VeggieVision Team Contributions

Project Overview:

VeggieVision is an image classification pipeline trained on a custom dataset of both bagged and unbagged produce. Deployed on Walmart self-checkout kiosks, it automatically recognizes fruits and vegetables in real time—so customers don't have to type in produce item names, dramatically reducing wait times. By incorporating on-device inference, VeggieVision delivers high accuracy and fast, reliable performance under a wide range of in-store conditions.

Team Members:

- Jordi Castro
- Dylan
- Samuel
- David (Leader)

1. Data Collection & Annotation

For the first round of data collection, each member was assigned two vegetables/fruits to buy at Walmart for the purpose of taking photos. For each produce item around 100 images were taken in different lighting conditions and angles (in store, at home, etc.) with variable difficulty. Each jpg file was renamed using a Python script and the following naming convention: [PRODUCE_NAME]_[DESC1]_[OPT. DESC2]_[EZ | MD | HD].jpg. Once renamed, these files were placed into the original repository in nested folders corresponding to the produce name and difficulty.

Next, the continuing data collection was split between the classification data (David) and the detection data (Samuel). The classification data was composed of around 1800 images; the detection dataset was around 782 manually annotated images. The images were captured on iPhones/Android devices.

2. Classification Model Development

Three EfficientNet models (B0, B1, B2) were trained on the classification dataset, with the final choice of model being the smallest, EfficientNetB0. All this training was completed by David through the university's HPC cluster.

3. Detection Model Development

Samuel focused on training the YOLOv8 model, which utilized a base model pre-trained on the COCO dataset provided by the Ultralytics library. The model was first evaluated on a

baseline testing image set from the week 1 collected data. Afterwards, the model was trained and evaluated on a kaggle dataset for fruits and veggies by rewa77: https://www.kaggle.com/datasets/rewa77/dataset-for-vegatables/data
And then trained and evaluated on another kaggle dataset for bagged fruits and vegetables by kvnpatel: https://www.kaggle.com/datasets/kvnpatel/fruits-vegetable-detection-for-yolov4.

Samuel then explored AutoDistilll to see if we could utilize GroundedSAM to zero-shot detect the fruits and vegetables in our collected data. However, this was unfruitful since the data annotations generated by groundedSAM were too messy and many duplicate bounding boxes were created (8+ boxes for a single item), which would require more manual labor than just manually labeling the data.

Next, Samuel manually labeled the collected data with bounding boxes for training detection models. We settled on Roboflow to provide the data labeling interface for drawing the bounding boxes to manually label our data, as well as to augment our data to generate more training images. The small week 1 dataset consisting of 199 images was manually labeled by Samuel, then he trained the first iteration of the yolov8 model (finetunev1) on that dataset.

After getting our finetunev1 model, Samuel explored utilizing this fine-tuned model to automatically label data he further collected (~582 images). This helped quite a bit as a lot of the images were able to be annotated by our finetunev1 model, so all that needed to be done was to clean up the boxes and add annotations where the model missed some detections.

The final version of our collected dataset (version 2) comprised 781 images (~2k after augmentation) and our yolov8 finetunev2 model was trained on this new dataset. Afterwards, Samuel took a video at the Walmart self checkout to evaluate the detection model's performance (video provided in the detection model code github).

Most of the code was done in google colab notebooks, but Samuel ported the code to python for the final version of our detection model for both training and inference. The colab notebooks are provided in the notebooks code github and show the progress from baseline all the way to finetunev2.

Jordi researched the real-time, light weight DEtection TRansformer (DETR): RT_DETR_v2. He spent a couple days reading up on the documentation on HuggingFace, making sure the model was a viable solution given our strict mobile demo constraints. He found that while DETRs in practice are dense, the real time DETR is lightweight and could be a fitting model. Thus, Jordi began testing a notebook and seeing if he could tweak it and train the Kavan Patel Fruits & Vegetables Dataset that other group members were using as a base performance metric. While attempting to train the RT_DETR_v2 model, Jordi ran into issues regarding the dataset format. He had to transcribe the dataset to proper YOLOv5 format, because the inference and annotation files were in the same directory. Extracting the images from the directory using a script, he was able to achieve the correct YOLOv5 format, then use a Google Colab Notebook to extract the annotations in json format, which is the COCO standard dataset format for DETR models. After extracting instances_train.json and instances_val.json, Jordi was able to organize the files into the annotations, train, and val directories.

