EC535 Final Project Technical Report

FoodVision

Gordon Zhu and Kevin Liu

Link to Git repo: https://github.com/gordonnzhuu/ec535-final-project

Link to YouTube video: https://www.youtube.com/watch?v=MiiAG8sFhY0

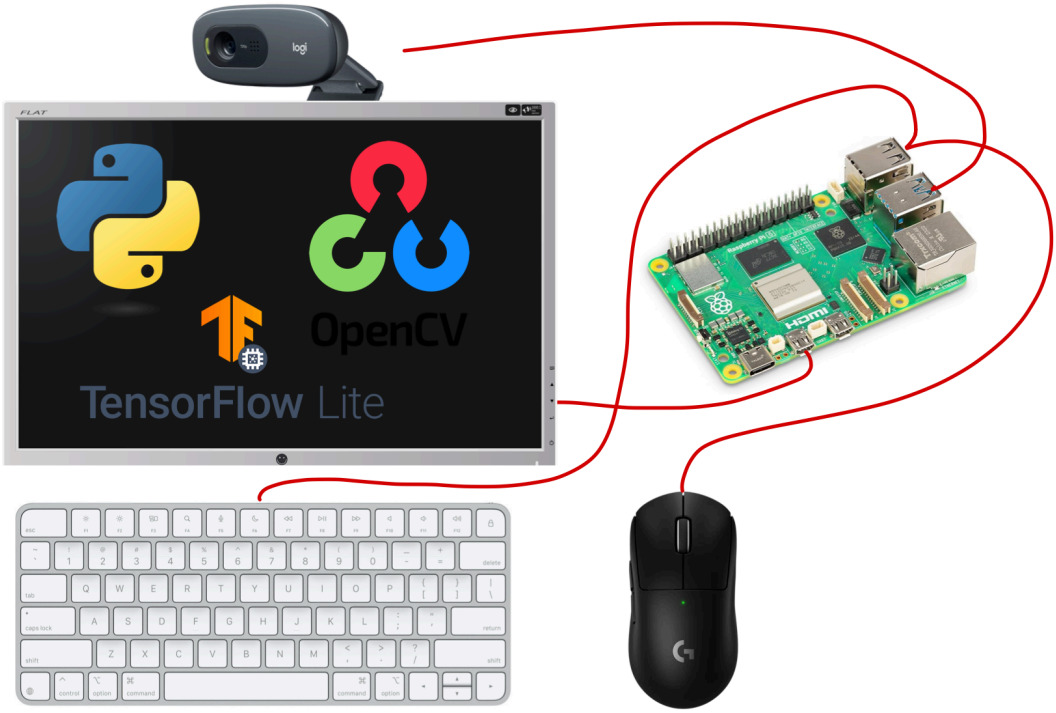

Figure 1: Overall System Design

Abstract

This project is a real-time food recognition system built on a Raspberry Pi 5 using a webcam and a lightweight machine learning model. Using OpenCV for image capture and TensorFlow Lite for real-time prediction, the system can identify different food items from a live video feed. Future plans include adding nutrition facts, using hardware acceleration, improving performance with multithreading, and possibly moving inferencing to the cloud for more advanced capabilities.

Introduction

There is a massive growing interest in health tracking, smart kitchens and artificial intelligence. By combining real-time image processing with efficient on-device AI, it demonstrates how useful and practical these innovations can be. As interest in personal wellness and smart home technologies grows, the ability to identify food in real-time opens up possibilities for applications like diet tracking, automated meal logging, and interactive kitchen assistants. This project aims to bring food classification to the edge by running machine learning models on low cost devices

to let the average consumer enjoy these advancements. Using a webcam to capture live video and TensorFlow Lite to run a pre-trained MobileNet model, the system identifies food items in real-time, making it fast, self-contained, and accessible. Beyond its technical capabilities, the project also celebrates cultural diversity, highlighting the universal role food plays in bringing people together across cultures. The motivation behind this work is to demonstrate that real-time AI applications are now possible on small, affordable hardware.

Methods

Computer Vision

OpenCV is used to access and process the frames from the Logitech USB webcam. Each frame captured is resized to 224 x 224 pixels to match the input requirements of the model and converted from BGR to RGB color format. This ensures compatibility with the TensorFlow Lite model and helps reduce the computational overhead.

```
    # Preprocess frame
    img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (w, h))
    data = np.expand_dims(img_resized.astype(inp_det['dtype']), axis=0)
```

For the food inferences, our device uses the TensorFlow Lite, which runs small and fast machine learning models on the Raspberry Pi. It was ideal for real-time predictions on low power embedded devices without needing a powerful computer. Another reason is that it supports quantized models. The model used is a quantized MobileNet V1.0, downloaded using the KaggleHub library, and trained specifically for food classification. This model also allowed me to train on my own dataset, which was great for optimization.

