UC Berkeley MIDS

W251 - Deep Learning in the Cloud and at the Edge

Final Project Report: *SmarterChef - An AI Based Recipe Generator*

Team: Anil Tipirneni, Gabriela May Lagunes, Jesse Miller, Tal Segal & Varun Tanna.

Submitted on August 2nd, 2022

Introduction 2

Previous Work and SOTA 2

Project Overview 3

Project Steps 3

- Image Annotation 3
  - Simple Mode: Public Datasets 3
  - Custom Mode: Original Dataset 4
  - Mixed Mode: Public + Original Datasets 5
- Model Architecture and Training 6
- Inference 8
- Recipe builder with GUI Interface 9

Challenges 9

Future Work 10

Repo Information 10

References 11

Introduction

According to the United Nations' Food and Agriculture Organisation (FAO) and US Department of Agriculture, the world wastes about 1.4 billion tons of food every year. Here in the United States, we discard more food than any other country in the world: nearly 40 million tons — 80 billion pounds — every year. That's estimated to be 30-40 percent of the entire US food supply and equates to 219 pounds of waste per person [1]. The majority of this food waste comes from homes, restaurants and grocery stores. Our goal for the project is to provide a solution that supports households to reduce food waste.

Previous Work and SOTA

The challenge of food recognition from images has been addressed with several objectives in mind. For instance, identifying food and ingredients from mobile phone photos can be an important tool in nutrient tracking for medical research and treatments [2][3][4], or implemented in self-checkout systems in restaurants, canteens and grocery shops [5]. There are plenty of datasets related to food that are publicly available, and the variety of the data quickly becomes a challenge. Datasets usually have a large number of classes, and the origin of the images is diverse. For example, for nutrition research purposes the datasets like UEC-FOOD10 [6] have become popular, but they are made from foods that are mostly consumed in Japan, so that introduces a bias from the beginning. Food as a subject is itself very complex. For instance, there was a project in which researchers used deep learning to successfully classify 73 different classes of bread alone [7].

From our literature review, we believe convolutional neural networks are the most commonly used algorithms for food recognition. Resnet50, Mask RCNN [2], deep convolutional networks [4], AlexNet and GooLeNet [8] are some examples of commonly used architectures for food recognition. Yolov5 is formed by CSPDarknet and PANet as shown in the figure below, and it has been successfully used for the development of complex dishes classifications and implemented in smart cantenees, so we decided to take advantage of the learnings of the class and use Yolov5 for our project [9].

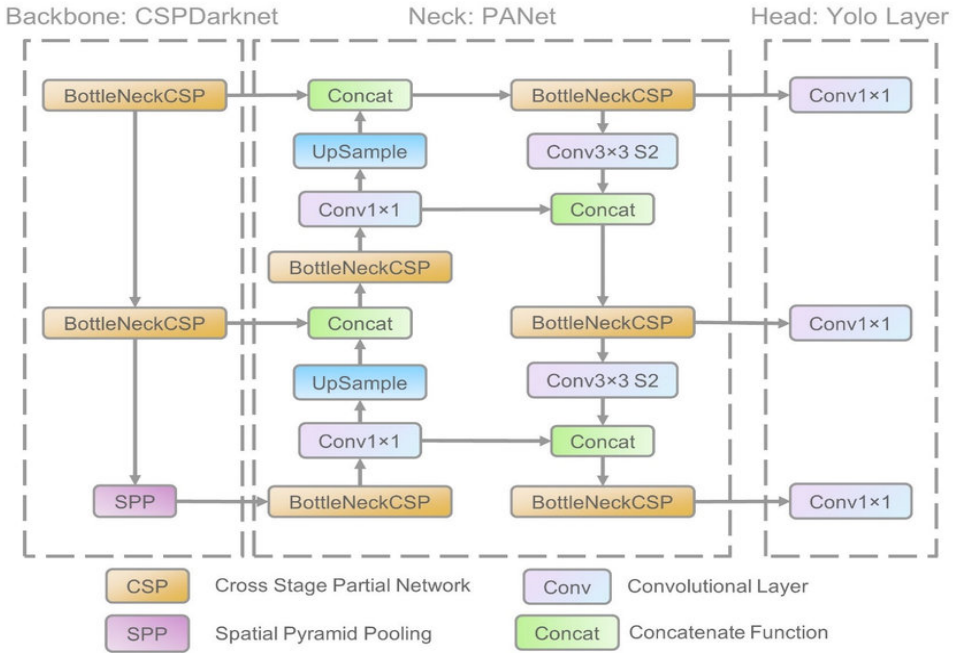


Figure 1: Yolov5 architecture diagram [10]

Project Overview

The objective of this project is to develop an end-to-end application, in the cloud and at the edge, that is able to recognise individual ingredients and get different recipes for the end user. We trained an object detector in the cloud, starting from the weights of the Yolov5m model [11]. We used Roboflow to generate our own dataset with over 10,000 annotated images from video frames pre-processed with augmentation techniques. The final model identifies 32 ingredients and achieves a mean Average Precision (mAP) that is competitive with SOTA architectures. We saw that one of the main challenges of developing a usable food identifier is the large variety of ingredients and dishes in the modern diet, which translates into the need of an object detector with many classes.

