCV for image processing
- - TensorFlow/Keras for ML model
- - HTML, CSS, JS for frontend

Steps:

- 1. Data preprocessing
- 2. Model training
- 3. Backend integration
- 4. User interface design
- 5. Nutrition mapping

Screenshots / Results

- Sample Results:
- - Image: Food.jpg
- - Detected: Apple
- - Calories: 52 kcal
- - Protein: 0.3g
- - Fats: 0.2g
- - Carbs: 14g

Testing Approach

- 1. Unit testing for backend routes.
- 2. Model accuracy evaluation.
- 3. Dataset validation tests.
- 4. UI interaction tests.
- 5. Integration testing for full pipeline.

Challenges Faced

- - Low quality images reduced accuracy.
- - Confusion between visually similar foods.
- - Dataset inconsistencies.
- - Need for efficient preprocessing.

Future Enhancements

- Multi-food detection in a single image.
- Add diet tracking history.
- Mobile application version.
- Voice-based food logging.
- Larger dataset for better accuracy.

Learnings & Key Takeaways

- - Understanding end-to-end ML application development.
- - Hands-on experience with Flask backend.
- - Improved knowledge of dataset handling.
- - Real-world challenges in computer vision.

References

- Kaggle Food Nutrition Dataset
- TensorFlow Documentation
- Flask Official Documentation
- OpenCV Library
- Research Papers on Food Detection Models