Better Ate Than Never:

Using deep learning and analytics to combat food waste

Dinesh Achuthan, Suryabrata Dutta, Ammara Essa, Girija Ghali

W251, Dec 2020

Table of Contents

1. Abstract *3*

2. Background *4*

3. Design *5*

3.1 Product Goals 5

3.2 Assumptions 6

3.3 Constraints 6

3.4 Out of Scope Tasks / Enhancements 7

3.5 End-To-End Pipeline 7

4. Implementation *8*

4.1 Compiling Training and Validation Images 8

4.2 Compiling Images of Fresh Produce 9

4.3 Compiling Images of Rotten Produce 10

4.4 Image Annotation and Augmentation 10

4.5 Training New Models with Yolov5 Transfer Learning 12

4.5.1 Environment Setup 12

4.5.2 Yolov5 Configuration 13

4.5.3 Yolov5 Model Training 13

4.5.4 Operational and Hyper Parameters 14

4.5.5 Results chart 15

4.6 Deploy the Trained Model on Xavier Edge Device 17

5. Lessons Learned *19*

5.1 End-to-End Pipeline 19

5.2 Model Training 20

6. Conclusion *21*

6.1 Future Enhancements 21

7. Appendix *22*

7.1 References 22

7.2 Additional Information 23

1. Abstract

How often have we heard pleas from non-profits organizations to help feed the underprivileged, both home and abroad? While many of us are privileged to not be impacted, there are still a staggering more than 820 million people for whom food insecurity is a daily struggle. You may be inclined to think that there is a global food shortage but you would be surprised to learn that the United Nations Food and Agricultural Organization (FAO) estimates one-third of all food produced in the world (approximately 1.3 billion tons) is lost or wasted every year. In the U.S. alone, an estimated 133 billion pounds of edible food worth over $161 billion goes to waste every year. That is approximately 219 pounds of waste per person!

Food waste does not only exacerbate the obvious problem of food insecurity. Its heavy carbon footprint has serious environmental ramifications as well. Any unused produce means that we also waste the resources required to grow and transport it: water, energy, fertilizer, labor and transportation etc. Once the food is disposed of in a landfill, it rots and becomes a significant source of methane - a potent greenhouse gas that contributes to global warming. In fact, food waste contributes to the largest volume of material in U.S. landfills accounting for 21% of the waste stream. In this paper we will demonstrate a proof of concept design which leverages deep-learning models on an edge device to help combat the issue of food waste.

2. Background

The production, distribution, consumption and ultimately the disposal of food items are part of a undeniably complex system. Food finds its way to landfills throughout the supply chain, from initial agricultural production down to final household consumption. Terms like food loss and food waste may often be used interchangeably but for clarity, we will highlight the difference between them. Food loss refers to any food that is discarded, incinerated or otherwise disposed of post-harvest along the food supply chain from harvest/slaughter/catch up to, but excluding, the retail level, and does not re-enter in any other productive utilization, such as feed or seed. Food waste refers to food that is discarded by retailers, food service providers and consumers. Such waste can occur in many ways such as:

- Fresh produce that does not meet "acceptable" aesthetic standards shape, size or color even though their taste or nutritional value remains completely uncompromised. Such produce is often removed from the supply chain during sorting operations and never even makes it to the consumer.
- Foods that are close to, at or beyond the "best-before" date are often discarded by retailers and consumers.
- Unused or leftover food products that are discarded by consumer households and restaurants etc.

Finally, food loss and waste varies across different categories, with fresh fruits and vegetables having one of the highest rates.

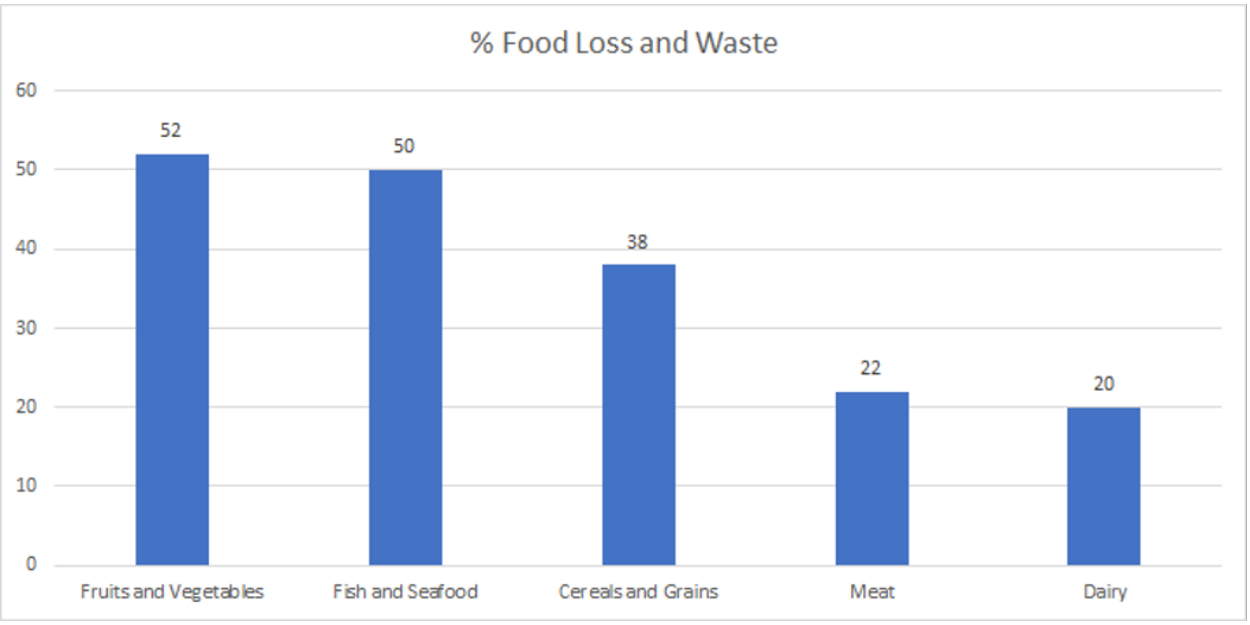


Fig 2.1. % Food Loss and Waste by Category (calculated collectively for USA, Canada, Australia, and New Zealand)
Source: FAO.org 2011

We firmly believe that most people understand that wasting food is undesirable. However, beyond good intentions, we also think that having a system that accurately quantifies the amount of food waste in any particular setting can help people take better informed remedial actions. Our product is designed to weigh, detect and distinguish between good unprocessed food products such as whole fruits and vegetables (that are both aesthetically acceptable and fit for consumption) and produce which has rotten. The data (which includes total weight of items placed on the scale and category of item detected etc) is stored in an Amazon cloud based DynamoDB. The ultimate goal is to help the user (be it a producer, retailer or consumer) gain insights into their supply chain / fresh produce inventory and subsequent waste footprint. Since waste can occur at several stages of the food cycle, the system can be placed at various interception points e.g.

- At the time of production / harvest
- Retail / grocery store level
- In the consumer's home

To start with, our product can distinguish between two categories (good or rotten) of apples, oranges and bananas. Additional training data can be used to train the model to include other common grocery items. Extensions of this product can have several applications. For instance, rather than focusing on rotten items, the model can be fine-tuned to detect and categorize aesthetically imperfect produce that is still perfectly suitable for user consumption. This information can then be shared with local shelters and food banks (via an app or push notification), thus bridging the gap between food surplus/waste and food insecurity.