Quantization

Quantization plays a critical role in reducing model size and computational demands. It is the process of converting continuous values into a smaller set of discrete values. In machine learning models, the values that the model uses to make decisions are normally stored as 32-bit floating-point numbers, which are precise but take up a lot of space and are slow to process. However, we can use quantization to shrink those values down. For our model, we quantized unsigned 8-bit integers. Although, this decreases our accuracy; the pros outweigh the cons. We made the decision to give up some degrees of accuracy for less memory and reduce thermal throttling. It also saves energy since smaller calculations are faster and require less power. This drastically improves performance and energy efficiency, making it suitable for real-time edge deployment.

4D Tensor

The model expects the input/image to be in a specific format called a 4D tensor or a multidimensional array. The shape (1, 224, 224, 3) means:

1 → Only one image is being passed in.
224, 224 → The width and height of the image.
3 → The number of color channels (Red, Green, Blue).

The webcam continuously takes 640x480 pixel frames and converts them into a 224x224 pixel matrix. This RGB matrix is then fed into our model in which it does the inference based on its pre-trained dataset of food. The inference returns an index for which is mapped to a csv containing the corresponding food.

Frame Skipping

To balance performance and efficiency, we adopted frame skipping that deliberately skips inferencing every third frame. Using a simple modulus, we were able to guarantee stable CPU temperatures while running our food classification model.

```
frame_count = 0
frame_interval = 3

while True:
    ret, frame = cap.read()
    if not ret:
        print("Failed to grab frame.")
        break

    frame_count += 1
    if frame_count % frame_interval != 0:
        continue
```

