Shaheed Zulfikar Ali Bhutto Institute of Science & Technology University

DEPARTMENT OF ROBOTICS & ARTIFICIAL INTELLIGENCE

Total Marks:_____

Obtained Marks:_____

Research Report

Final Project Report

Project Title:

Food Vision Using EfficientNet & ResNet

Submitted To: Sir Khawaja Tahir

Student Name: Ali Haider, Maria Ali

Enrollment No: 22108108, 22108160

DEPARTMENT OF ROBOTICS & ARTIFICIAL INTELLIGENCE

➢ Overview:

The project demonstrates the integration of a deep learning-based food classification system with a Flask web application. This implementation provides a comprehensive solution for recognizing food items from images and displaying corresponding nutritional information, particularly calorie data. The codebase consists of two major components:

1. **Backend (Model Training and Prediction Pipeline)**:
   - A modularized implementation for training, evaluating, and deploying a deep learning model using TensorFlow.
   - Focused on the **Food-101** dataset, subset selection, and model optimization.
2. **Frontend (Flask Web Application)**:
   - Provides an intuitive user interface to upload images, classify food items, and retrieve nutritional information.

There is no deployment mechanism implemented in this project; it is intended to run locally on a developer's or user's machine.

➢ Detailed Components Analysis

1. Backend - Deep Learning Model

The backend is structured into several modular Python classes that handle specific tasks, ensuring scalability and reusability.

❖ ClassifierBase

- Serves as an **abstract base class** for any classifier implementation.
- Key attributes:
  - dataset_url: URL to download the dataset (Food-101 dataset in this case).
  - subset_classes: Number of food classes to use in the training pipeline.
  - input_shape: Dimensions of the input images.
  - batch_size: Number of images processed in a single training step.
- Defines abstract methods:
  - prepare_dataset(): For dataset preparation (implemented in subclasses).
  - create_generators(): For creating data generators for training and validation.

❖ DataHandler

- Handles dataset preparation:
  - Downloads the Food-101 dataset.
  - Extracts a subset of classes based on subset_classes and randomly samples images (images_per_class).
- Performs **directory cleanup and creation** for subsets.
- Facilitates dataset exploration with a method to list available data classes.

❖ **FoodClassifierWithCalories**
- Implements the specifics of food classification:
  - **EfficientNetB4 Backbone**:
    - Pretrained on ImageNet, used as a feature extractor.
    - Fine-tuned with additional dense layers for classification.
  - **Regularization**:
    - Dropout and L2 regularization to prevent overfitting.
  - **Loss Function**:
    - Uses categorical_crossentropy for multi-class classification.
- Integrates calorie mapping:
  - A JSON mapping associates each class with its calorie count.
- Provides a method to save class indices with calorie data to a JSON file for later use.

❖ **ModelTrainer**
- Manages the training process:
  - Applies **data augmentation** with ImageDataGenerator:
    - Techniques like rotation, flipping, and zooming improve generalization.
  - Implements **early stopping** and **learning rate reduction** to optimize training.
  - Saves the best-performing model during training using checkpoints.
- Handles evaluation:
  - Provides methods for validation and saving/loading models.

❖ **Visualizer**
- Adds functionality for predictions and visualization:
  - Visualizes predicted vs. actual classes along with calorie information.
  - Uses matplotlib to display image samples with corresponding predictions.

2. Frontend - Flask Web Application

The Flask application acts as the interface for users to interact with the trained model. The implementation includes multiple routes with tailored functionality.

Key Routes

1. **/ (Index Route)**:
   - Renders an HTML form for users to upload food images.

2. **/predict**:
   - Processes uploaded images:
     - Saves them in a static directory (static/uploads).
     - Preprocesses the image to match the input size of the model.
   - Predicts the class using the deep learning model.
   - Retrieves calorie information for the predicted class.
   - Displays the prediction and nutritional details (calories, protein, fat, carbohydrates, fiber).
   - Handles errors gracefully, such as unsupported file types or missing files.

3. **/search**:
   - Implements a **search feature** for food items:
     - Allows users to search for specific food names.
     - Returns all matching items with their nutritional information.

4. **/view/<food_name>**:
    - Displays detailed nutritional information for a specific food item.

❖ **Preprocessing Pipeline**

- Images are resized to 224x224 pixels (model input size).
- Pixel values are normalized to the range [0, 1].
- A batch dimension is added to allow compatibility with the trained model.

❖ **Templates and User Interaction**

- Uses HTML templates (index.html, result.html, search.html, details.html) for rendering pages.
- Passes data from the backend (predictions and nutritional details) to templates for dynamic rendering.

➢ **Notable Features**

❖ **Backend**

1. **Efficient Subset Selection**:
    - The DataHandler class allows selecting a subset of food classes and images, enabling experimentation with smaller datasets.
2. **Data Augmentation**:
    - Provides robust augmentation techniques to improve model generalization.
3. **Calorie Mapping**:
    - Extends classification by linking food classes to calorie data.
    - Allows integration of additional nutritional details like protein, fat, carbohydrates, and fiber.
4. **Class Balancing**:
    - Automatically computes class weights to address imbalanced datasets, ensuring fair model training.
5. **Visualization**:
    - Visualizes predictions alongside images for better interpretability.

❖ **Frontend**

1. **Comprehensive Search Functionality**:
    - Enables users to search for nutritional details by food name.
    - Dynamically filters results based on user input.
2. **Detailed Prediction Output**:
    - Displays predicted class, image, and detailed nutritional breakdown.
3. **Error Handling**:
    - Includes checks for missing files, unsupported file types, and prediction errors.

➢ **Suggestions for Improvement**

❖ **Backend Enhancements**

1. **Advanced Nutritional Mapping**:
    - Expand the calorie mapping to include detailed nutrition facts (e.g., sodium, vitamins).

2. **Better Augmentation**:
    - Include additional techniques like random cropping and channel shifts.
3. **Explainability**:
    - Add tools like Grad-CAM to visualize the regions of the image that influenced predictions.

❖ **Frontend Enhancements**

1. **Improved UI/UX**:
    - Use modern CSS frameworks like Materialize or custom designs for a polished look.
    - Add a progress indicator during image upload and prediction.
2. **File Validation**:
    - Add client-side validation for image uploads (e.g., size and format).
3. **Interactive Nutrition Dashboard**:
    - Provide a dashboard for users to explore calorie and nutritional data visually.

❖ **General Suggestions**

1. **Scalability**:
    - Add pagination or filtering for large datasets.
    - Modularize calorie mapping to handle updates efficiently.
2. **Testing**:
    - Implement unit tests for critical functions in both the backend and frontend.

➢ **Final Accuracy Result:**

- Training Accuracy achieved 94% and Validation Accuracy achieved 79%.

➢ **Potential Applications**

1. **Health and Fitness**:
    - Integrate with diet-tracking apps for automated meal logging.
2. **Educational Platforms**:
    - Serve as a tool for teaching concepts of nutrition and machine learning.
3. **Restaurant Menus**:
    - Automate menu calorie estimation for restaurants.
4. **Food Delivery Apps**:
    - Provide nutritional details for dishes in online food delivery platforms.