3. Design

3.1 Product Goals

End Product Goal: A deep learning based edge device capable of accurately performing multi-class classification of food items at varying degrees of freshness, ranging from freshest to rotten. This data will enable users (businesses or individuals) to better understand gaps (or surplus) in their food supply chain and help them take timely actions that reduce both food and monetary loss. Furthermore, by identifying and isolating food items before they become unviable for consumption and connecting the customer to local nonprofits etc, this product can help alleviate the issue of food scarcity. Multiple edge devices from multiple locations streaming food classification data to centralized location where a deep learning model is trained and continuously pushed back to the edge device and in the process captures key metadata to make better business decisions.

Modified Product Goal: End-to-end pipeline with manual model deployments and storage from a single edge device to show the capability of deep learning to create the unrealized business value thus far (Fig. 3.1).

3.2 Assumptions

- Only 6 categories of fruits will be chosen for this experiment
- Only one end to end pipeline with a single edge device will be used.
- mAP@0.5 i.e. mean Average Precision to be reported at 0.5 Intersection over Union (IoU)
- No automation of model deployment as envisioned in our original design.

3.3 Constraints

- **Limited time:** Not surprisingly, the short duration of the project proved to be a challenge to deploy a fully optimized and trained model with end to end automation. We did manual deployment of trained models, manual labelling to get the end to end pipeline working. We did not leverage the weight parameter to further augment our final thoughts by projecting the cost saving from the rotten fruits to the business.
- **Scarcity of rotten food images:** Surprisingly, there are not enough unique rotten fruit images readily available on the internet. Though we applied various forms of image augmentation techniques to increase the training dataset, we felt the number of images were still not enough for optimal model training.
- Interfacing with the Jetson GPIO pins: our original design utilized the Jetson to read out the load cell weight. However, the firmware for the Jetson did not set and reset the GPIO pins in the correct way for us to read the load cell amplifier correctly, so we had to move this part of our system to a separate Raspberry Pi and connect via MQTT.
- **Budget**:
    - AWS Instance: For full training and even for higher batch side to train a model like Yolov, we need upfront investment on planned infrastructure
    - Licensed images: Model performance improves with a more robust training dataset. However, we were not able to find 1000s of unique rotten food images free of cost. While we paid for several licensed images, it quickly became clear that this would become a very expensive exercise with diminishing return on investment.
    - Hardware: We procured various types of hardware, such as a 20kg load cell, an HX711 load cell amplifier, raspberry pi and of course the rather expensive JETSON XAVIER NX edge device. The original plan was to stream data from at least two edge devices to the cloud database but ultimately we deployed one edge device in the end to end pipeline due to implementation complexities.

3.4 Out of Scope Tasks / Enhancements

- Multi-class classification of various levels of freshness has not been implemented but it is a candidate for future enhancements.
- Further hyper parameters tuning could not be done due to time constraints
- Automatic sync between the model trained on the cloud and the Xavier. Prevent model drift by implementing a streamlined process to regularly update the model on the cloud with additional training data.

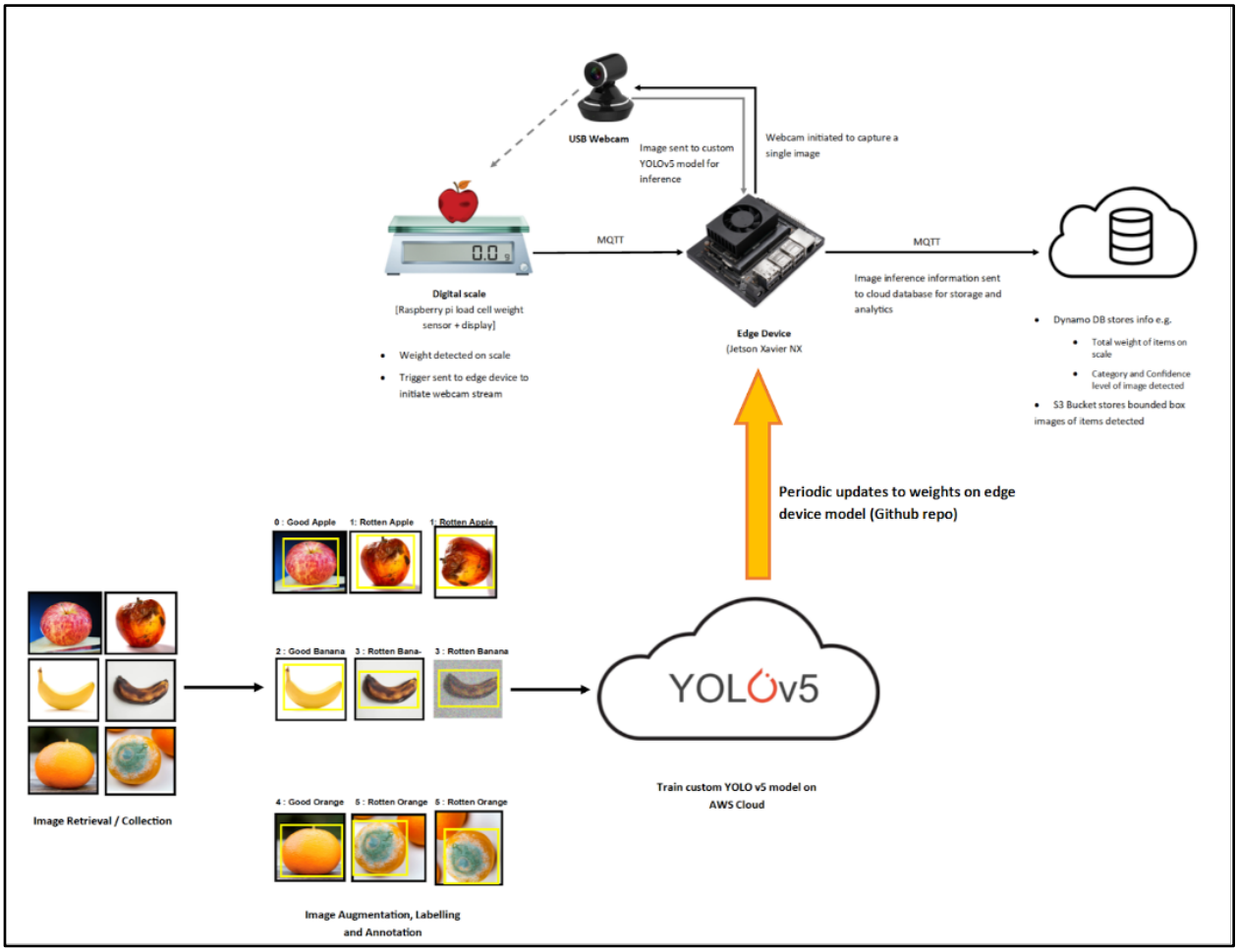


Fig 3.1 End-to-End Pipeline Design

3.5 End-To-End Pipeline

- The training images are compiled and sent to an AWS instance dedicated to training custom models on YOLOv5. The trained weights are saved and pushed to a shared location where the edge device can access and utilize

- The edge devices consist of a few pieces of hardware:
    - A 20kg load cell is connected to an HX711 load cell amplifier and acts as our digital scale and trigger.
    - Both the amplifier and an LCD display is connected to a Raspberry Pi via GPIO pins (our initial design had these connected directly to the GPIO pins on the Jetson, but we were unable to get the firmware to read the HX711 I/O correctly)
    - The Jetson NX does the inference - the USB webcam is directly connected to the Jetson
- When a piece of fruit is placed on top of the scale, the digital scale triggers and sends an MQTT message to the Jetson MQTT broker with weight value. The detection script running the trained weights on the Jetson subscribes to this message - when it receives it, it runs the detection algorithm on a single image, and generates an MQTT message with the image and inference data to send to the cloud MQTT broker.
- Another (small) cloud instance contains another MQTT broker and a script that subscribes to the MQTT message sent by the Jetson device. When it receives a message, it saves the image to an S3 bucket and saves the metadata to a DynamoDB instance

