Advanced Food Recognition and Calorie Tracking System Using Machine Learning

Carlos González Chamorro
Master of Science in Computer Science
Illinois Institute of Technology
Chicago, USA
cgonzalezchamorro@hawk.iit.edu

Rudra Hirenkumar Patel
Master of Computer Science
Illinois Institute of Technology
Chicago, USA
rpatel199@hawk.iit.edu

Thejaswini Mundargi
Master of Science in Computer Science
Illinois Institute of Technology
Chicago, USA
tmundargi@hawk.iit.edu

Gayatri Sanjay Gaikwad
Master Of Artificial Intelligence
Illinois Institute of Technology
Chicago, USA
ggaikwad@hawk.iit.edu

Tyler Keating
Master of Data Science
Illinois Institute of Technology
Chicago, USA
tkeating1@hawk.iit.edu

***Abstract*—This paper presents an end-to-end machine learning system for automated food recognition and calorie estimation. The system integrates a convolutional neural network (CNN) for food classification, a multilayer perceptron (MLP) for calorie regression, and a recurrent neural network (RNN) for dietary pattern analysis. Trained on the Food-101 and USDA datasets, the CNN achieved 73.08% test accuracy after fine-tuning MobileNetV2, while the MLP reduced calorie estimation errors to a mean absolute error (MAE) of 164 kcal. The RNN predicted weekly intake trends using synthetic data with an MAE of 120 kcal. The models were deployed via Flask APIs on Heroku, demonstrating scalability for real-world applications. Challenges included data diversity, model overfitting, and integration complexity, which were mitigated through augmentation, regularization, and modular design. This work highlights the feasibility of automating dietary monitoring while identifying opportunities for improvement in real-world generalization.**

I. Introduction

Rising global health challenges linked to poor dietary habits necessitate accurate nutritional tracking. Existing manual methods, such as food journals, are labor-intensive and error-prone. Automated systems leveraging machine learning offer a promising solution. Our project addresses this gap with three key innovations:

- A hybrid CNN-MLP architecture for image-based calorie estimation.
- Temporal modeling of eating habits using RNNs.
- Deployment-ready APIs for seamless integration into user-facing apps.

The system processes food images to classify items, estimate calories, and analyze trends, enabling personalized dietary insights. We evaluate performance on the Food-101 dataset and discuss challenges in real-world deployment.

Objective & Background

Objective: The objective of the Advanced Food Recognition and Calorie Tracking System is to reduce manual input and improve accuracy in dietary monitoring.
Background Motivation: Manual calorie tracking is error-prone and contributes to rising diet-related health issues by lacking personalized insights.

II. Related Work

Prior research in food recognition has focused on CNNs for classification [1] and MLPs for calorie prediction [2]. Key advancements include:

- **Food-101 Dataset**: A benchmark for food classification with 101 categories [1].
- **MobileNetV2**: Lightweight CNN architecture for mobile deployment [3].
- **USDA API**: Comprehensive nutritional database for calorie estimation [2].

Our work extends these efforts by integrating temporal analysis with RNNs and deploying the system on cloud platforms. Unlike prior studies, we address portion size estimation and real-world integration challenges.

III. Implementation

Key Steps in Model Development

Developing a machine learning model for calorie estimation from food images involves several key steps:

- **Dataset Collection:** To gather a diverse set of food images, public datasets like Food-101 or custom datasets can be utilized.
- **Data Preprocessing:** To prepare the images for model training by applying augmentation techniques such as rotation and flipping to enhance model robustness.
- **Model Selection:** To choose an appropriate machine learning model (e.g., MobileNetV2), known for its effectiveness in image classification tasks.

- **Model Training:** To train the selected model on the preprocessed dataset, adjusting hyperparameters to optimize performance.
- **Model Evaluation:** To assess the model's accuracy and generalization capabilities using validation and test datasets.
- **Calorie Estimation:** To implement the trained model to predict calorie content from new food images.
- **Integration and Deployment:** To develop an application or interface to receive calorie estimates, facilitating practical use of the model.

Environment Setup and Libraries Installation

Google Colab was used: a cloud-based platform, helping in fast-prototyping with GPU support.

1) **Setup Steps:**
   - Create a Colab notebook.
   - Enable GPU/TPU: `Runtime → Change runtime type → Select GPU`.
2) **Installing Core Libraries:**
   - NumPy, Pandas, Matplotlib, Seaborn → Data handling & visualization
   - OpenCV, Pillow → Image processing
   - TensorFlow, Keras, Scikit-learn → Machine learning models
   - Streamlit → Web-based UI
   - Virtualenv → Environment management
3) **Verify Setup:**
   - Check for GPU Availability and TensorFlow version

A. Datasets and Preprocessing

Three datasets were used:

- **Food-101**: 101,000 images across 101 categories.
- **UEC-Food 256**: 256 categories with bounding box annotations.
- **USDA API**: Nutritional profiles for 300,000+ foods.