The main components in the project are:

- **Web Application** - Our user interface was developed on Fast API, which allows us to expose various HTML elements using python scripts. Upon launch, our main container runs a web service on a designated port using Uvicorn.
- **Camera Input** - Upon clicking the scan button on the web application, we are able to start a camera stream within the container of the same edge device, which captures frames and checks for recognized food items within the frames.
- **Model -** Using the downloaded model from the cloud we were able to load it into our edge container and have it taken in image inputs from the camera. Upon detection of an item, we were able to show the frame and the confidence level at which the item was detected.
- **Database -** Once a food item is scanned by the camera and recognized by the model, we create a SQL database using the sqlite storage engine. Our database.py script initialises this Sqlite database using the SQLAlchemy library as an ORM to allow easier access within our other python scripts.
- **Recipe Generator -** We are using a public api endpoint from our main.py script (https://api.spoonacular.com/recipes/findByIngredients) to retrieve a list of recipes that would be acceptable given our search parameters which include the aforementioned ingredient list.

Project Steps

1. Image Annotation

In order to generate the final dataset that we used for the project, which can be found here, we went through 3 iterations of model training using 1) public datasets 2) an original dataset made by us and 3) a combination of the two.

1) Simple Mode: Public Datasets

As the name suggests, in this first iteration we used pre-made, public datasets. There are multiple datasets available such as ai-cook, food320 and food232 in Kaggle and other sources. We decided to use the data from ai-cook [12] because of the results they reported and the variety of the food categories they used. The dataset is available on Roboflow and is composed of 516 annotated images of various ingredients inside a fridge. We used an augmented version of the dataset with 3050 images. We ran the custom trained YOLO5s model interference on the jetson camera to ensure that we can detect the food items. We realised that there were three problems – the detection confidence

is lower than expected (banana in the image below), misdetection (86% confidence that they were apples, while they truly are limes), and phantom detection (the model was constantly seeing spinach in white space). So, we moved on to custom mode as explained in the next section.

Figure 2: Ai-cook dataset annotation example (left). Example of inference of the first model (right).

2) Custom Mode: Original Dataset

In this mode, we created videos for different food items on one of our cameras. Multiple videos were created with different white/dark backgrounds, and capturing different angles and positions of the items. These videos were then converted to images using ffmpeg with a 2 frames per sec setting. These images were uploaded and annotated on Roboflow. We ran the custom trained YOLO5 model interference on the jetson camera to ensure that the can detect the food items. The inference yielded much better results than the simple mode. With this new learning, we moved on to mixed mode. See the image below for examples of annotation and inference.

Figure 3: Custom dataset annotation examples

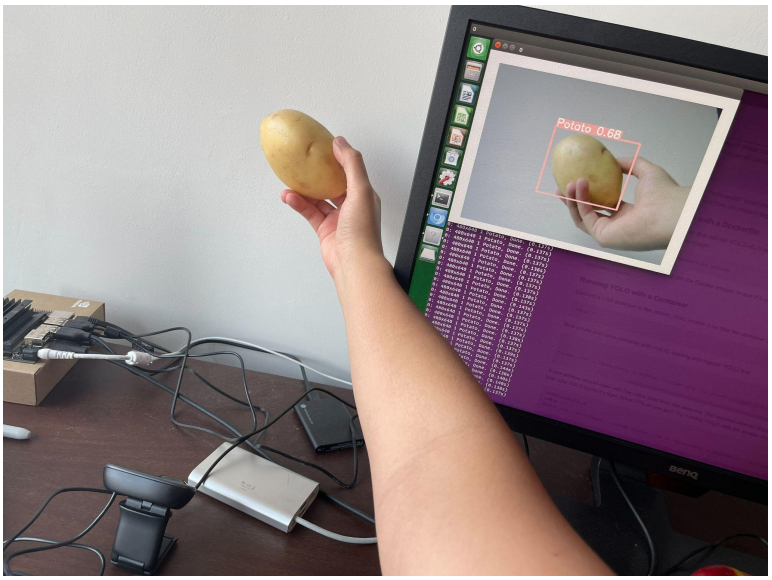


Figure 4: Example of inference from video stream on model trained with custom data.

