CSCI 184: Applied Machine Learning Final Project Report
Title: Food Recognition App (IOS platform)
Name: Xijing Wang (Thomas)

Introduction

Inspiration and Motivation

As someone who is passionate about maintaining a healthy diet, I often find myself wanting to better understand the nutritional value of my meals, particularly the calorie content and ingredients. An application capable of providing such insights would be invaluable for effective diet management. However, building such an app is a complex challenge that requires a solid foundation in food recognition technology. Accurately identifying food items is a crucial first step before integrating supplementary information like calorie counts and ingredient details. This project serves as a foundational effort to address this preliminary task, aiming to pave the way for a robust solution that empowers individuals to take control of their dietary habits with confidence and precision. In this project, I will demonstrate how to utilize an existing model and dataset to train an effective deep learning model focused on food recognition. Additionally, I will outline the process of adapting the trained model to fit the iOS platform architecture, enabling its integration into an iOS application.

Method

Dataset

Overview

The dataset used is Food-101 from Kaggle, featuring 101 food categories. It is pre-split into training and testing sets with train.txt and test.txt, saving significant preparation time. However, additional preprocessing is needed, including resizing images, normalizing pixel values, and formatting the data for input into the MobileNetV2 model. Each type has 1000 images with 750 as training and 250 as testing.

The Food-101 dataset was originally used in the research paper "Food-101 – Mining Discriminative Components with Random Forests" by Bossard, L., Guillaumin, M., and Van Gool, L., presented at the 13th European Conference on Computer Vision (ECCV) in 2014.

Dataset Customization

Due to limitations of my computer, I only choose 15 classes from the 101 dataset to be in this project.

- **class_labels =** ['cheesecake', 'chocolate_cake', 'chocolate_mousse', 'fish_and_chips', 'french_fries', 'fried_calamari', 'fried_rice', 'gyoza', 'hamburger', 'hot_dog', 'oysters', 'pho', 'pizza', 'sushi', 'tacos']
- **Image Dimensions**: Standardized to 224x224 pixels
- **Training sample size:** 11250 images belonging to 15 classes, each type has 750 images.
- **Validation sample size:** 3750 images belonging to 15 classes, each type has 250 images.
- Using **Data Augmentation** in Training images in **ImageDataGenerator** function to improve generalization. Augmentations include:
    - Rescaling pixel values (1./255).
    - Random rotations, shifts, zooming, and horizontal flips.

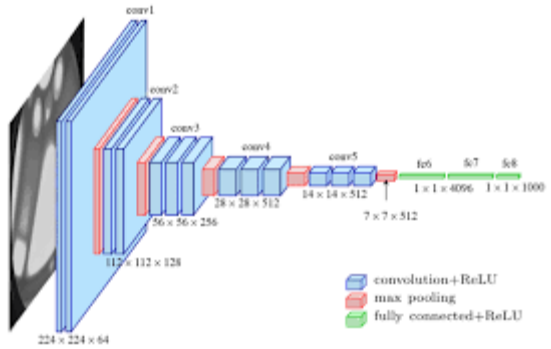
Model

In this task, I selected MobileNetV2 as the CNN model for the food recognition task. As it suggests by its name, MobileNetV2 is a lightweight convolutional neural network that is particularly well-suited for mobile application. While it may not achieve the highest accuracy compared to some larger models such as ResNet, EfficientNetV2, NASNetLarge and etc., its compact size and fast inference time make it an preferable choice for delivering a responsive and efficient experience on mobile devices for the users. The primary reason for choosing MobileNetV2 is its ability to strike a balance between accuracy and model size, which is critical for real-world applications with limited computational resources. This trade-off ensures optimal performance while maintaining usability, especially in mobile environments.

Compared to other compact neural networks such as NASNetMobile (5.3 MB) and EfficientNetV2B2 (10.2 MB), MobileNetV2 (3.5 MB) is significantly smaller—approximately 151% and 219% smaller, respectively. Although NASNetMobile and EfficientNetV2B2 achieve higher accuracy, as reported in the Keras Applications benchmark, the simplicity of my dataset and task makes MobileNetV2 a more suitable choice. Its smaller size ensures greater efficiency, which is particularly important for mobile applications.