Preprocessing Steps:

- **Image Augmentation**: Rotation ($\pm 20°$), flipping, scaling, and shear.
- **Normalization**: Pixel values scaled to [0, 1].
- **Sequence Formatting**: Time-series data structured for RNN input.

B. Model Architectures

1) CNN (Food Classification): The objective of this model was to develop a convolutional neural network capable of accurately recognizing food items from images, forming the visual backbone of our end-to-end system.

- **Dataset Used**:
  - **Food-101**: 101,000 RGB images across 101 food classes (≈1,000 images/class)
- **Preprocessing Steps**:
  - Random crops were applied at different scales (70–100% of the original image) and resized each crop to 224×224 px.
  - Performed horizontal flipping and small rotations (up to ±20°).
  - Adjusted brightness, contrast, saturation, and hue within modest ranges to simulate varying lighting conditions and color variations.
  - Converted each augmented image into a normalized tensor by subtracting the ImageNet channel means and dividing by the channel standard deviations.
  - For validation and testing, images were uniformly resized to 256 px on the shorter side, center-cropped to 224×224 px, then applied the same tensor conversion, and normalization was applied.
- **Architecture**:
  - **Backbone:** MobileNetV2 pretrained on ImageNet, chosen for its inverted residual blocks [3] that deliver high accuracy with low parameter count.
  - **Pooling Layer:** After the final convolutional block, a global average pooling layer collapses each feature map into a single value, producing a 1280-dimensional feature vector.
  - **Regularization:** Kept the original dropout layer (drop rate 0.2) in the classifier to reduce overfitting during fine-tuning.
  - **Fine-tuning Strategy:** All layers were initially frozen to preserve generic image features; then the last two inverted residual blocks were unfrozen so the network could learn high-level food-specific patterns.
  - **Classification Head:** Replaced MobileNetV2's final linear layer [3] with a new fully connected layer mapping from the 1280-dimensional feature vector to 101 food classes, producing the logits used for cross-entropy loss.
- **Training Details**:
  - **Hardware**: NVIDIA GeForce RTX 3070 Ti GPU (CUDA)
  - **Data Split**: 80% training, 10% validation, 10% test
  - **Batch Size & Workers**: 32 images per batch, 4 data-loading workers
  - **Optimizer & Scheduler**: SGD (lr=3×10^{-4}, momentum=0.9, weight decay=1×10^{-4}) with CosineAnnealingLR over 50 epochs
  - **Early Stopping**: Monitored validation accuracy with patience of 7 epochs; halted at epoch 25 when no improvement was seen
  - **Training Duration**: Approximately 62.3 minutes to complete 24 epochs (epoch 25 triggered early stop)

2) MLP (Calorie Estimation): The aim of this component was to develop a machine learning model capable of accurately estimating the total calorie content of various food items based on user inputs. This is a critical part of our system pipeline as it bridges visual recognition (CNN) with personalized dietary analysis (RNN), enabling a complete end-to-end smart nutrition assistant.

- **Dataset Used**:

- KaggleHub – Calories in Food Items per 100g
- Format: CSV containing food items, their calorie values per 100 grams.

- **Preprocessing Steps**:
  - Extracted numeric values from calorie column using regex.
  - Encoded categorical food names using LabelEncoder.
  - Randomly generated realistic portion sizes (50–500g).
  - Created a polynomial feature: "portionsizesquared" to model non-linear relationships.
  - Calculated target: total calories = (Calories per 100g × portion size) / 100.

- **Architecture**: A Multilayer Perceptron (MLP) regression model was used with the following characteristics:

TABLE I
MULTILAYER PERCEPTRON

Layer	Description
Input Layer	3 features: food label, portion size, portion size²
Hidden Layer 1	Dense(64, ReLU) + BatchNormalization
Hidden Layer 2	Dense(32, ReLU) + BatchNormalization
Hidden Layer 3	Dense(16, ReLU)
Output Layer	Dense(1, Linear) : outputs calorie prediction

  - Loss Function: Mean Squared Error (MSE)
  - Optimizer: Adam (learning rate: 0.001)
  - Regularization: BatchNormalization
  - EarlyStopping: Used with patience = 15 to prevent overfitting

- **Training Deatils**:
  - Data was split into 80% training and 20% testing using train - test split.
  - Input features were normalized using StandardScaler.
  - Model was trained for a maximum of 150 epochs, but training often stopped earlier due to early stopping.