3) Mixed Mode: Public + Original Datasets

At the end we combined previously annotated images from ai-cook and the custom images we created. We wanted to try to get our model to learn from a dataset that had yielded good results in previous projects, but we wanted to add new food categories and make sure our model would be able to generalise well on a video stream. We annotated 4,193 images, added the 516 images from ai-cook, implemented rotation and cutout, change of exposure, and added blur and noise. At the end we created a dataset with ~11K images. The training/validation/test splits and the augmentation techniques are shown in the figure below. Due to the limitations of the free version of Roboflow, we were only able to generate 3 images out of each original image.

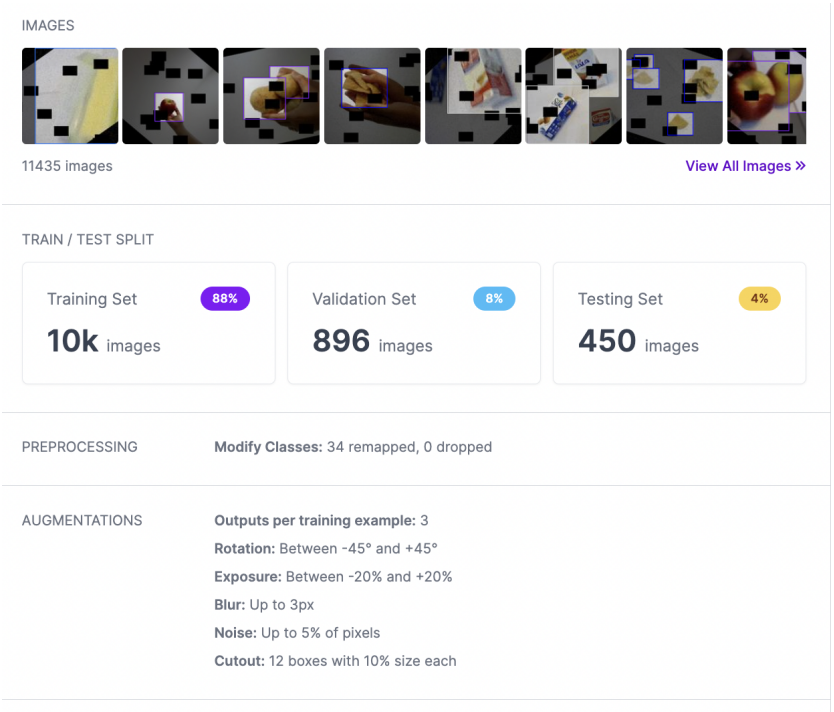


Figure 5: Training/Validation/testing split for mixed mode

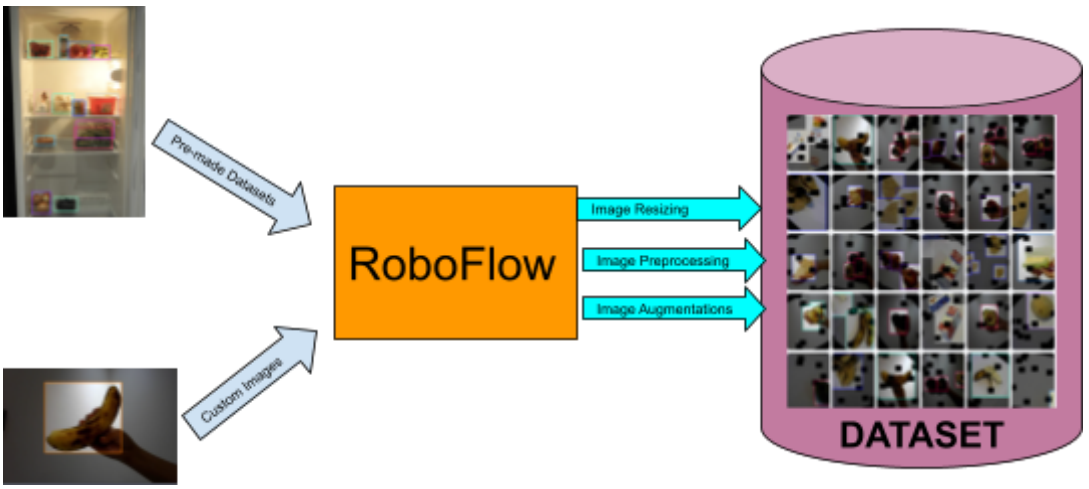


Figure 6: Schema of generation of final dataset

From here we were able to extract them from the Roboflow api, and load them into our training container for model training.

