AI-Powered Food Classification and Calorie Estimation Using the Food-101 Dataset

Abdallah Waked

INFO-6147 Deep Learning with PyTorch

Dr. Mohammed Yousefhussien

Apr 12, 2025

Table of Contents

Proposal

AI-Powered Food Classification and Calorie Estimation Using the Food-101 Dataset

Project Description:

The goal of this project is to estimate the caloric value of food based on its category using the Food-101 dataset. The Food-101 dataset is a vast resource for training a Convolutional Neural Network (CNN) for food classification. Once classified, the identified food category will be queried in a nutritional database (JSON file) to approximate its caloric value.

Why is it good?

This project makes it easier to identify different foods and track their calorie count easily. Instead of going in search of nutritional information manually, users simply take a picture of the food and get an immediate estimate of the calories. This is especially helpful for individuals who are trying to eat healthy or track their diet.

How do you think you will do it?

I will achieve this by training the Food-101 dataset using a Convolutional Neural Network (CNN) to classify food images in PyTorch as a machine learning library. To enhance classification accuracy, I will utilize a pre-trained model such as ResNet18. Additionally, I will define and optimize the best hyperparameters to ensure optimal performance. After training, I will test the model to achieve high accuracy while preventing overfitting. Finally, I will integrate the model with a JSON file to retrieve the caloric value of each classified food item.

What data will you use?

I will use Food-101dataset, and I will split it into 70% training, 15% validation, and 15% testing for effective model evaluation. (Link)

How will you evaluate your system performance?

I will evaluate the system's performance by checking how accurately it classifies food images, which means calculating the percentage of correctly identified images. I'll also compare training loss and validation loss using a graph to spot any signs of overfitting. If the model is overfitting, I'll fine-tune it by adjusting hyperparameters, applying regularization techniques, or using data augmentation to help it generalize better. The goal is to make sure the model works well on new, unseen images rather than just memorizing the training data.

System Design

To make the project accessible to users and simulate a real-world application, I divided the project into two parts:

1. Training Application

Implemented in Python using PyTorch and includes data loading, preprocessing, model training, checkpointing, evaluation, and exporting of the trained model.

Data loading

To optimize performance and avoid redundant downloads, the application first checks if the model (approximately 5 GB) is already stored locally

Data Summary

Dataset Used: Food-101

Classes: 101 food categories

Total Images: around 101,000

Data Split:

- 70% Training
- 15% Validation
- 15% Test

Data Preprocessing and Augmentation Techniques

To prepare the dataset for training and improve model generalization, the following preprocessing steps were applied:

Resizing: All images are resized to 224×224 pixels, aligning with the input requirements of ResNet18.

Data Augmentation:

Random Horizontal Flip: Increases dataset variability by flipping images horizontally.

Random Rotation: Randomly rotates images to make the model robust to orientation changes.

Normalization: Images are normalized using the standard ImageNet mean and standard deviation, ensuring consistency with the pretrained model parameters.

Dataset Splitting: The dataset is split into training, validation, and test sets to facilitate accurate model evaluation and avoid overfitting

Model Architecture (**ResNet18)**

Structure:

Stage	**Layer Details**	**Output Size**
Input	Image Input	3 × 224 × 224
Conv1	7×7 conv, 64 filters, stride 2 + BatchNorm + ReLU	64 × 112 × 112
MaxPool	3×3 max pool, stride 2	64 × 56 × 56
Layer1	2 × BasicBlock (64 filters)	64 × 56 × 56
Layer2	2 × BasicBlock (128 filters), stride 2	128 × 28 × 28
Layer3	2 × BasicBlock (256 filters), stride 2	256 × 14 × 14
Layer4	2 × BasicBlock (512 filters), stride 2	512 × 7 × 7
AvgPool	Global average pooling	512
Fully Connected (fc)	Linear layer → 1000 classes (ImageNet) 101 classes	101

Experiment Modes:

Mode	**Description**	**Trainable Parameters**
Baseline	Random init, no pretrained	All layers
Pretrained (Freeze)	Use ImageNet weights, freeze all layers except layer4 + fc	Layer4, fc only
Pretrained (Unfreeze)	Use ImageNet weights, allow all layers to be updated	All layers fine-tuned

Loss Function: CrossEntropyLoss

Optimizer: Adam Optimizer

Training Process & Hyperparameter Tuning

Device Used: GPU (if available), with CPU fallback for local testing.