4. Implementation

4.1 Compiling Training and Validation Images

We decided that for this specific implementation the YOLOv5 model was not required to identify all types of objects and the model has been trained with the intention of distinguishing between "good" food items (food items without visible blemishes or rotting) and "rotten" food items only. For the initial deployment, we focused on three items: apples, bananas and oranges. This results in a total of 6 categories with the following labels

Label	Food Item
0	Good Apple
1	Rotten Apple
2	Good Banana
3	Rotten Banana
4	Good Orange
5	Rotten Orange

Table 4.1 Food item categories to be detected and classified

4.2 Compiling Images of Fresh Produce

The COCO dataset contains over 300k images (with labels) for 80 categories of objects, including common fruits and vegetables. Therefore instead of sourcing new images of good fruits from scratch, we leveraged the COCO dataset to compile training and validation images for good apples, bananas and oranges. This was accomplished via a Python code that inspects each label .txt file and saves any images that include labels associated with bananas (label: 46), apples (label: 47) and oranges (label: 49). Moreover, the original label file is modified to remove any labels and annotations not required for training (e.g. cat in Fig. 4.1.1) and renumber the label index to match the new labels in Table 4.1, as shown in Fig 4.2(a) and Fig 4.2(b). Approximately 500 images each were extracted for good apples, bananas and oranges and thus we did not need to increase the dataset via augmentation. While we could have easily compiled more good fruit images from the COCO dataset, we tried to keep the number of good training images consistent with that of rotten training images, which we found to have limited availability. Finally, we manually removed any images in which the unspoiled fruits were not in a whole state e.g. the food items were sliced, mashed etc.

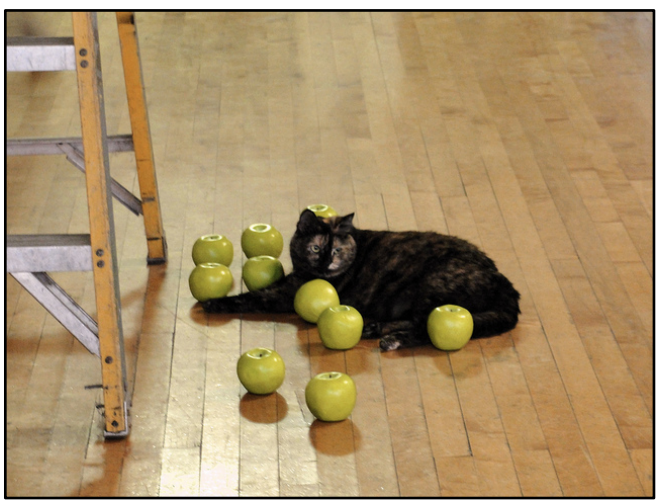


Fig 4.1. COCO Dataset Image: 000000089978

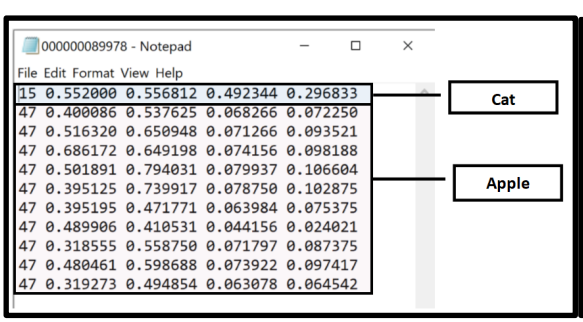


Fig 4.2(a) Original COCO Label for Image 000000089978

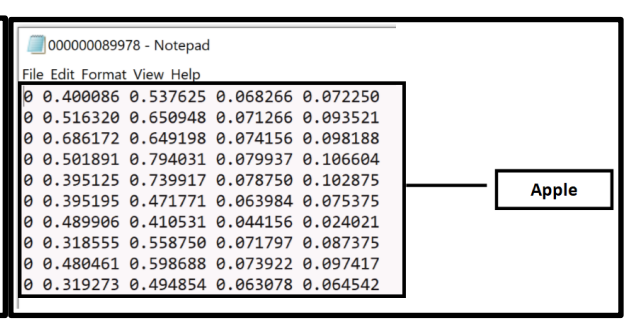


Fig 4.2 (b) Original COCO Label for Image 000000089978

4.3 Compiling Images of Rotten Produce

The COCO dataset did not prove to be a comprehensive source of rotten food images and therefore we sourced these images ourselves from the internet. This included both purchasing licensed images as well as using freely available images. However, after the first few iterations of searching, we found that rotten fruit images were not as readily available as their unspoiled counterparts. We ultimately were able to find ~100 or so sufficiently unique images each of rotten apples, bananas and oranges. Given this limited training data, we utilized several image augmentation techniques (described in Section 4.2) to expand the training dataset in order to improve the model performance and its ability to generalize.

4.4 Image Annotation and Augmentation

As mentioned in the previous section, the COCO dataset provided sufficient images for good fruits (including necessary annotations/labels) such that we did not need to employ image augmentation to increase that particular training set. However, the same cannot be said for rotten fruit images. For Rotten fruit images we scraped the internet and gathered ~100 unique images for each of the rotten groups - Rotten Apple, Rotten Banana, Rotten Orange. We decided to artificially augment these ~100 images, as more data yields better performing deep learning models. Our assumption is more information can be extracted from images by augmenting them.

The most common problem we faced with image augmentation is constructing refined labels. We programmatically constructed the labels for augmented images, but the transformation scale is not model robust. [image below]. We repeated this approach for various augmentation methods like horizontal flip, scaling, Rotation, Shearing and resizing. And the problem with refined labels persisted with each one of these techniques.

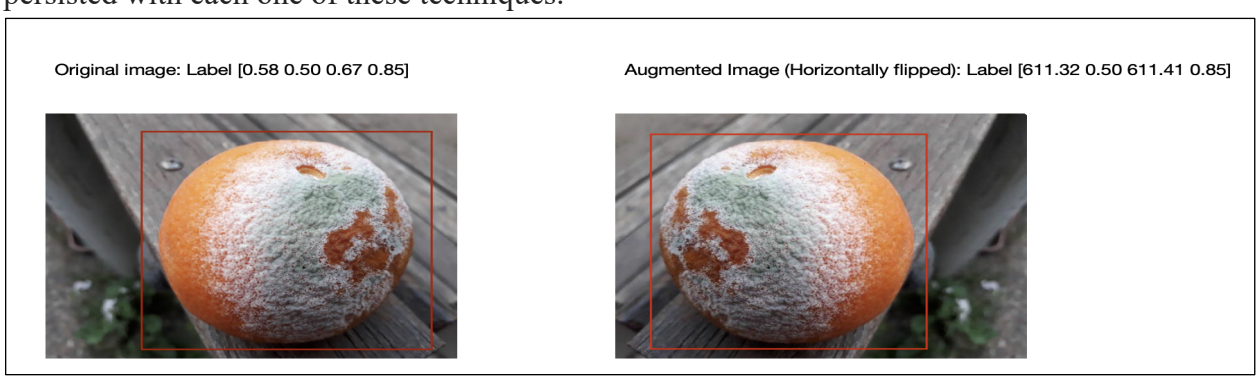


Fig 4.4

After careful consideration, we decided to manually construct the labels post augmentation.
Step:1 We performed the below Image augmentation methods for each of the original images for rotten fruit that we scraped from the internet.

A. Horizontal Flip: The image is flipped along the horizontal axis.
B. Vertical Flip: The image is flipped along the vertical axis.
C. Rotate 45 degrees: The image is rotate 45 degrees
D. Add noise: Adds random noise, here we added a noise of 0.5
E. Blur Image: The image is blurred.

