UNIVERSAL FOOD RECIPE GENERATOR

Capstone Project Report

Deep Learning-based Food Recognition System
with Recipe Generation

Date: January 20, 2026

Technologies: Python, PyTorch, EfficientNet-B0, Gradio
Dataset: 181 Food Categories | 113,900 Training Images
Accuracy: 84.8%

1. Executive Summary

This project presents a Universal Food Recipe Generator that uses deep learning to recognize food items from images and provide corresponding recipes. The system can identify 181 different food categories, including 80 Indian dishes and 101 Western/International dishes.

KEY ACHIEVEMENT: 84.8% accuracy on 181 food categories with minimal overfitting (+2.5% gap)

Project Highlights

- Trained a deep learning model using EfficientNet-B0 with transfer learning
- Combined Indian Food Images (80 classes) and Food-101 (101 classes) datasets
- Achieved 84.8% validation accuracy with only +2.5% train-val gap
- Developed a web-based interface using Gradio for easy interaction
- Implemented comprehensive recipe database with 181 complete recipes

Technologies Used

- Python 3.x - Primary programming language
- PyTorch 2.6.0 - Deep learning framework
- EfficientNet-B0 - Pre-trained CNN architecture
- CUDA 12.4 - GPU acceleration
- Gradio - Web interface framework
- NVIDIA RTX 3050 - Training hardware (4GB VRAM)

2. Problem Statement

Food recognition is a challenging computer vision task with applications in dietary tracking, restaurant menu digitization, and recipe recommendation systems. Existing food recognition models are primarily focused on Western cuisine and fail to recognize foods from other cultures, particularly Indian cuisine.

The Challenge

When testing the original inversecooking model (Facebook Research) with Indian food images, the model completely failed to recognize them. For example:

TEST CASE:
- Input Image: Garlic Naan (Indian bread)
- Model Prediction: SALMON (78.3% confidence)
- Actual Food: Naan

This misclassification occurred because:

- The original model was trained exclusively on Western cuisine (Recipe1M dataset)
- Indian food has distinct visual characteristics not present in Western training data
- The model had zero exposure to Indian culinary items during training

Resear

There is a significant gap in food recognition systems for multicultural cuisine support. Most existing models are biased toward Western food categories, leaving a large portion of global cuisine unrecognized. This project aims to bridge this gap by creating a unified model that recognizes both Indian and Western cuisines.

3. Objectives

Primary Objectives

- Develop a food recognition model capable of identifying both Indian and Western cuisines
- Achieve high accuracy (>80%) while avoiding overfitting
- Create a user-friendly web interface for practical application
- Build a comprehensive recipe database for recognized foods

Secondary Objectives

- Understand and address the overfitting problem in deep learning
- Implement transfer learning techniques effectively
- Optimize model training for limited GPU resources (4GB VRAM)
- Document the complete development process for reproducibility

Success Criteria

Metric	Target	Achieved
Validation Accuracy	>80%	84.8%
Overfitting Gap	<10%	+2.5%
Food Categories	>100	181
Web Interface	Functional	Complete
Recipe Database	All classes	181 recipes

4. Dataset Description

4.1 Indian Food Images Dataset

Source: Kaggle (iamsouravbanerjee/indian-food-images-dataset)

Property	Value
Total Categories	80 Indian dishes
Total Images	~4,000
Images per Class	~50
Image Format	JPEG/PNG
Resolution	Variable (resized to 224x224)

Sample categories: Biryani, Butter Chicken, Naan, Dosa, Idli, Samosa, Gulab Jamun, Jalebi, Paneer Tikka, Dal Makhani, Palak Paneer, Chole, Paratha, and 67 more.

4.2 Food-101 Dataset

Source: ETH Zurich Computer Vision Lab (via torchvision)

Property	Value
Total Categories	101 food types
Total Images	101,000
Images per Class	1,000
Train/Test Split	750/250 per class
Image Format	JPEG

Sample categories: Pizza, Hamburger, Sushi, Ramen, Tacos, Steak, Cheesecake, Ice Cream, Pancakes, French Fries, Hot Dog, Pasta, and 89 more.