2. Model Architecture and Training

From our previous work with training data on the edge and in the cloud, we decided to train our model on the cloud to leverage computing power available there. We provisioned a g4dn.2xlarge Elastic Compute Instance on AWS and ran a docker container on it which pulled from a version of the pytorch image. From inside this pytorch container, we were able to train our model on the dataset that was generated using roboflow, as described above. After our training script ran and the validation results were satisfactory, we extracted the model weights from the container to deploy in our user interface. As mentioned in the SOTA section, we decided to use Yolov5 for our project. As the figure below shows, Yolov5 outperforms EfficientDet in object detection in the CoCo AP validation and it gives different options of robustness based on the same architecture shown in Figure 1 (*Previous Work and SOTA* section).

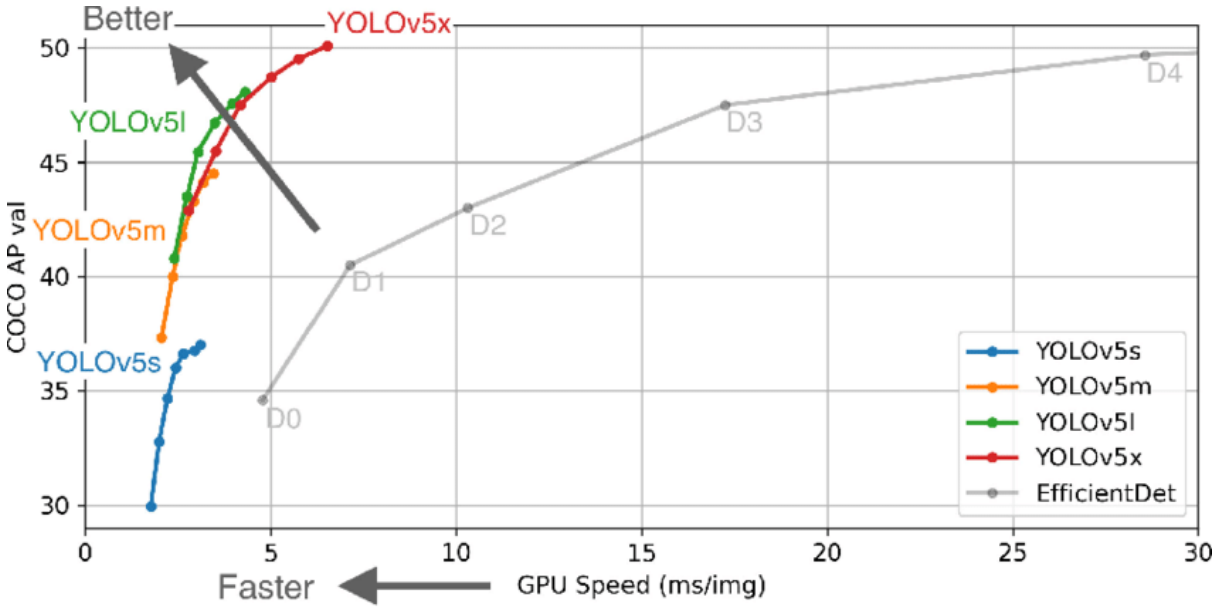
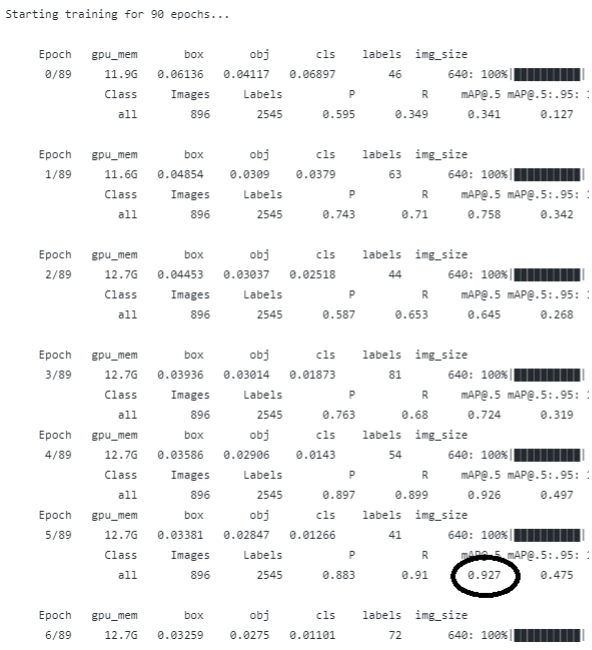