Step2: We used makesense.ai for image annotation and labelling. For each of the 3 rotten fruit classes, we manually labelled and annotated for ~100 original images and ~500 augmented images.

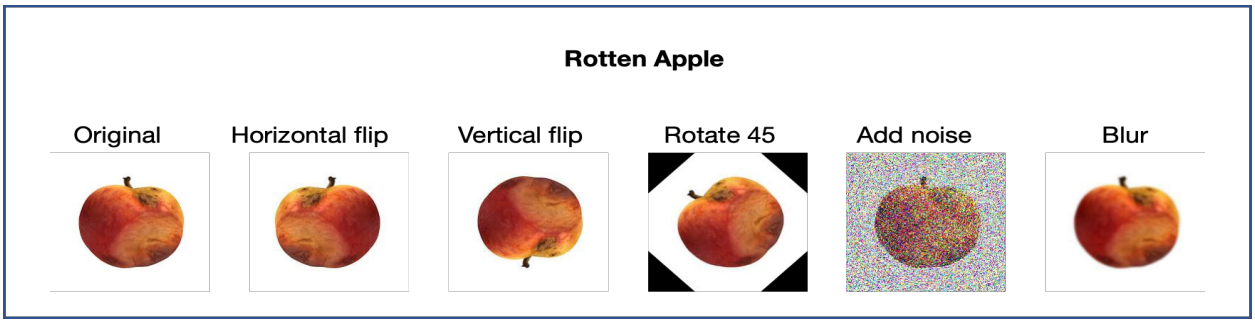


Fig 4.4 (a)

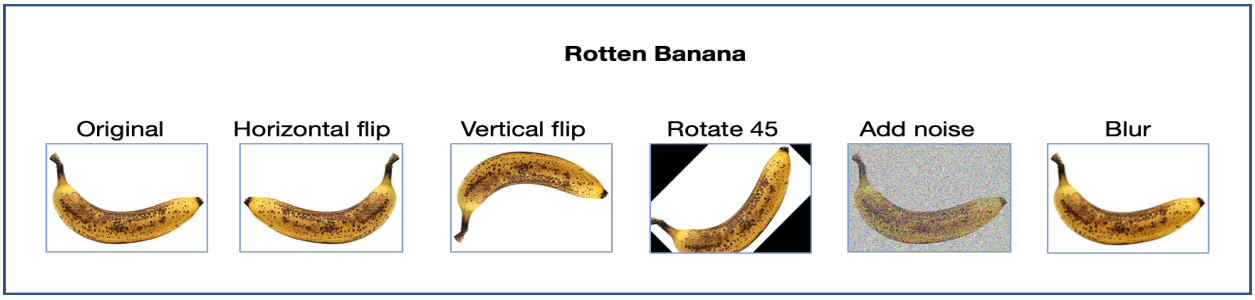


Fig 4.4 (b)

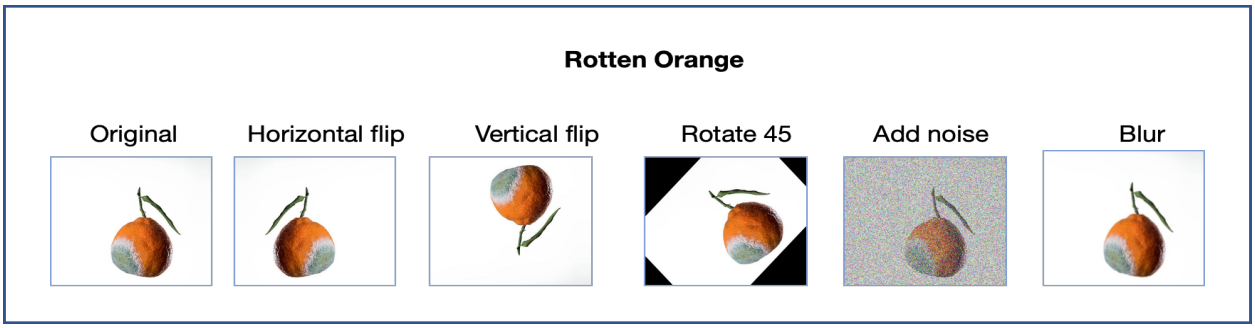


Fig 4.4 (c)

The goal of our project is to build a product that will identify, classify the fresh produce and notify the users. So, we trained our model on images that have multiple classes.