However, if future tasks become more complex—such as incorporating food ingredient recognition or food weight estimation—I would consider models like NASNetMobile or EfficientNetV2B2 to prioritize accuracy while still maintaining reasonable efficiency.

Bottom graph is Visualization of Model Structure, and right one is detailed explanation of original model structure in MobileNetV2



Layer Type	Input Size	Output Size	Kernel Size	Stride	Expansion Factor
Initial Conv	224x224x3	112x112x32	3x3	2	-
Inverted Residual Block	112x112x32	112x112x16	3x3	1	1
Inverted Residual Block x2	112x112x16	56x56x24	3x3	2	6
Inverted Residual Block x3	56x56x24	28x28x32	3x3	2	6
Inverted Residual Block x4	28x28x32	14x14x64	3x3	2	6
Inverted Residual Block x3	14x14x64	14x14x96	3x3	1	6
Inverted Residual Block x3	14x14x96	7x7x160	3x3	2	6
Inverted Residual Block x1	7x7x160	7x7x320	3x3	1	6
Final Conv	7x7x320	7x7x1280	1x1	1	-
Global Avg Pooling	7x7x1280	1x1x1280	-	-	-
Fully Connected	1x1x1280	1x1x1000	-	-	-

Model Customization for Food-101

In the beginning, I excluded the top layer of the MobileNetV2 for customization of the food-10 dataset. I also added some customized layers including a **GlobalAveragePooling2D** layer to reduce the spatial dimensions, a **Dropout layer** (rate = 0.5) to prevent overfitting, and a **Dense output layer** with a **softmax activation** function to classify the 101 food categories.

Configuration

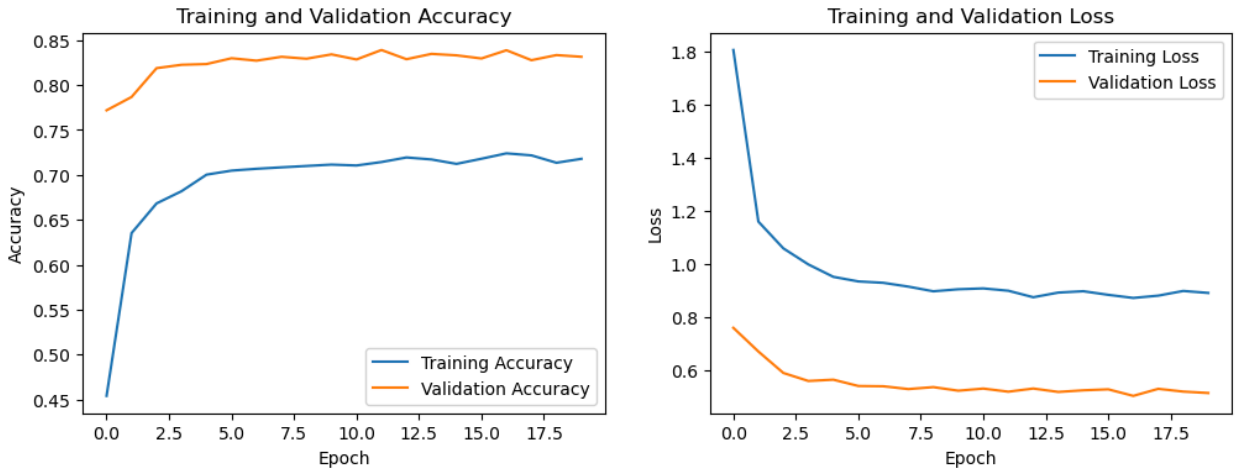
The model use **Adam optimizer** (learning rate = 0.001) to optimize training performance. The loss function is **categorical crossentropy** since it's multi-class classification tasks, and **Accuracy** is used as the evaluation metric.

Training and Testing

Using **batch size 64** considering the limitation of my computer and efficiency in both time management and model performance. The model is trained for up to **30 epochs**, with **early stopping** based on **validation loss** to prevent overfitting. The **EarlyStopping callback** monitors the validation loss, and training halts if the loss does not improve for 3 consecutive epochs. The best weights are restored.

The testing process that uses the trained model evaluates its performance on unseen validation data after each epoch. Accuracy and loss metrics are computed for both training and validation datasets to represent progress and generalization.