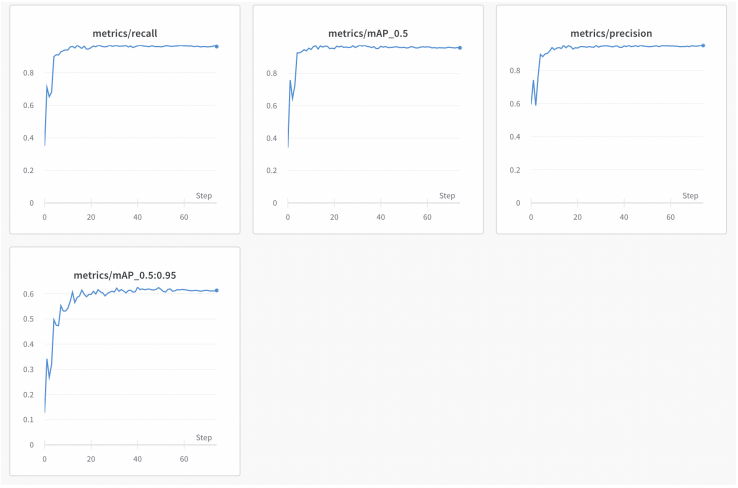
Figure 7: Yolov5 vs EfficientDet on COCO AP validation. Taken from [11]

As shown in the diagram above, YOLOv5 has multiple options available. To validate that we are able to get the pipelines up and running, we started with YOLOv5s for the first inference. We trained a couple of models starting from YOLOv5s, first training with public datasets and then with our own dataset. The validation results of these models gave a mAP value between 0.5-0.6 per class, but they were not capable of identifying anything outside of their own validation datasets. For instance, if inference ran over a photograph with a different format than the dataset used for training, or used over a video stream, then no food item was identified. We then started using YOLOv5m and we saw a significant improvement in training and validation results. Below we show the logs, validation results and monitoring of train and val metrics (from WandB) for our final model. Our mAP 0.5 was above 90% before the 10th epoch of training, and mAP@.5:.95 surpassed 60% at the end of the training. The full logs can be found in the project repo.



Class	Images	Labels	P	R	mAP@.5	mAP@
all	896	2545	0.949	0.967	0.969	0.625
apple	896	437	0.984	0.974	0.99	0.661
avocado	896	25	1	0.971	0.995	0.806
banana	896	273	0.981	0.965	0.981	0.716
beef	896	24	0.771	0.875	0.747	0.186
blueberries	896	33	0.984	0.97	0.966	0.593
bread	896	41	0.951	1	0.985	0.647
butter	896	142	0.977	0.902	0.97	0.54
carrot	896	37	0.891	0.946	0.948	0.582
cheese	896	49	0.999	0.98	0.994	0.718
chicken	896	27	0.984	1	0.995	0.611
chicken_breast	896	35	0.768	0.886	0.809	0.313
chocolate	896	29	0.956	0.931	0.959	0.579
corn	896	48	0.99	0.938	0.993	0.625
eggs	896	60	0.993	1	0.995	0.625
flour	896	58	0.999	0.948	0.973	0.645
goat_cheese	896	11	0.959	1	0.995	0.681
green_beans	896	47	0.918	0.957	0.982	0.625
ground_beef	896	22	0.813	0.99	0.953	0.529
ham	896	7	0.958	1	0.995	0.534
heavy_cream	896	44	0.988	1	0.995	0.673
lime	896	28	0.944	1	0.995	0.749
milk	896	138	0.903	0.935	0.969	0.662
mushrooms	896	56	0.992	1	0.995	0.691
onion	896	42	0.949	1	0.995	0.741
potato	896	273	0.964	0.971	0.984	0.643
shrimp	896	42	1	0.988	0.995	0.616
spinach	896	38	0.923	0.895	0.957	0.658
strawberries	896	51	0.992	0.98	0.985	0.688
sugar	896	78	0.944	1	0.987	0.701
sweet_potato	896	32	0.957	1	0.992	0.725
tomato	896	59	0.995	1	0.995	0.612
tortilla_chip	896	259	0.941	0.934	0.932	0.626

Speed: 0.3ms pre-process, 9.9ms inference, 1.3ms NMS per image at shape (32, 3, 640, 640)
Results saved to [1mruns/val/exp11[0m



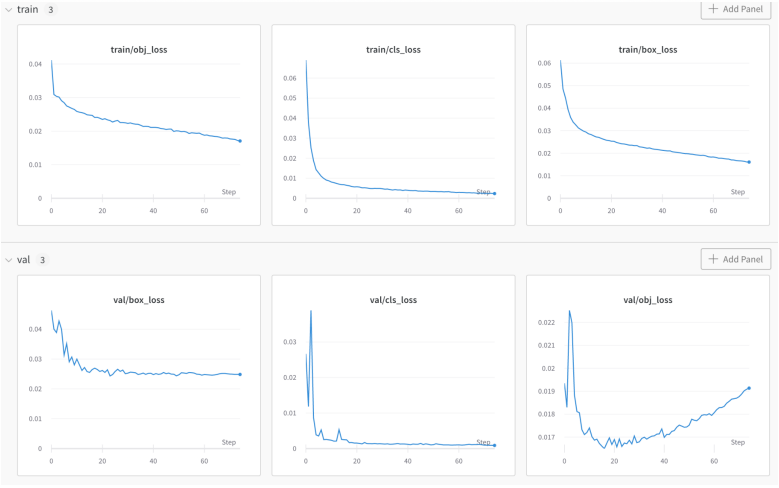


Figure 8: Train and validation logs, metrics and results with our final dataset and Yolov5m architecture.