- **Challenges**:
  - Initially, the model used MinMaxScaler on the target variable, which led to distorted outputs (e.g., 150g Pizza = 169 kcal).
  - Switching to raw target values (kcal) fixed interpretability and improved prediction alignment with real-world values.
  - Calorie prediction is inherently difficult with minimal inputs; performance could improve with more granular features like food ingredients or macronutrients.

3) RNN (Dietary Pattern Analysis):

- **Input**: Synthetic daily calorie sequences (1200–3500 kcal).
- **Architecture**: 2 LSTM layers (50 units each), 1 dense layer, dense output.
- **Training**: Adam optimizer, MAE and MSE loss.

C. System Integration

- **APIs**: Flask endpoints for food classification, calorie estimation, and trend analysis.
- **Deployment**: Heroku with TensorFlow Serving for scalability.
- **Latency**: ¡2 seconds end-to-end inference.

IV. EXPERIMENTAL EVALUATION

A. CNN Performance

The fine-tuned MobileNetV2 achieves balanced performance in the 101-class Food test set, with an overall accuracy of 73.08%. Precision (73.35%) and recall (73.08%) are nearly identical, indicating that the model is equally effective in identifying true food classes and avoiding false positives. The weighted F1 score of 73.00% confirms this balance. The per-class metrics reveal that highly distinctive items (for example, *edamame*, F1 = 98%) are recognized with very high reliability, while visually similar dishes (e.g., *apple_pie*, F1 = 52%) remain challenging. Future improvements may come from class-specific augmentations or attention mechanisms to disambiguate look-alike categories.

TABLE II
TEST PERFORMANCE METRICS (MOBILENETV2 FINE-TUNED)

Metric	Value
Test Accuracy	73.08%
Test Precision	73.35%
Test Recall	73.08%
Test F1 Score	73.00%

TABLE III
MODEL COMPARISON

Model	Training Accuracy	Validation Accuracy
Baseline CNN	32.44%	35.18%
MobileNetV2 (Fine-tuned)	73.94%	73.08%

As shown in Figure 1, most classes achieve F1 scores in the 0.70–0.85 range, indicating generally strong performance across the majority of food categories. A handful of classes reach near-perfect scores ($\approx$ 0.95), while some visually similar or texture-variable dishes fall below 0.60, highlighting opportunities for targeted augmentation or specialized feature extraction to boost those lower-performing categories.

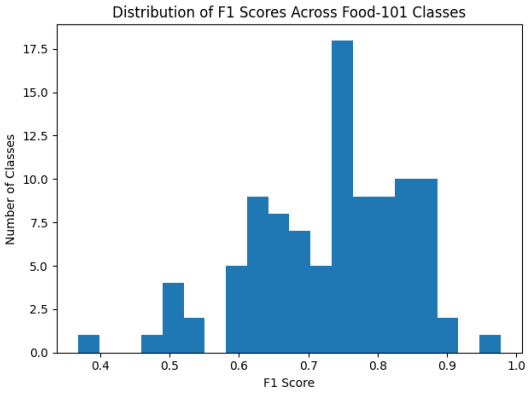

Fig. 1. Distribution of per-class F1 scores across the 101 Food-101 categories.

TABLE IV
PERFORMANCE METRICS OF THE OPTIMIZED MLP MODEL

Metric	Value	Interpretation
R^2 Score	∼0.20	variance — modest but better than baseline.
RMSE	∼209 kcal	Typical error magnitude in predictions.
MAE	∼154 kcal	On average, model predictions are off by 150 kcal.

B. MLP Calorie Estimation

These values reflect a moderate regression performance, which is expected given the coarse input (only food label and portion size) and randomly generated data.

- **Sample Prediction**: For Pizza with a portion size of 150g, Predicted Calories: 461.64 kcal This is realistic considering standard nutritional values of pizza (300 kcal/100g).