Model Performance



Training and Validation Accuracy:

The training accuracy begins around 45% and steadily increases, reaching approximately 70% by the end of the training process. This indicates that the model is learning from the training data effectively. The validation accuracy starts higher than the training accuracy (around 77.5%) and remains relatively stable throughout training, increasing into the range between 80% and 85%. The gap between training and validation accuracy indicates that the model generalizes well to the validation data and is not overfitting significantly.

Training and Validation Loss:

Both training and validation loss decreases sharply in the early epochs and gradually decreases after 1 epoch suggests that the model is learning effectively.

The difference between the accuracy and loss in the validation and training could be due to data augmentation that I used in the training dataset which makes the image hard to be learned compared to validation dataset.

App Design and Structure

App Structure

The App is designed using Swift since it's really good for ios applications and easy to integrate transformed Machine Learning Model.

The app need to achieve two main components:

1. **ContentView**

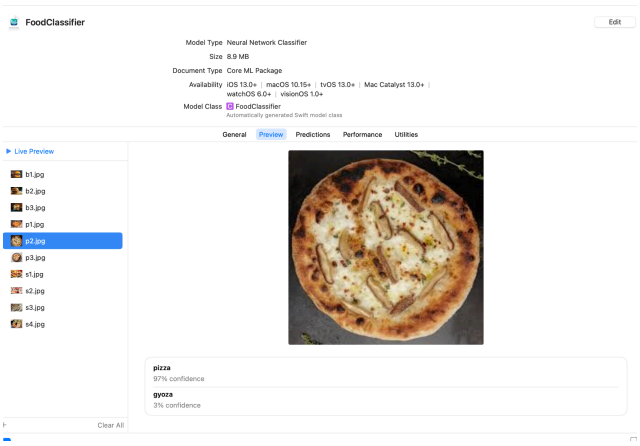
   The component can be treated as Frontend, which is a SwiftUI view that acts as the primary interface for users. Allows users to select an image from the camera or photo library. Displays the classification result with a confidence percentage from the message of ImagePicker.

2. **ImagePicker**

   The component can be treated as Backend, which is a UIViewControllerRepresentable wrapper for integrating UIKit's UIImagePickerController in SwiftUI. Handles image selection, image preprocessing, using ml model to predict the processed image, receive the result and transform result into readable format including show the according food labels and calculating confidence levels. Then it will send the result into the displaying component in the CounterView.

Model Integration

The model was trained on a computer using TensorFlow and Keras. To integrate it into an iOS application, I used a specialized library called **coremltools** to convert the model into a format compatible with iOS. Setting up the environment for **coremltools** required following specific instructions provided on their website. Additionally, the latest versions of TensorFlow are not fully tested with **coremltools**, which can lead to unexpected errors. As a result, the environment setup process took a considerable amount of time. After transforming from **FoodClassifier.h5** into **FoodClassifier.mlpackage**, the integrating process will be straightforward, I can just drag the mlpackge file under the directory of my App folder, then import the ml model from ImagePicker directly. In addition, before integrating into app direct, xcode also provide a preview function of the ml model which allow to do some simple tests, which is really useful for checking the ml before implementing.



App Testing

After completing all the steps, I tested the app on the iPhone simulator to ensure all components functioned smoothly. Initially, I noticed that the model's predictions on the phone were inaccurate and sometimes nonsensical compared to its performance on the computer. To identify the issue, I imported both the transformed model (used in the app) and the original model back into my computer and ran predictions on the same dataset for comparison. The results were nearly identical, indicating that the issue was not with the model itself.

```
1/1 [==============================] - 0s 28ms/step
Image: s4.jpg
  TensorFlow - Predicted Class: pizza (50.39% confidence)
  Core ML - Predicted Class: tacos (43.85% confidence)
--------------------------------------------------
1/1 [==============================] - 0s 24ms/step
Image: s1.jpg
  TensorFlow - Predicted Class: sushi (99.83% confidence)
  Core ML - Predicted Class: sushi (99.90% confidence)
--------------------------------------------------
1/1 [==============================] - 0s 24ms/step
Image: s3.jpg
  TensorFlow - Predicted Class: sushi (68.12% confidence)
  Core ML - Predicted Class: sushi (88.38% confidence)
--------------------------------------------------
1/1 [==============================] - 0s 23ms/step
Image: s2.jpg
  TensorFlow - Predicted Class: sushi (93.86% confidence)
  Core ML - Predicted Class: sushi (82.28% confidence)
--------------------------------------------------
1/1 [==============================] - 0s 25ms/step
Image: p3.jpg
  TensorFlow - Predicted Class: pizza (96.26% confidence)
  Core ML - Predicted Class: pizza (99.12% confidence)
```