We saw that the combination of different kinds of images with the augmentations described above allowed our final model to perform well in the validation set (see Figure 8), and to learn

information from both the ai-cook dataset and our own images. This allowed the model to correctly identify foods in static images of different formats and in a video stream.

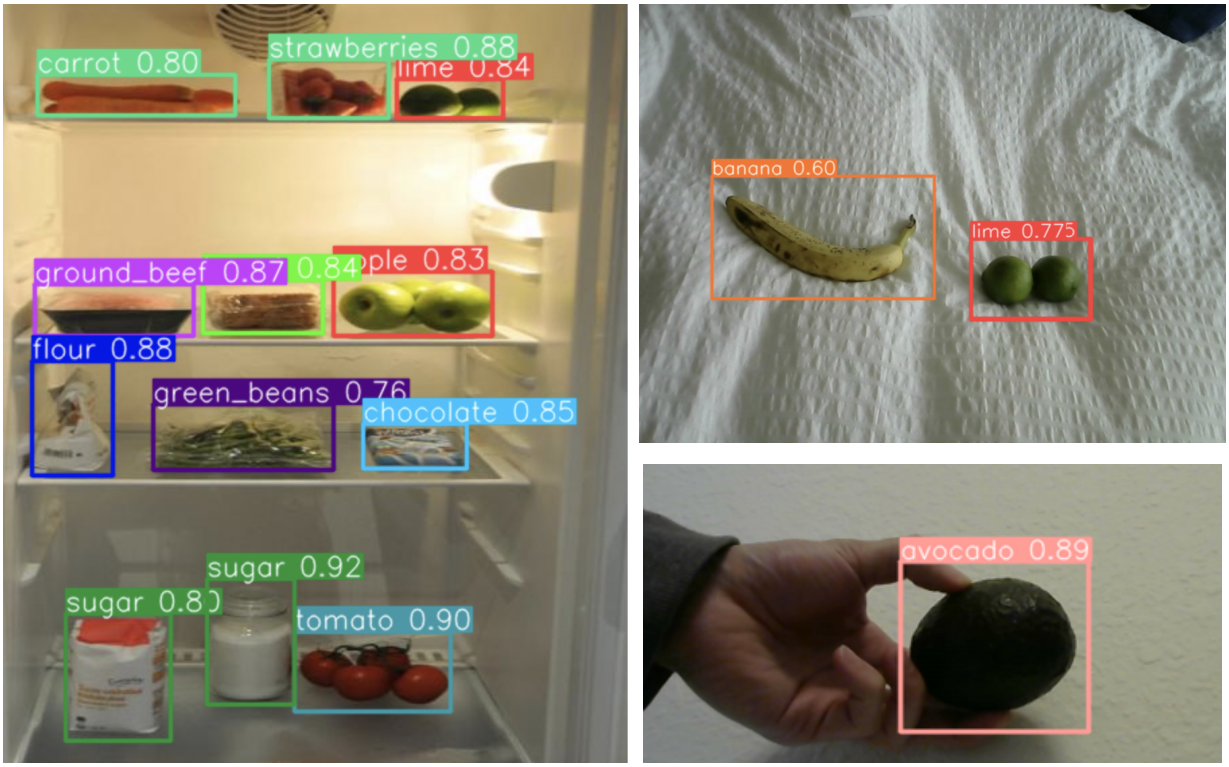


Figure 9: Examples of inference in our last model.

3. Inference

In order to connect to our model and make it usable from an application standpoint, our team has created an environment on the edge device (Jetson Nano) where we are able to connect to the camera, capture frames containing different foods, and add these to an inventory list. The architecture uses a similar pytorch container including all dependencies to start our web service, load our model, stream from the camera, and set up our user interface.