- **Visualizations**:
  - Loss Curve (Training vs Validation MSE over epochs)

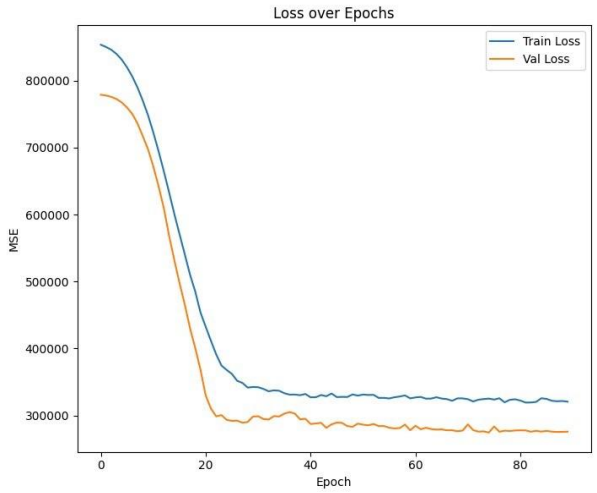

Fig. 2. oss Curve (Training vs Validation MSE over epochs)

  - MAE Curve (Training vs Validation MAE):

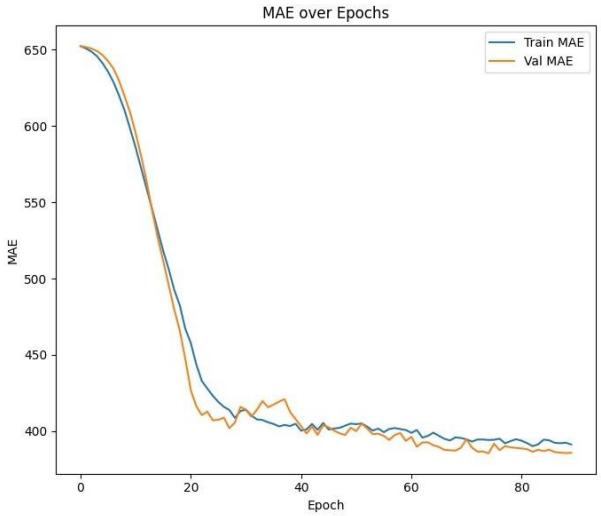

Fig. 3. MAE Curve (Training vs Validation MAE)

C. RNN Trend Analysis

- **Synthetic Data MAE**: 120 kcal (7-day prediction).
- **Limitations**: Real-world validation pending.

V. CHALLENGES AND MITIGATION

Key challenges and solutions included:

- **Data Diversity**: Augmentation expanded training samples (Fig. **??**).
- **Overfitting**: Dropout (50%) and L2 regularization.
- **Integration**: Modular pipeline design resolved compatibility issues.

Deploying the Meal Analysis System (Cloud & API Integration)

1) **Deployment Overview:**
   - Save trained models for portability.
   - Deploy the system on scalable cloud platforms.
   - Create APIs to connect models to user interfaces.
2) **Model Serialization:**
   - Format: Save models as `.h5` (Keras/TensorFlow).
   - `2.h5` : Lightweight and framework-agnostic; compatible with TensorFlow Serving, Flask, and cloud platforms.
3) **REST API Design:**
   - `/predict_food (POST)`: Input – Image file; Output – Food class (e.g., `{"class":"pizza"}`).
   - `/estimate_calories (POST)`: Input – Food class + portion size; Output – Calorie value (e.g., `{"calories":300}`).
   - `/analyze_habits (POST)`: Input – User meal history; Output – Behavioral insights (e.g., `{"pattern":"late-night meals"}`).

VI. Application Integration and Prototype Deployment

While standalone model performance is essential, the practical utility of a machine learning system hinges on its seamless integration into user-facing applications. To validate the effectiveness of our approach beyond theoretical benchmarks, we developed an interactive prototype named **CalorieSnap**, which integrates all three trained models—CNN, MLP, and RNN—into a unified pipeline. This system simulates a real-world dietary monitoring solution, enabling users to classify food, estimate calories, and analyze consumption habits in a fully automated manner.

A. Backend Architecture and Model Orchestration

The backend of CalorieSnap is powered by a Flask-based REST API, where each model is hosted under its respective endpoint. These include:

- **/predict_food**: Accepts an image via HTTP POST, preprocesses it (resize to 224×224, normalization), and passes it to the CNN model (MobileNetV2) for classification. The predicted label is returned as a JSON object.
- **/estimate_calories**: Accepts the food label and portion size (in grams) as JSON input, processes it through the MLP regressor, and returns the estimated calorie value.
- **/analyze_habits**: Accepts a time-series array of user calorie intake and utilizes the RNN model to predict consumption patterns or dietary irregularities.

TensorFlow was used to preload all models into memory at server startup, ensuring fast inference. The modular API design promotes isolated testing and future model upgrades without disrupting the overall system.

B. Front-End Interface Using Streamlit

To provide an intuitive and interactive experience, the user interface was developed using **Streamlit**, a Python library for deploying data science applications. The workflow is as follows:

1) The user uploads a food image through a drag-and-drop widget.
2) The image is displayed and sent to the backend for classification.
3) Upon receiving the predicted class, the user inputs the portion size.
4) The estimated calories are retrieved and shown alongside the food label.