Results

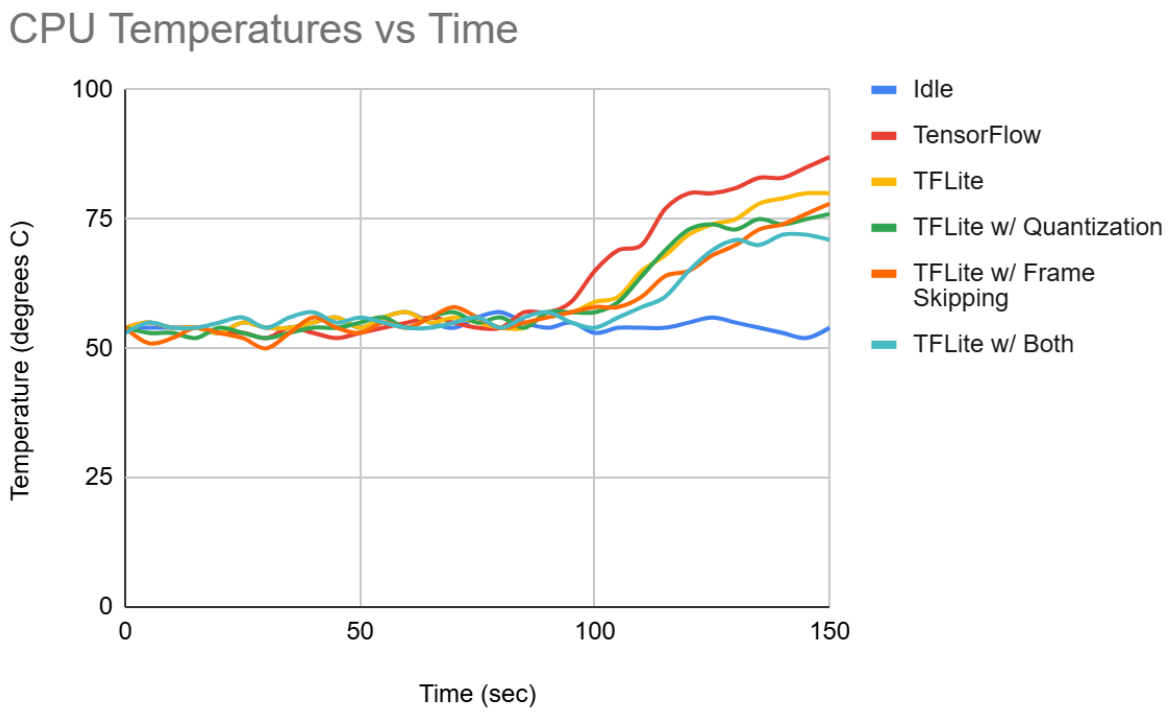

Figure 2: Thermal Performance with Different Optimizations

The plot shows that running the full-sized TensorFlow graph (red) is a furnace, pushing the CPU from a resting ~52 °C to nearly 90 °C by the 2-minute mark. Dropping down to TensorFlow Lite (yellow) shaves a few degrees, but the real gains come from lightweight inference tricks. Quantizing the TFLite model (green) knocks another ~5 °C off the peak, while frame-skipping on the non-quantized model (orange) yields a similar benefit by simply doing less work. When the two techniques are combined (teal), the curve tops out around 72 °C—roughly a 17 °C margin over stock TensorFlow and only ~20 °C above the idle baseline (blue), all while sustaining usable inference throughput. In short, every optimization step visibly throttles heat generation, and quantisation + frame-skipping together give the best thermal headroom for prolonged real-time use.

Limitations and Future Work

While the project successfully demonstrates real-time food classification on a Raspberry Pi, there were several limitations. When foods looked similar to other foods, the model sometimes would not be able to tell the difference. In addition, the inference speed was limited by the Raspberry Pi's CPU which led to delays in the classification. If we were to continue working on this project, there are several features that we would try to implement:

Displaying nutrition

One major feature planned is the ability to display nutrition facts for the recognized food item in real time. Once the system identifies a food, it could pull preloaded nutrition data from a local or online database and overlay relevant facts like protein, calories and sodium on the screen. This feature would make the project more useful for health-conscious users, fitness tracking, or diet planning. It would also enhance the educational value of the system, especially for kids or those learning about food and nutrition. The reason this feature was omitted was because the raspberry pi was already heating up from just the food recognition and inference alone. We thought adding this feature might burden the Raspberry Pi too much or fry it. Since we wanted to ensure the Pi remains working for the presentation we chose to scrap the idea. Once we optimize the rest of the food classification software, we can try implementing this feature.

Edge TPU or the Pi's Integrated GPU

Currently, all model inferences run on the Raspberry Pi's CPU, which is not optimized for machine learning tasks and limits the efficiency. Using an Edge TPU or the Raspberry Pi's integrated GPU could dramatically improve speed and efficiency. These dedicated hardware components are designed for running AI models and can process inferences much faster than a CPU while using less power. Offloading inferencing to the TPU or GPU would increase responsiveness, allow the system to run inference on every frame instead of skipping, and allow for running more complex or higher-accuracy models without sacrificing performance.

Heatsink

During extended use, the Raspberry Pi's processor can emit a lot of heat which can be dangerous. This thermal throttling slows down the system and reduces inference speed. Installing a heatsink or a small fan would help manage the temperature more effectively, ensuring consistent performance even during long sessions. It's a low-cost hardware upgrade but important for real-world usability, especially if the system is to be deployed in enclosed spaces or as a long-running smart kitchen assistant. Unfortunately, we did not acquire any necessary materials for a possible heat sink so it ultimately was not implemented.

Cloud-Based Inference

While running inference locally is great for speed and offline use, it limits the size and complexity of the models that can be used. In the future, the system could send images to a cloud server for inference using a larger, more accurate model hosted on platforms like AWS, GCP, or Azure. This would allow support for more food categories, better handling of edge cases, and improved classification accuracy. Cloud inference would also enable automatic updates and remote model improvements without requiring hardware changes. While it introduces a dependency on internet access, it could be offered as a toggleable feature for users who want enhanced performance.

Conclusion

This project demonstrated that real-time food classification is achievable on low-cost edge devices like the Raspberry Pi using a quantized machine learning model and efficient image processing with OpenCV. We successfully integrated hardware and software components to build a system that can identify food items live through a webcam. Throughout the process, we learned the importance of model optimization, the trade-offs between accuracy and performance, and how preprocessing and system design affect real-time responsiveness. We also gained hands-on experience with TensorFlow Lite, system integration, and the practical challenges of deploying AI in real-world settings.

What technical skills have been gained from this project?

Working with OpenCV in Python, machine learning, real-time image processing, deep learning employment, and model optimization

References

[1] https://www.kaggle.com/models/google/aiy

[2] https://www.ultralytics.com/blog/what-is-model-optimization-a-quick-guide

[3] https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[4] https://ai.google.dev/edge/litert