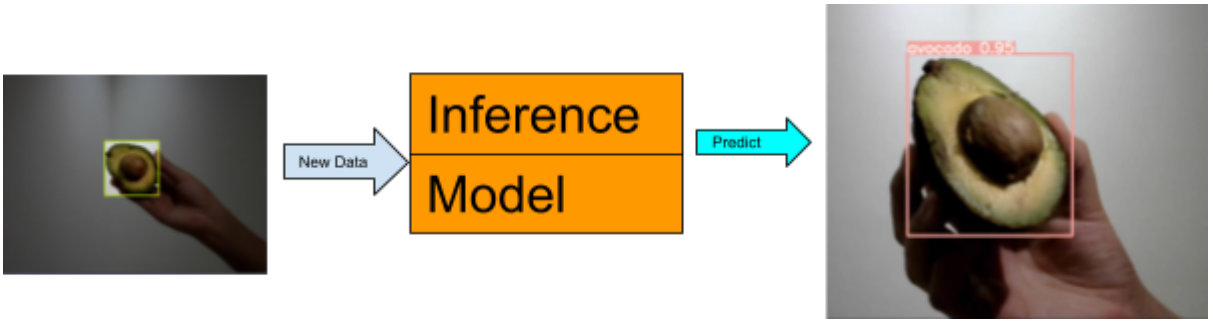


Figure 4: Model Inference

4. Recipe builder with GUI Interface

The user can add food into the inventory database by clicking on the add ingredient button. At that moment the webcam and model are activated, a pop-up appears showing the webcam to help the user scan in their item. Once an ingredient has been detected, the frame will freeze and show the detected ingredient in order to have the user confirm if their food has been correctly identified. If the user confirms, the food is added to the ingredients inventory and the recipe list will generate based on the current inventory. Currently the recipe generator provides the top 10 suggestions for recipes, and tells the user which ingredients they already have accounted for and which they would need to supplement.

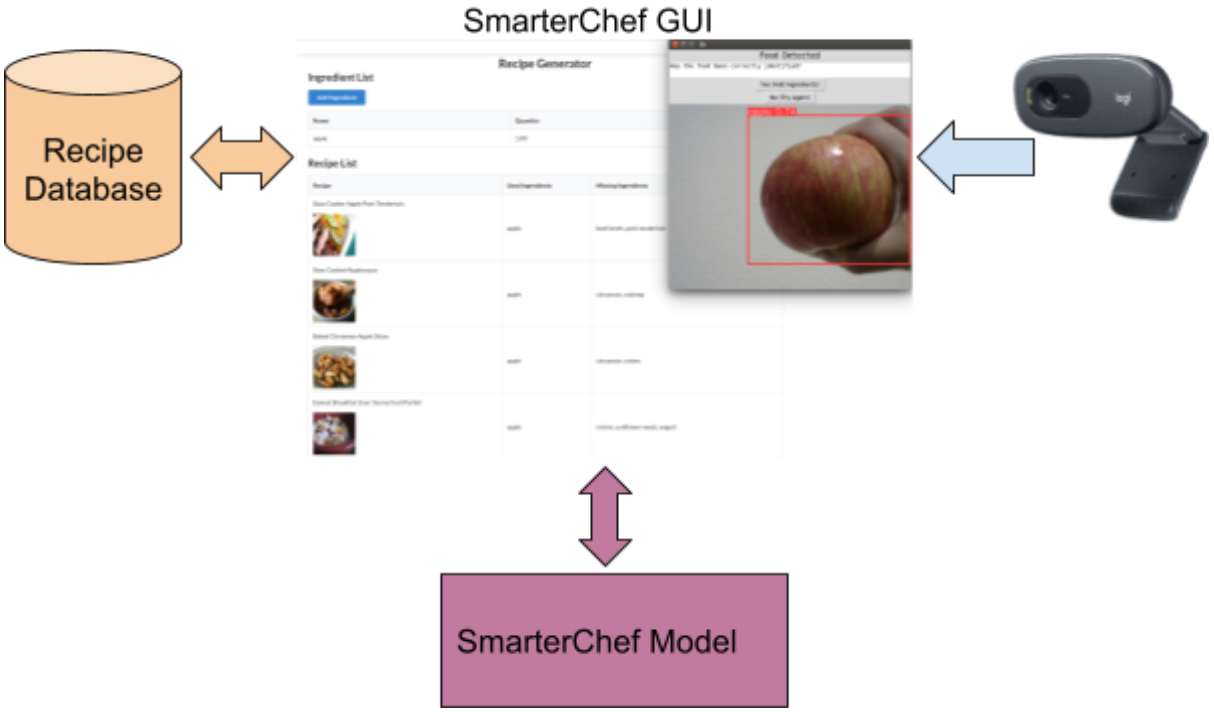


Figure 5: GUI interface

Challenges

1) **Datasets:** Wide breadth of food datasets: there are many datasets related to food that are publicly available, and the variety of the data quickly becomes a challenge. Datasets usually have a large number of classes, and the origin of the images varies considerably. For example, for nutrition research purposes the datasets like UEC-FOOD10 [2] have become popular, but they are made from foods that are mostly consumed in Japan, so that introduces a bias from the beginning. Food as a subject is itself very complex. For instance, there was a project in which researchers used deep learning to successfully classify 73 different classes of bread alone [3].
2) **Edge Device:** We had used the Jetson Nano 4G device to run our user interface. It is a powerful small computer ideal for neural networks and object detection but had a hard time keeping up with the load of our application. The combination of a constant video stream and model detection during the scan would cause a lag to occur. It would also take several minutes for the application to load the model in and be ready for the user.