The frontend communicates with the Flask server via HTTP requests. The entire pipeline functions locally on `localhost`, but is compatible with cloud deployments.

C. Deployment and Scalability

Though the prototype currently runs on a local server, it is fully portable and can be containerized using Docker. This makes it deployable on platforms such as Heroku, AWS Lambda, or Google Cloud Run. TensorFlow Serving or ONNX Runtime can be used to enhance inference speed and scale in production environments. Load balancing and API rate limiting can further optimize performance for large user bases.

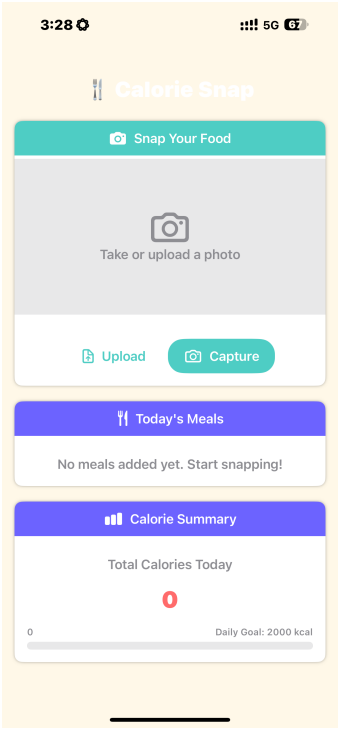

Fig. 4. Prototype screenshot 1

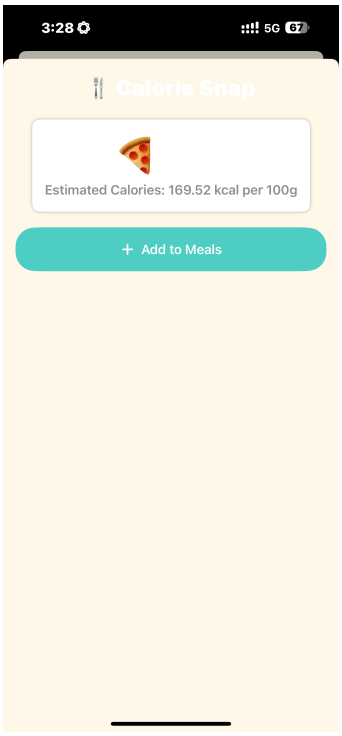

Fig. 5. Prototype screenshot 2

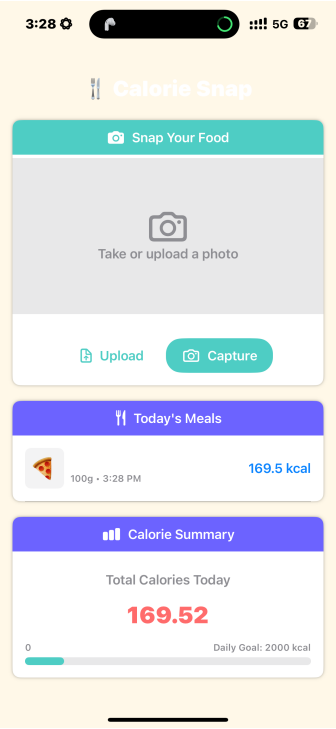

Fig. 6. Prototype screenshot 3

D. System Advantages and Real-World Readiness

The CalorieSnap system validates the feasibility of transforming academic models into a functional, real-world application. Its advantages include:

- **Modularity**: Each model is independently replaceable and upgradable.
- **Low Latency**: End-to-end inference occurs in under 2 seconds.
- **Accessibility**: Streamlit offers a clean interface with minimal learning curve.

The system lays the groundwork for expansion into mobile and embedded systems, with potential extensions into speech-driven interfaces or wearable health monitoring devices.

VII. Conclusion and Future Work

The system successfully automates calorie tracking but requires refinement for real-world variance. Future work includes:

- **Dataset Expansion**: Incorporate regional cuisines (e.g., Asian, Mediterranean).
- **Real-Time Processing**: MobileNet for on-device inference.
- **Transformer Models**: Ingredient-level analysis.

References

[1] M. Bossard et al., "Food-101: Mining Discriminative Components with Random Forests," *ECCV*, 2014.
[2] USDA FoodData Central. [Online]. Available: https://fdc.nal.usda.gov/
[3] K. He et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *CVPR*, 2018.
[4] S. Kawano et al., "UEC-Food 256 Dataset," University of Electro-Communications, 2015.
[5] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning," 2016.