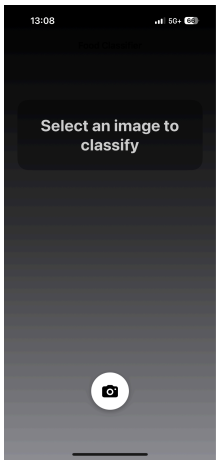
Realizing the problem might lie in the data preprocessing within the app, I reviewed and rewrote the preprocessing logic in the ImagePicker component. After implementing these changes, the app's predictions significantly improved, resulting in much more accurate classifications.

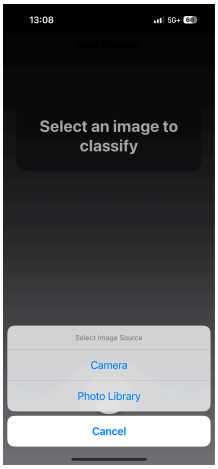
Error Summary

The most frequent problem that I faced in the app implementation was model transformation, data preprocessing function in the model,

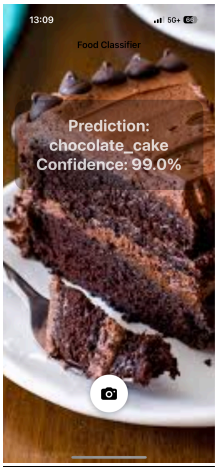
Demos

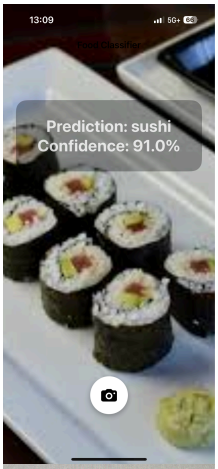
Photo Library =>

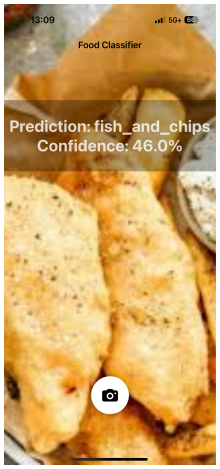


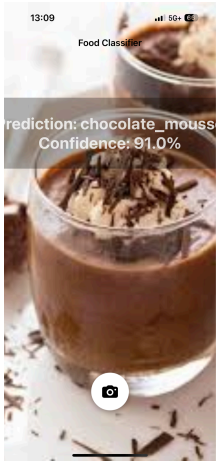


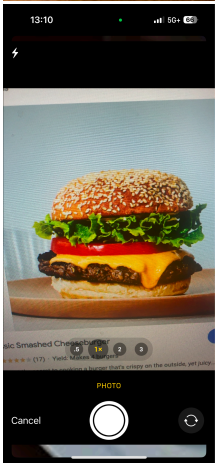
Camera =>

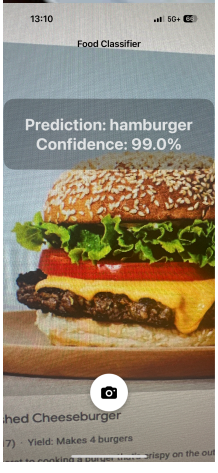












Work Cite:

Bossard, L., Guillaumin, M., & Van Gool, L. (2014). Food-101–mining discriminative components with random forests. In Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part VI 13 (pp. 446-461). Springer International Publishing.https://link.springer.com/chapter/10.1007/978-3-319-10599-4_29

Food-101 Dataset: https://www.kaggle.com/datasets/dansbecker/food-101

Keras Model Information: https://keras.io/api/applications/

CoreMLTools: https://apple.github.io/coremltools/docs-guides/