Now that the COCO dataset was zipped and uploaded to Google Drive, Jordi was able to proceed with the training of the RT_DETR_v2 model. He used a small backbone checkpoint,

“PekingU/rtdetr_v2_r18vd”, provided by HuggingFace. He created a custom data class to be able to use his local dataset (unzipped from Google Drive) with HuggingFace’s Transformer Classes. After following various tutorials and performing exhaustive debugging, Jordi was unable to successfully train the model using the derived COCO Kavan Patel dataset. The Google Colab Notebook is here. The reason trainer.train() was resulting in a RuntimeError is likely due to an incorrect or corrupt file when translating the dataset from YOLOv5 to COCO format. The model likely saw a mismatch in an annotation of an image that did not exist and crashed. Upon hearing the success of Samuel and David’s models, Jordi adapted and transitioned to designing the demo application.

4. Mobile Design

The mobile design was done in large by Jordi Castro. To begin, the rough outline of the mock-up was sketched on a piece of paper. Next, a Figma was created to design the views, following the vision of the sketch. The prototype was shared with the team. After receiving approval from each team member, Jordi dove into Swift and iOS app development research. Leveraging his experience in mobile app development using AndroidStudio, Kotlin & Jetpack Compose, and web app development using React and Next.js, Jordi quickly found similarities in components and project structure as he developed the UI of the app. The application was built using SwiftUI and model loading was done using the ONNX Runtime library with UIKit. The EfficientNetB0.swift file, which loaded in the model from the .ort file, was written by David in UIKit. Jordi had to bridge the gap between SwiftUI and UIKit to get the model's predictions displayed and stored in each view. For more information on specific views and state, view the “Read-me” here.

5. Presentation

The development of the VeggieVision presentation for Walmart's AI Workshop involved a methodical process that balanced technical accuracy with effective communication. The process began with gathering critical information about the technical models through collaboration with Samuel, Jordi, and David. These sessions focused on understanding the key differences between EfficientNetB0 and YOLOv8 models, particularly considering factors most relevant to decision-makers, such as processing speed, accuracy, and implementation complexity.

After collecting detailed technical information, the next step involved distilling complex machine learning concepts into accessible visual comparisons. This included creating side-by-side model comparisons highlighting differences in capabilities and limitations, and developing visual metaphors and simplified diagrams that communicated technical concepts effectively. The presentation incorporated Walmart's signature blue and yellow color palette to maintain brand identity while using clean, minimalist design principles with ample white space to present complex technical information without visual overload.

The "Challenges Faced" section was crafted to honestly reflect technical hurdles while emphasizing problem-solving approaches rather than setbacks. This approach resulted in a

presentation that successfully maintained technical integrity while being accessible to AI specialists and business stakeholders, while visually aligning with Walmart's corporate identity.

6. Financial Analysis

Dylan’s detailed financial analysis demonstrates that implementing a solution to reduce manual produce checkout time by 37.1 seconds per transaction would generate substantial returns.

With a current transaction time of 180 seconds, this improvement reduces checkout time to 142.9 seconds, increasing throughput capacity by 25.96% (from 20 to 25.19 transactions per hour per lane). In a typical Walmart Supercenter processing 4,000 transactions daily (1.46 million annually), this efficiency gain translates to 379,016 additional potential transactions per store per year.

At an average transaction value of $45, each store could generate $17,055.720 in additional revenue potential. When scaled across Walmart's 4,600 US locations and adjusted for realistic implementation factors (35% self-checkout usage, 60% transactions with produce, 40% revenue conversion rate), this initiative would deliver approximately $6.59 million in additional annual revenue. These calculations are achieved through addressing a single operational bottleneck that affects virtually every retail transaction involving produce items.

(Need Internal Data to Validate and Run Further Calculations)