Epochs: Variable, typically around 10 epochs with early stopping to prevent overfitting.

Batch Size: Started with 32, later reduced to 8 due to device limitations.

Learning Rate: Initially set to 0.001. Planned to reduce to 0.0003 for fine-tuning, but this could not be applied due to Colab quota limits and slow local device performance.

Training Features:

Progress Monitoring: tqdm progress bars.

Checkpointing: Model saved after each epoch.

Best Model Saving: Separate storage for the best-performing model.

Training History: Saved for later visualization and analysis.

Experiments Conducted:

Learning Rates: 0.001 and planned 0.0003.

Batch Sizes: 32 initially, reduced to 8.

Observations:

Pretrained models significantly outperformed the baseline.

Freezing early layers accelerated training.

Unfreezing all layers improved accuracy at the cost of longer training time.

Evaluation Results

Metrics Used:

Accuracy

Classification Report (Precision, Recall, F1-Score)

Confusion Matrix

Results Summary:

Best accuracy achieved: (you will fill after testing, e.g., 82.4%)

Pretrained Unfreeze model performed the best overall.

Misclassifications primarily occurred between visually similar food items.

Visualizations:

Training/Validation loss and accuracy curves

Confusion matrix heatmap

2. Deployment Application (Web App)

The second part of the project is a Streamlit web application hosted online:

Link: https://food-101-web-dl.streamlit.app/

To handle the large model file size and avoid manual uploads, the app automatically downloads the trained model from Google Drive at runtime. This effectively bypasses file size limitations on GitHub.

Features:

Users can upload an image of food to classify.

Once the model predicts a food category, users can click on the prediction to explore recipe options.

The application makes a live API call to Spoonacular (https://api.spoonacular.com/) to retrieve relevant recipes and nutritional facts for the selected dish.

GitHub Repository:

Repository: https://github.com/akw-waked/Food-101-Web

Includes:

Training code

Streamlit web application code

Model loading and API integration logic

The Streamlit application is designed to automatically download the model from Google Drive during runtime, allowing efficient loading without manual steps.

Future Improvements

Explore lighter architectures such as MobileNetV2 for faster inference.

Increase data augmentation variety to improve model generalization.

Integrate real-time camera input for user-friendly calorie estimation.

Connect to an external, more extensive nutritional database.

Challenges and Limitations

While the project is well-planned, several potential challenges and limitations have been identified:

- **Large Model Size:**
  Pretrained models like ResNet18 increase accuracy but result in larger model files, which may cause limitations in cloud environments like Google Colab (quota limits) and local devices with limited storage and processing power.
- **Colab GPU Quota:**
  Free-tier usage of Google Colab provides limited GPU time. Extended training sessions or hyperparameter tuning might exceed the available quota.
- **Hardware Constraints:**
  Local device performance may not be sufficient for intensive training, requiring adjustments such as reducing batch size or training time.
- **Dataset Complexity:**
  The Food-101 dataset contains diverse food images with varying quality, backgrounds, and lighting conditions, which may challenge the model's generalization.
- **API Limitations:**
  Reliance on the Spoonacular API introduces dependency on external service availability and API rate limits, which could impact real-time data retrieval.
- **Time Constraints:**
  Due to time limitations, advanced fine-tuning techniques (such as lower learning rate fine-tuning and extended epochs) may not be fully implemented.

Despite these challenges, careful planning such as using model checkpointing, batch size adjustments, and leveraging pretrained models will help mitigate many of these limitations.

Conclusion

This project successfully accomplished its objective of classifying food images and presenting corresponding caloric values. By utilizing the Food-101 dataset for dish prediction and integrating the Spoonacular API for real-time retrieval of food details, the application delivers practical and informative results. Leveraging pretrained models, particularly ResNet18, significantly enhanced classification accuracy and reduced training time, making the solution both efficient and effective.