4.3 Combined Dataset Statistics

FINAL COMBINED DATASET: 181 classes, 113,900 training images

Cuisine	Categories	Train Images	Val Images
Indian	80	~3,200	~800
Western	101	~75,750	~19,960
TOTAL	181	113,900	20,760

4.4 Data Preprocessing

- All images resized to 224x224 pixels (EfficientNet input size)
- Normalized using ImageNet mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225]
- Training augmentation: Random horizontal flip, rotation (15 deg), color jitter
- Validation: Only resize and normalize (no augmentation)
- Class prefixes added: 'indian_' and 'western_' for identification

5. Model Architecture

5.1 Base Architecture: EfficientNet-B0

EfficientNet is a family of convolutional neural networks that achieve state-of-the-art accuracy with fewer parameters through compound scaling. EfficientNet-B0 is the baseline model, offering an excellent balance between accuracy and computational efficiency.

WHY EFFICIENTNET-B0?
- Pre-trained on ImageNet (1000 classes, 1.2M images)
- Only 5.3M parameters (fits in 4GB VRAM)
- Efficient inference for web application
- Excellent transfer learning performance

5.2 Architecture Diagram

```
Input Image (224 x 224 x 3)
         |
         v
+--------------------+
| EfficientNet-B0    | <-- Pre-trained on ImageNet
| (Feature Extractor)|     (1280 output features)
+--------------------+
         |
         v
+--------------------+
| Adaptive Avg Pool  |
+--------------------+
         |
         v
+--------------------+
| Dropout (0.25)     |
+--------------------+
         |
         v
+--------------------+
| Linear (1280->512) |
+--------------------+
         |
         v
+--------------------+
| BatchNorm + ReLU   |
+--------------------+
         |
         v
+--------------------+
| Dropout (0.25)     |
+--------------------+
         |
         v
+--------------------+
| Linear (512->181)  | <-- 181 food classes
+--------------------+
         |
         v
    Output Logits
```

5.3 Model Parameters

Component	Details
Backbone	EfficientNet-B0 (pre-trained)
Feature Dimension	1280
Hidden Layer	512 neurons
Output Classes	181
Dropout Rate	0.25
Total Parameters	~5.3 million
Trainable (Phase 1)	~1.4 million (classifier only)
Trainable (Phase 2)	~3.9 million (partial unfreeze)

5.4 Transfer Learning Strategy

Transfer learning leverages knowledge from a pre-trained model (ImageNet) and adapts it to a new task (food recognition). This approach is essential when:

- Limited training data is available
- Computational resources are constrained
- The source domain (ImageNet) shares features with target domain (food images)

The EfficientNet backbone already understands basic visual features (edges, textures, shapes, colors) from ImageNet training. We only need to train it to recognize food-specific patterns.

6. Training Methodology

6.1 Two-Phase Training Approach

We implemented a two-phase training strategy to effectively leverage transfer learning:

PHASE 1: Warmup (3 epochs)
- Freeze EfficientNet backbone completely
- Train only the classifier head
- Learning rate: 0.001
- Purpose: Adapt classifier to new classes without disrupting pre-trained features

PHASE 2: Fine-tuning (22 epochs)
- Unfreeze last 3 blocks of EfficientNet
- Train both backbone and classifier
- Learning rate: 0.0001 with cosine annealing
- Purpose: Adapt backbone features to food domain

6.2 Training Hyperparameters

Hyperparameter	Value
Optimizer	AdamW
Learning Rate (Phase 1)	0.001
Learning Rate (Phase 2)	0.0001
Weight Decay	0.01
Batch Size	32
Epochs (Total)	25 (3 warmup + 22 fine-tune)
LR Scheduler	Cosine Annealing
Loss Function	Cross Entropy
Dropout	0.25

6.3 Data Augmentation

To improve generalization and reduce overfitting, we applied the following augmentations:

Augmentation	Parameters
Random Horizontal Flip	p=0.5
Random Rotation	+/- 15 degrees
Color Jitter	brightness=0.2, contrast=0.2
Random Resized Crop	scale=(0.8, 1.0)
Normalization	ImageNet mean/std

6.4 Hardware Configuration

Component	Specification
GPU	NVIDIA GeForce RTX 3050 Laptop
VRAM	4 GB
CUDA Version	12.4
PyTorch Version	2.6.0+cu124
Training Time	~8.5 hours

7. Experimental Results

7.1 Model Evolution

We conducted multiple experiments to understand and address the overfitting problem:

Model	Dataset	Classes	Accuracy	Gap	Status
V1	Small	90	65.5%	+32%	Overfitting
V2	Small	90	45.6%	-13%	Underfitting
V3	Small	90	63.6%	+15.5%	Slight overfit
FINAL	Large	181	84.8%	+2.5%	EXCELLENT!

7.2 Understanding the Gap Metric

The "Gap" represents the difference between training accuracy and validation accuracy:

- Gap > 20%: Severe overfitting (model memorized training data)
- Gap 10-20%: Moderate overfitting
- Gap 0-10%: Healthy generalization
- Gap < 0%: Underfitting (model too constrained)

FINAL MODEL GAP: +2.5% - Indicates excellent generalization!

7.3 Training Progress (Final Model)

Epoch	Train Acc	Val Acc	Gap	Notes
3	37.4%	51.3%	-13.8%	End of warmup
10	76.1%	80.3%	-4.2%	Gap closing
16	83.4%	83.5%	-0.1%	Near perfect
25	87.2%	84.8%	+2.5%	Final (Best)

7.4 Key Findings

FINDING 1: More Data > More Regularization
With only 35 images per class, no amount of dropout or augmentation could prevent overfitting. With 629 images per class, overfitting was virtually eliminated.

FINDING 2: Transfer Learning is Highly Effective
Using pre-trained EfficientNet-B0 allowed us to achieve 84.8% accuracy with only 113K images, which would be impossible training from scratch.

FINDING 3: Two-Phase Training Works
The warmup phase prevented catastrophic forgetting of pre-trained features, while fine-tuning adapted them to the food domain.

7.5 Accuracy Improvement Summary

Metric	Before (V1)	After (Final)
Validation Accuracy	65.5%	84.8%
Overfitting Gap	+32%	+2.5%
Food Categories	90	181
Training Images	3,150	113,900

The final model achieved a 19.3% improvement in accuracy and reduced overfitting by 29.5 percentage points!

8. Challenges and Solutions

Challenge 1: Model Could Not Recognize Indian Food

PROBLEM: The original inversecooking model was trained on Recipe1M (Western cuisine only) and completely failed on Indian food images.

SOLUTION: Downloaded Indian Food Images dataset (80 categories) and combined it with Food-101 to create a multicultural training set.

Challenge 2: Severe Overfitting (V1 Model)

PROBLEM: First model achieved 96% training accuracy but only 65% validation accuracy - a 32% gap indicating the model memorized training data.

SOLUTION: Initially tried heavy regularization (V2) which caused underfitting. Final solution was to increase dataset size from 3K to 113K images, which naturally prevented overfitting.

Challenge 3: Limited GPU Memory (4GB VRAM)

PROBLEM: RTX 3050 Laptop has only 4GB VRAM, limiting batch size and model complexity.

SOLUTION: Used EfficientNet-B0 (smallest variant) with batch size 32. Implemented efficient data loading and mixed precision concepts.

Challenge 4: PyTorch 2.x Compatibility

PROBLEM: Original inversecooking code used torch.uint8 for masks which is deprecated in PyTorch 2.x.

SOLUTION: Updated mask dtype from torch.uint8 to torch.bool throughout the codebase.

Challenge 5: Class Imbalance

PROBLEM: Food-101 has 1000 images per class while Indian dataset has only 50 images per class.

SOLUTION: Used weighted sampling and data augmentation to balance the representation during training.