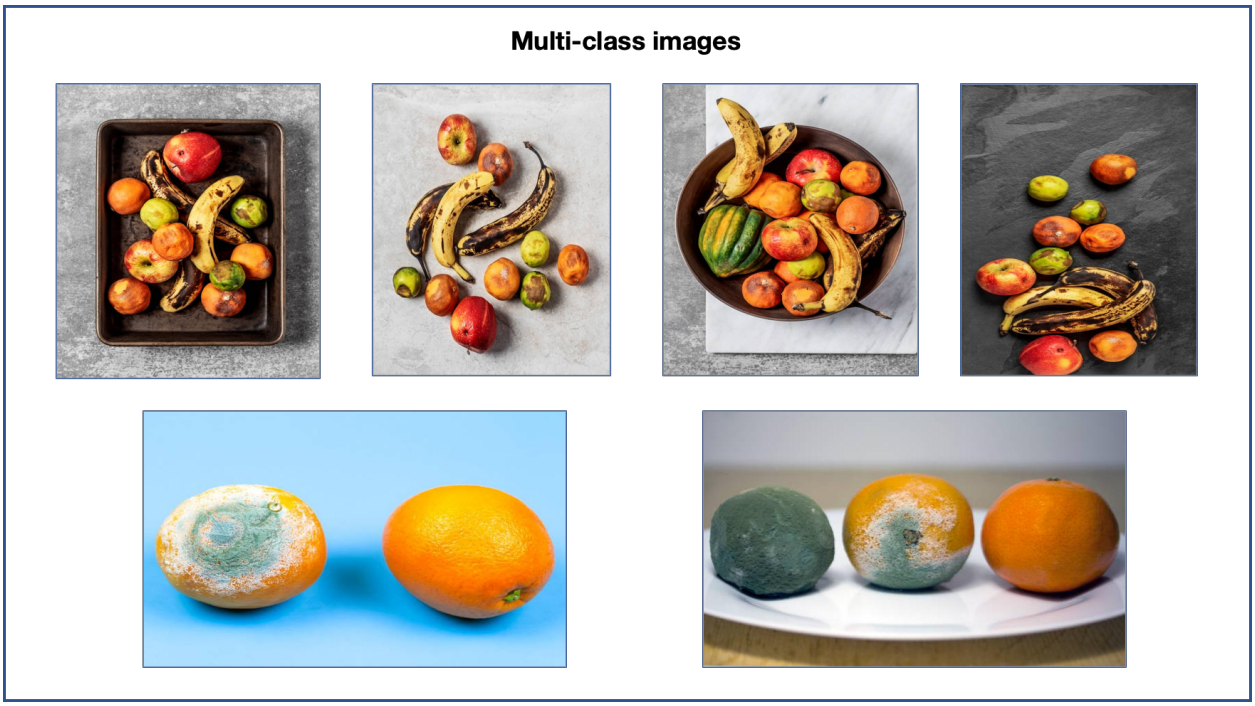


Fig 4.5

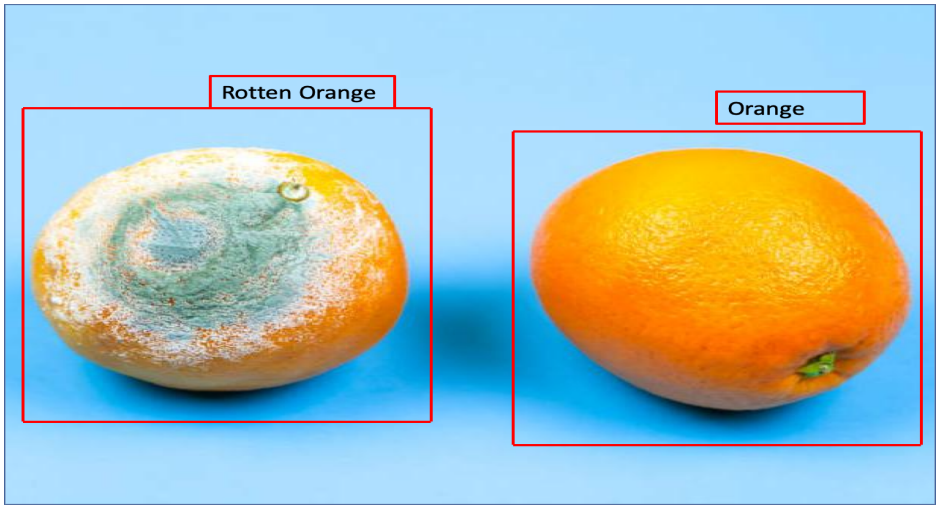


Fig 4.6

4.5 Training New Models with Yolov5 Transfer Learning

Once the rotten and good apples, bananas and oranges are transformed and labeled, the images are split into train and test datasets with 80:20 ratio.

4.5.1 Environment Setup

A docker image is used to avoid environment related inconsistencies. The same docker file could be run either on the Edge Jetson device or on AWS. Below is the docker file used for this project.

```
*********************************Yolov5.Docker ********************************
# https://ngc.nvidia.com/catalog/containers/nvidia:pytorch
FROM nvcr.io/nvidia/pytorch:20.10-py3

# Install dependencies
RUN pip install --upgrade pip
# COPY requirements.txt.
# RUN pip install -r requirements.txt
RUN pip install gsutil

# Create working directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
# Copy contents
COPY . /usr/src/app
********************************** Yolov5.Docker ********************************
```

4.5.2 Yolov5 Configuration

In our model training configuration file, we enter the training images location as well as the location for validation images. The number of classes for classification "nc" in our case is 6. The classification names are configured next. The order of labels should match the names. In our case when we labelled the images, we used 0: apple, 1: rotten-apple and so on. Once this configuration file is set, we are ready for model training and validation

```
********************************** coco-mids.yaml ********************************
train: ../coco-mids/images/train2020/  # approx 2000 images
val: ../coco-mids/images/val2020/  # approx 200-300 images

# number of classes
nc: 6

# class names
names: ['apple', 'rotten-apple', 'banana', 'rotten-banana', 'orange', 'rotten-orange']
********************************** coco-mids.yaml ********************************
```

4.5.3 Yolov5 Model Training

To train a model in transfer learning mode, we leveraged the existing model and weights. The pretrained model and weights are classified as x (extra-large), l(large), m(medium) and s(small).

We used “s” weights and “s” models to quickly test our initial theories as it is the smallest and runs faster. Running the “s” model also helped to resolve configuration issues - for example, on our initial run we realized our labels were out of order and were able to resolve this quickly. Next we moved to the “x” model which is the largest. We used the AWS g.4dn.2xlarge instance and ran 50-60 model trainings by tweaking batch, epoch and image augmentations. Some of the results were captured and shown below. The operational and hyper parameters are shown in the **appendix Table 7.1**. In our experiment we used different combinations of operational parameters of “batch size” and “epochs” for different yolov5 models and evaluated the precision and recall curve along with mAP score.

Table 4.3: Summary of subset of models tested

Experiment Name	Yolov Model	Batch Size	Epoch	mAP@0.5
exp15	x	16	20	0.712
exp5	x	8	20	0.696
exp16	x	16	100	0.637
exp17	x	16	25	0.672
exp18	x	8	25	0.659
exp4	x	8	20	0.697
exp3	l	16	5	0.506
exp7	m	8	5	0.518

4.5.4 Operational and Hyper Parameters

With our AWS instance, we ran into memory constraints quickly. When we chose a batch size above 16, we encountered resource exceptions. With more time, we would experiment with an instance with a larger GPU memory bandwidth to see if our mAP score improves.

The operational and hyper parameters for the experiment are shown in **appendix Table 7.1**. The main hyperparameters we modifier are model size, epochs and batch size.

4.5.5 Results chart

The overall experiment results charts and data are shown below. The precision, recall, map@0.5 are the three primary metrics we utilized to tune and assess model training performance. Both the precision and recall chart show the better results were obtained during 15-18 epochs for experiment 5 and 15-25 epoch for experiment 16. The first row of the chart is for training and the second row is for validation.

Precision is measured to show how often the model guesses correctly and recall is the measure to show whether the model guessed every time that it should have guessed. Object detection systems make predictions in terms of a bounding box and a class label. In theory, the bounding box prediction is incorrect if it is the wrong class, but where should we draw the line on bounding box overlap? The Intersection over Union (IoU) provides a metric to set this boundary at, measured as the amount of predicted bounding box that overlaps with the ground truth bounding box divided by the total area of both bounding boxes. We have charts for both 0.5 IoU as well as 0.5 to 0.95 IoU.

In this experiment we learnt the importance of bounding box labelling exercise for the training images. Getting the precise bounding box for the training image set is a very key and important task to get better mean average precision.

In the below chart both the precision and recall are max around 16-18 epochs and so is mean average precision.

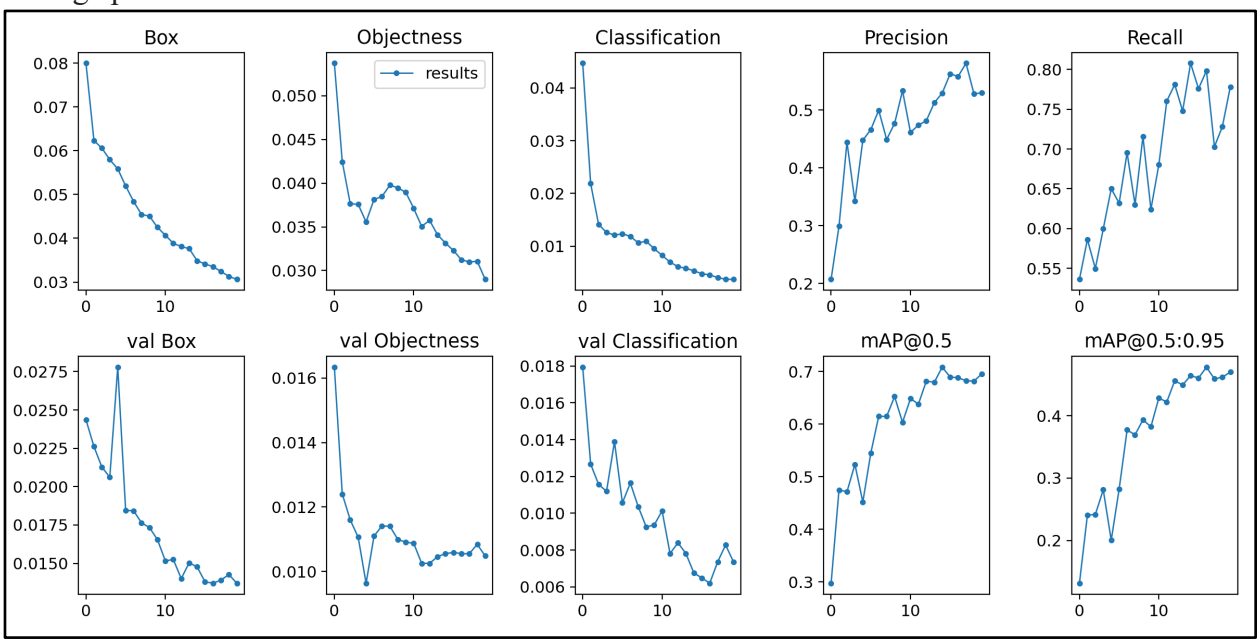


Fig 4.3.1 : EXP5 - batch=8, epoch=20 - Results:

In the chart below, both the precision and recall are at its peak approximately around 25 epochs. More epoch did not add much value with either precision or with recall.