3) **Video Detection:** Initially, we had trained our model using only still food images we had found on publicly available datasets. While we were able to achieve a high validation accuracy, the results did not replicate when tested through the stream of our Jetson. In order to resolve this issue we had to create our own supplemental dataset based off of still frames of food shot through our Jetson camera. There is a limitation of object detection in which the camera and the environment that the training data is taken from will affect the accuracy of the results and practical use.
4) **Custom Annotation:** As we learned in class, custom annotation of images is the first thing needed to create an original dataset. This task, however, requires a considerable amount of time in order to achieve a good volume of training data. To try to overcome this, and to increase the robustness of our model at the same time, we implemented augmentation techniques based on the AI-Cook project results [12]. An important limitation we found while doing this was that the free version of roboflow only allows you to increase thrice the number of your original dataset through augmentations. In other words, we could only create three images out of an originally annotated image in the free version of Roboflow. This meant that we needed to annotate a large number of images by hand in order to create a large enough dataset, even after applying augmentation techniques. The final dataset we used for this project can be found in this link. To create it we annotated 4,193 images, added the original images from the ai-cook project, and implemented the augmentations described in the dataset section.

Future Work

1) **Cloud Implementation:** To make our application scalable we would ideally send the frames of the stream to a container sitting in the cloud which would perform the model evaluation on the cloud and simply send back the matches. Removing the model implementation from the edge device would decrease the load required of the device of our end user and would avoid a lot of the more complicated software setup.
2) **Mobile Device:** Ideally our application would allow for support on a mobile platform. Since one of our main target demographics would be grocery shoppers, a mobile application would significantly increase their ease of use.
3) **Database Knowledge:** Currently we have focused our model training on only a subset of the most popular food. Ideally we would expand upon the number of ingredients that our model is able to detect to provide our end users a larger database.
4) **Automatic Detection:** For the ease of our end user, we would ideally tune the model to be able to create the database of ingredients on its own through automatic monitoring of the user's grocery basket and all food items in their fridge or pantry.

Repo Information

To run the program on the Jetson Nano, you can clone the github repo:

https://github.com/jmiller558/w251finalproject.git

The repo contains instructions in its README.md file, and all files necessary to build the docker image and then run the docker container and program.

References

[1] *Food Waste in America. Facts and Statistics*. RTS 2022. https://www.rts.com/resources/guides/food-waste-america/

[2] *The Food Recognition Benchmark: Using Deep Learning to Recognize Food in Images.* Monhaty et al. Front. Nutr., 06 May 2022. Sec. Nutrition Methodology https://doi.org/10.3389/fnut.2022.875143

[3] *Food Image Recognition by Deep Learning*. Assoc. Prof. Steven HOI. FOODAi. School of Information Systems Singapore Management University https://images.nvidia.com/content/APAC/events/ai-conference/resource/ai-for-research/FoodAI-Food-Image-Recognition-with-Deep-Learning.pdf

[4] *Study for Food Recognition System Using Deep Learning*. Nareen O. M. Salim *et al* 2021 *J. Phys.: Conf. Ser.* 1963 012014 https://iopscience.iop.org/article/10.1088/1742-6596/1963/1/012014/meta

[5] *LogMeal. Artificial Intelligence and Deep Learning Solutions for Food Recognition. Official Page.* https://www.logmeal.es/#team

[6] http://foodcam.mobi/dataset100.html

[7] Pishva D, Kawai A, Hirakawa K, Yamamori K, Shiino T. Bread recognition using color distribution analysis. *IEICE Trans Inf Syst*. (2001) 84:1651–9.

[8] FoodNet: Recognizing Foods Using Ensemble of Deep Networks. Pandey et al. https://arxiv.org/abs/1709.09429v1

[9] *Food recognition using deep learning networks and order history for smart canteen checkout automation.* Rinat Sadekov. March, 2022. https://www.researchgate.net/publication/359051832_Food_recognition_using_deep_learning_networks_and_order_history_for_smart_canteen_checkout_automation

[10] *Figure: The network architecture of Yolov5.* Taken from A Forest Fire Detection System Based on Ensemble Learning. Xu et al. https://www.researchgate.net/figure/The-network-architecture-of-Yolov5-It-consists-of-three-parts-1-Backbone-CSPDarknet_fig1_349299852

[11] *Yolo Documentation*. https://docs.ultralytics.com/

[12] AI Cook Image Dataset. Roboflow. https://universe.roboflow.com/karel-cornelis-q2qqg/aicook-lcv4d