9. Web Application

9.1 Interface Overview

The web application provides an intuitive interface for users to upload food images and receive recipe recommendations. Built using Gradio, it offers:

- Drag-and-drop image upload
- Real-time food recognition
- Top 5 predictions with confidence scores
- Complete recipe display (ingredients + instructions)
- Cuisine type identification (Indian/Western)

9.2 App

Feature	Description
Image Upload	Accepts JPEG, PNG, WebP formats
Recognition	Identifies food in <1 second (GPU)
Predictions	Shows top 5 foods with confidence %
Recipes	181 complete recipes with ingredients
Accessibility	Web-based, no installation needed

9.3 How to Run

```
# Start the web application
cd inversecooking/src
python web_app_large.py

# Open in browser
http://127.0.0.1:7860
```

9.4 Supported Foods

INDIAN (80 dishes): Biryani, Butter Chicken, Naan, Dosa, Idli, Samosa, Gulab Jamun, Jalebi, Paneer Tikka, Chole, Dal Makhani, Palak Paneer, Paratha, Pav Bhaji, Rasgulla, and 65 more...

WESTERN (101 dishes): Pizza, Hamburger, Sushi, Ramen, Tacos, Steak, Pasta, Cheesecake, Ice Cream, French Fries, Hot Dog, Caesar Salad, Tiramisu, Pancakes, Waffles, and 86 more...

10. Conclusions and Future Work

10.1 Conclusions

This project successfully developed a Universal Food Recipe Generator capable of recognizing 181 different food categories from both Indian and Western cuisines. Key achievements include:

- 84.8% validation accuracy with minimal overfitting (+2.5% gap)
- Successful integration of multicultural food datasets
- Effective use of transfer learning with EfficientNet-B0
- Functional web application for practical use
- Comprehensive recipe database for all recognized foods

The project demonstrated that the overfitting problem in deep learning can be effectively addressed through increased training data rather than excessive regularization.

10.2 Limitations

- Accuracy varies by food category (some visually similar foods are confused)
- Limited to 181 predefined categories
- Requires GPU for real-time inference
- Indian dataset has fewer images per class than Western dataset

10.3 Fu

- Expand to more cuisines (Chinese, Mexican, Mediterranean, etc.)
- Implement multi-label classification for dishes with multiple items
- Add nutritional information to recipes
- Deploy as cloud-based API for mobile applications
- Implement ingredient detection for partially visible foods
- Add user feedback loop to improve model over time

11. Technical Specifications

11.1 Software Requirements

Package	Version
Python	3.8+
PyTorch	2.6.0+cu124
torchvision	0.21.0+cu124
Gradio	Latest
Pillow	Latest
NumPy	Latest

11.2 Hardware Requirements

Component	Minimum Specification
GPU	NVIDIA GPU with 4GB+ VRAM
RAM	8 GB
Storage	20 GB (for datasets and model)
CUDA	11.0+

11.3 Project File Structure

```
inversecooking/
|-- src/
|   |-- web_app_large.py      # Main web application
|   |-- train_large_model.py  # Training script
|   |-- SESSION_NOTES.md      # Project documentation
|
|-- data/
|   |-- large_model/          # Trained model files
|   |-- indian_food/          # Indian food dataset
|   |-- large_food_dataset/   # Food-101 dataset
|
|-- docs/                     # PDF documentation
|-- archive/                  # Old/unused files
```

12. References

1. Tan, M., & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML 2019.

2. Bossard, L., Guillaumin, M., & Van Gool, L. (2014). Food-101 - Mining Discriminative Components with Random Forests. ECCV 2014.

3. Salvador, A., et al. (2017). Learning Cross-modal Embeddings for Cooking Recipes and Food Images. CVPR 2017.

4. Banerjee, S. (2020). Indian Food Images Dataset. Kaggle. https://www.kaggle.com/datasets/iamsouravbanerjee/indian-food-images-dataset

5. Facebook Research. inversecooking - Cooking with Neural Networks. GitHub Repository. https://github.com/facebookresearch/inversecooking

6. PyTorch Documentation. https://pytorch.org/docs/stable/index.html

7. Gradio Documentation. https://gradio.app/docs/

--- End of Report ---