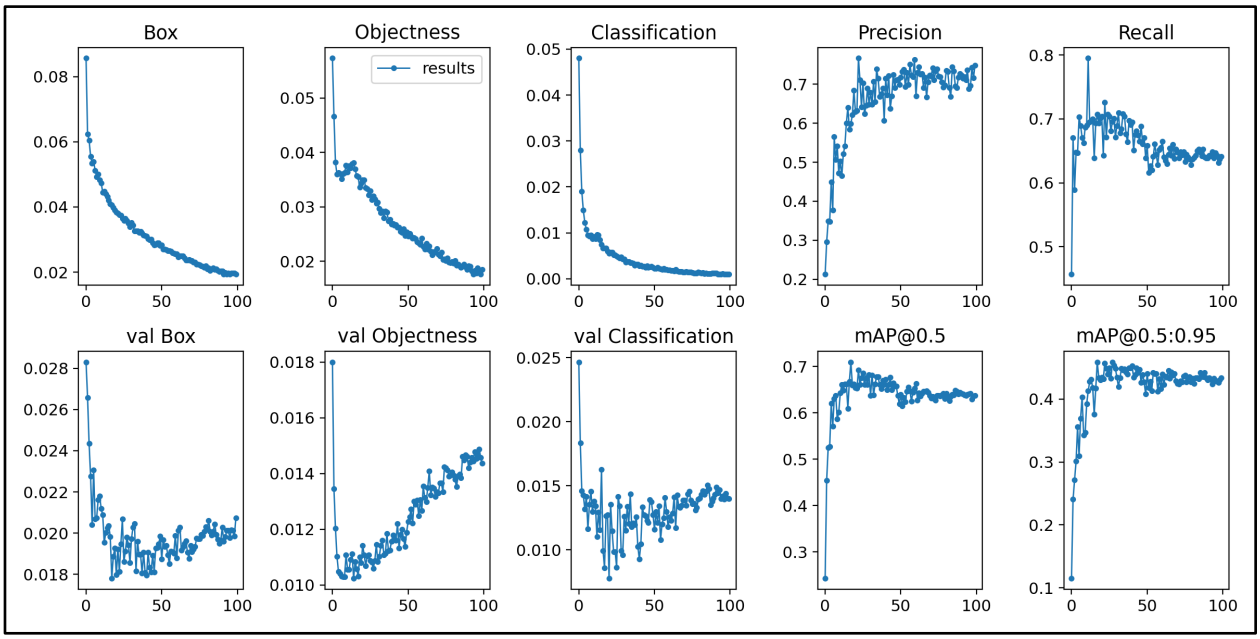
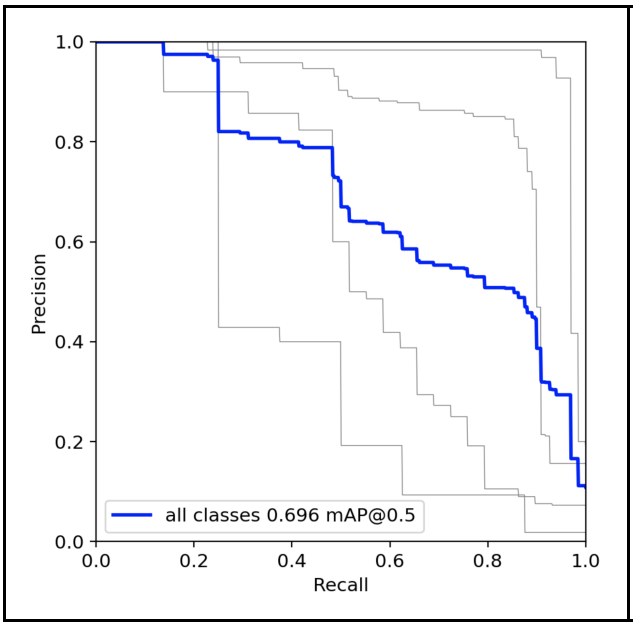


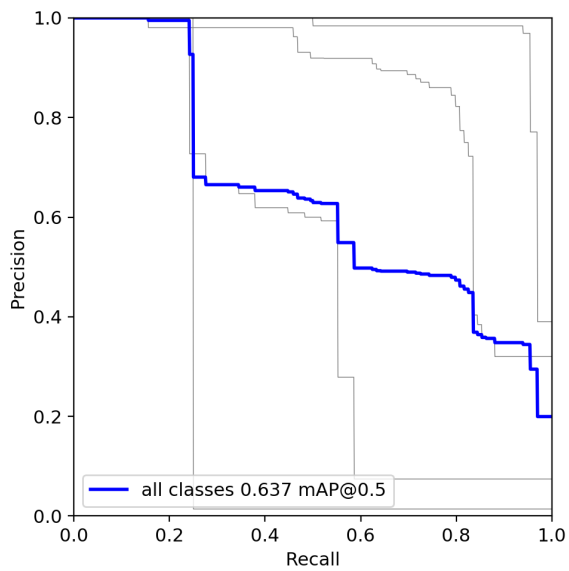
Fig 4.3.2 EXP16 - batch=16, epoch=100 - Results



The precision recall curve tells us the mean average precision @0.5 is 0.696 for batch size 8, 16 epochs for yolov5 x model when initialized from x-weights. Any mAP value greater than 0.5 is considered to have predictive power, and there is potential to optimize the experiment by experimenting with different batch size, epochs and also by adding additional images for training and finally by tweaking the hyper parameters.

Fig 4.3.3 EXP5 - batch=8, epoch=20 - Precision Recall Curve

EXP16 - batch=16, epoch=100 - Precision Recall Curve



Another sample precision recall curve for batch size 16 and 100 epochs. Spending time and effort in putting a precise bounding box for training image labels is very important to get a higher mean average precision score.

Fig 4.3.4 : Various stages of ripeness of a banana

4.6 Deploy the Trained Model on Xavier Edge Device

Once the model is trained on AWS GPU machines, the best.pt file is transferred to the Xavier for real time rotten and good fruit prediction. We initially utilized Git LFS to transfer the large trained weight files, but eventually switched to S3 buckets.

We conducted a test of the E2E pipeline with a handful of trained weights. We were constrained by the number of rotten fruits we could produce - we were able to obtain one sample for each category. The rotten specimens were chosen to be extremely rotten, while the fresh specimens did not have any flaws. Because our sample size was essentially 1, we only collected 5 data points per sample at different positions (we turned the subject to a different angle between collecting each data point). Each data point has the confidence level with which that fruit was detected in the image - the idea being that we can set a “threshold” for rotten fruit given our model performance and consider any rotten fruit label above this threshold of confidence to be actually rotten. This gave us a rough estimate of our model’s performance, but we do not assert these results holistically evaluated our models.

Table 4.6 - Confidence Levels of Inference on Jetson Xavier Edge Device [0,1]

	Exp 3	Exp 4	Exp 5	Exp 6
Good Red Apple	0.56	0.92	0.86	0.90
Rotten Red Apple	0.56	0.27	0.54	0.64
Good Banana	0.23	0.88	0.74	0.77
Rotten Banana	0.37	0.25	0.36	0.33
Good Orange	0.62	0.91	0.87	0.93
Rotten Orange	0.14	0.57	0.80	0.73

Exp 3: Batch Size 16, 5 Epochs
Exp 4: Batch Size 8, 20 Epochs
Exp 5 & 6: Batch Size 16, 20 Epochs (different hyperparameters)

Based on this limited testing, we observed that:

- Exp 3 performed worse with both good and rotten fruit, indicating that more epochs were needed for convergence
- Exp 4, with a batch size of 8, consistently classified good fruit with high confidence, but failed to recognize rotten fruit as well as the other experiments. We hypothesize that because we are utilizing transfer learning to train our models, a batch size of 8 is not large enough to effectively learn the distinction between rotten and good fruit (rotten detection is underpowered)
- Exp 6 performed slightly better than Exp 5 on good fruit, but Exp 5 performed better than Exp 6 on rotten fruit. Because the latter is more important to our use case, we chose to iterate on the hyperparameters selected in Exp 5.

5. Lessons Learned

5.1 End-to-End Pipeline

- Interfacing with GPIO pins on the Jetson Xavier is very finicky. The GPIO pins do not follow the same conventions as GPIO pins on the Raspberry Pi, and many python software libraries like RPi.GPIO are not compatible. We had a lot of difficulty specifically getting the HX711 load cell amplifier to work when directly connected to the Jetson - some pins would not read out the right values or update correctly. We were not able to figure out the root cause of this issue, and instead pivoted to reading out the values on a separate Raspberry Pi. (This added more latency to the final project)

- We had similar issues while trying to get the Raspberry Pi camera working on the Jetson. Using this Github repository, we were able to detect images successfully, but we were unable to pass through this interface effectively to our docker container (there was a system timing issue). We utilized a generic USB webcam instead.

- Despite the MQTT protocol allowing for messages up to 260 mb in size, we encountered issues sending images larger than ~15-20kb. We experimented with encoding methods that would break up the image into chunks of 6kbs and rearrange them on the cloud, but in the end decided to downscale the image for simplicity. A productionized pipeline would require a more robust delivery system.

- The lighting of the subjects did not drastically change the model performance in the E2E pipeline, with one key exception: green apples were being misclassified as oranges under warm lights. To counteract this, we added a cool white (5000K) light source near the platform. More experiments with lighting and camera angles could be helpful.

- We also faced challenges transferring large files trained weights through Git. We started with Git-LFS to overcome some of these challenges, but when we reached our bandwidth cap, we moved to S3 to transfer these files between the training cloud instance and the E2E pipeline/edge device

5.2 Model Training

- Because we were constrained by the number of training images we could use for our model, we decided to utilize transfer learning from the pretrained YoloV5 weights, which had already been trained on good fruit. However, this led to many of our models recognizing rotten fruit as good fruit. The inference of the rotten fruit is essentially underpowered.

    - Larger models (yolov5-l and yolov5-x) fared better than their smaller counterparts in distinguishing between rotten and fresh fruit. We hypothesize this was because they could capture more subtle differences in the images
    - The underperformance of detecting rotten fruit also originates from our method. We were able to leverage the COCO dataset to find many images of fresh fruit. However, for the rotten fruit we had a more limited set of images that we had to augment to balance the classes. We were limited by diversity.

- During training in the cloud, the main limiting factor was memory. We kept on encountering out-of-memory exceptions on our instances, and this prevented us from increasing parameters like batch size. We ended up utilizing the G4dn.2xlarge instance with a max batch-size of 16

6. Conclusion

Quantifying and understanding the scope and characteristics of food waste is the first step in developing any preventative and recovery measures, be it a producer, retailer or individual consumer. By deploying our system at the appropriate interception points, the user can not only gain insights into what kinds of items are ultimately ending up in the landfill but also in what quantities, which in turn translates into monetary loss. Over time, the user can also identify patterns that may emerge due to seasonality, popularity of an item etc. All this information combined will enable the user to better adjust production, sale or consumption to minimize their waste footprint, thereby helping alleviate this global issue, one apple at a time.

6.1 Future Enhancements

We recognize the potential for growth of this system. On the model side, the most important short-term enhancement is to add more categories of rotten fruits, vegetables, other food items like breads, canned foods, processed and unprocessed foods etc. In the long term, we would like to have a multi-class classifier for each individual item that can distinguish between the various stages of freshness. We are particularly interested in this functionality because there is an opportunity to identify food items that may have passed their aesthetic prime (and are thus removed from the grocery stand) but are still fit for consumption as their taste or nutritional value remain uncompromised. In example shown in Fig 6.1, a retailer could potentially remove bananas at stage "4" and can connect with local food banks or shelters to donate the items instead of throwing them away.

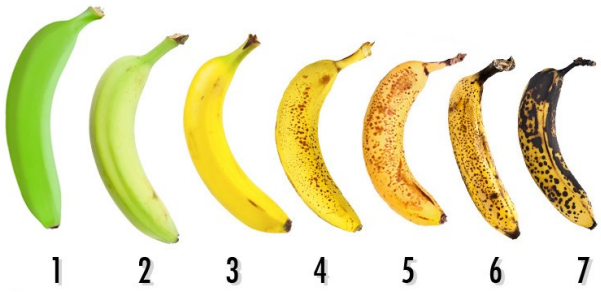


Fig 6.1 : Various stages of ripeness of a banana

Another crucial way we can add value to this pipeline is to add a feedback mechanism between the inference data collection and the model training step. The images that are taken from the edge device can be fed back through the model training step to continuously improve the performance of our system. To avoid false classifications from amplifying poor performance, we add a human verification step in between data collection and retraining. With enough data flowing through the system, we can also leverage this design to utilize other metadata we collect like weight using external sensors and deploy this design to more complex environments and a wider variety of food products.

7. Appendix

7.1 References

1. UC Berkeley MIDS - W251. "DS W251 Final Project GitHub Repository." *GitHub*, http://github.com/w251-final-project-fall2020/final_project
2. Au, Eden. "Uploading Large Files to GitHub." *Medium*, Towards Data Science, 27 Dec. 2019, http://towardsdatascience.com/uploading-large-files-to-github-dbef518fa1a
3. Codebox. "Codebox/image_augmentor." *GitHub*, http://github.com/codebox/image_augmentor
4. "Common Objects in Context." *COCO*, http://cocodataset.org/
5. Git-Lfs. "Git-Lfs/Git-Lfs." *GitHub*, 2 Dec. 2020, http://github.com/git-lfs/git-lfs/blob/master/docs/man/git-lfs-migrate.1.ronn?utm_source=gitlfs_site
6. Géron, Aurélien. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, 2020.
7. JetsonHacksNano. "JetsonHacksNano/CSI-Camera." *GitHub*, http://github.com/JetsonHacksNano/CSI-Camera
8. Kathuria, Ayoosh. "Data Augmentation For Bounding Boxes: Flipping." *Paperspace Blog*, Paperspace Blog, 2 Dec. 2019, http://blog.paperspace.com/data-augmentation-for-bounding-boxes/
9. *Make Sense*, http://www.makesense.ai/
10. Ultralytics. "Ultralytics/yolov5." *GitHub*, http://github.com/ultralytics/yolov5
11. Ultralytics. "Ultralytics/yolov5." *GitHub*, http://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

7.2 Additional Information

CMD used to train the model

```
docker run --privileged  -e DISPLAY=$DISPLAY --runtime nvidia --rm  -v /tmp:/tmp --shm-size=4g -ti yolov5 python3 train.py --batch 16 --epochs 5 --cfg ./models/yolov5m.yaml --data ./data/coco-mids.yaml --weights ./weights/yolov5m.pt
```

CMD used to run the model detection on the Xavier

```
docker run --privileged --runtime nvidia --network final_project --rm  -v /data:/data -e DISPLAY -v /tmp:/tmp -ti final_project python3 detect.py --source 0 --weights runs/train/exp5_x.16.20/weights/best.pt --conf 0.1
```

Table 7.1: Operational and Hyper Parameters used in experiments

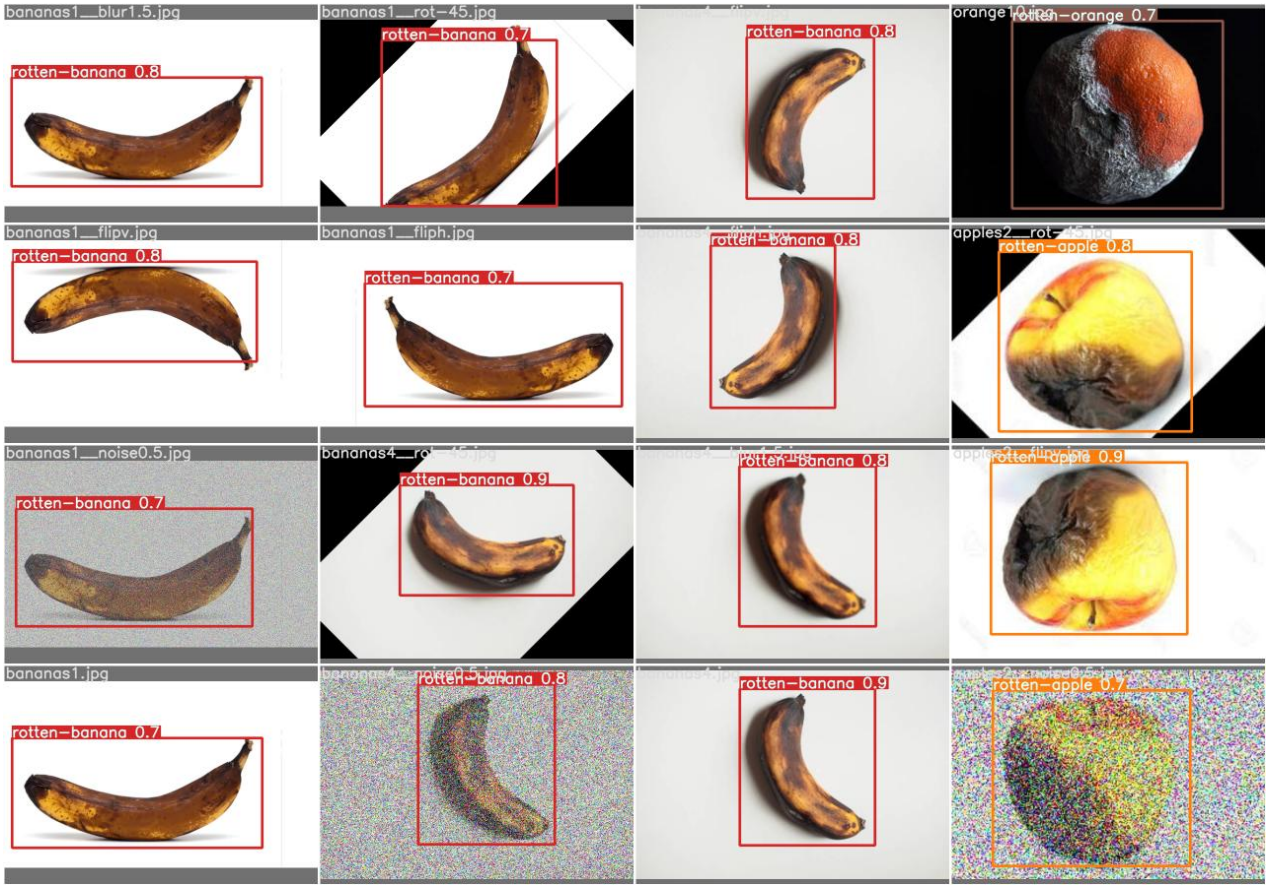
Operational Parameters (Example: Exp5)	Hyper-Parameters (Default for all experiments)
weights:./weights/yolov5x.pt	lr0:0.01
cfg: ./models/yolov5x.yaml	lrf: 0.2
data: ./data/coco-mids.yaml	momentum: 0.937
hyp: data/hyp.scratch.yaml	weight_decay: 0.0005
epochs: 20	warmup_epochs: 3.0
batch_size: 8	warmup_momentum: 0.8
img_size: 640 x 640	warmup_bias_lr: 0.1
rect: false	box: 0.05
resume: false	cls: 0.5
nosave: false	cls_pw: 1.0
notest: false	obj: 1.0
noautoanchor: false	obj_pw: 1.0
evolve: false	iou_t: 0.2
bucket: "	anchor_t: 4.0
cache_images: false	fl_gamma: 0.0

image_weights: false	hsv_h: 0.015
device: ''	hsv_s: 0.7
multi_scale: false	hsv_v: 0.4
single_cls: false	degrees: 0.0
adam: false	translate: 0.1
sync_bn: false	scale: 0.5
local_rank: -1	shear: 0.0
log_imgs: 10	perspective: 0.0
workers: 8	flipud: 0.0
project: runs/train	fliplr: 0.5
name: exp	mosaic: 1.0
exist_ok: false	mixup: 0.0
total_batch_size: 8	
world_size: 1	
global_rank: -1	
save_dir: runs/train/exp5	

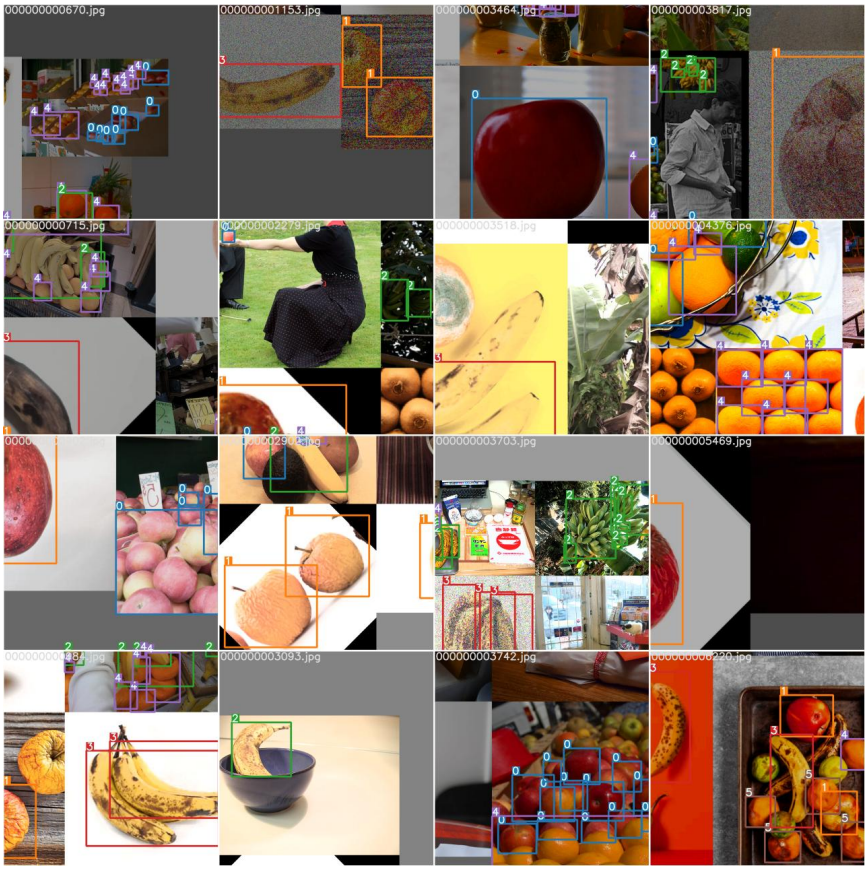
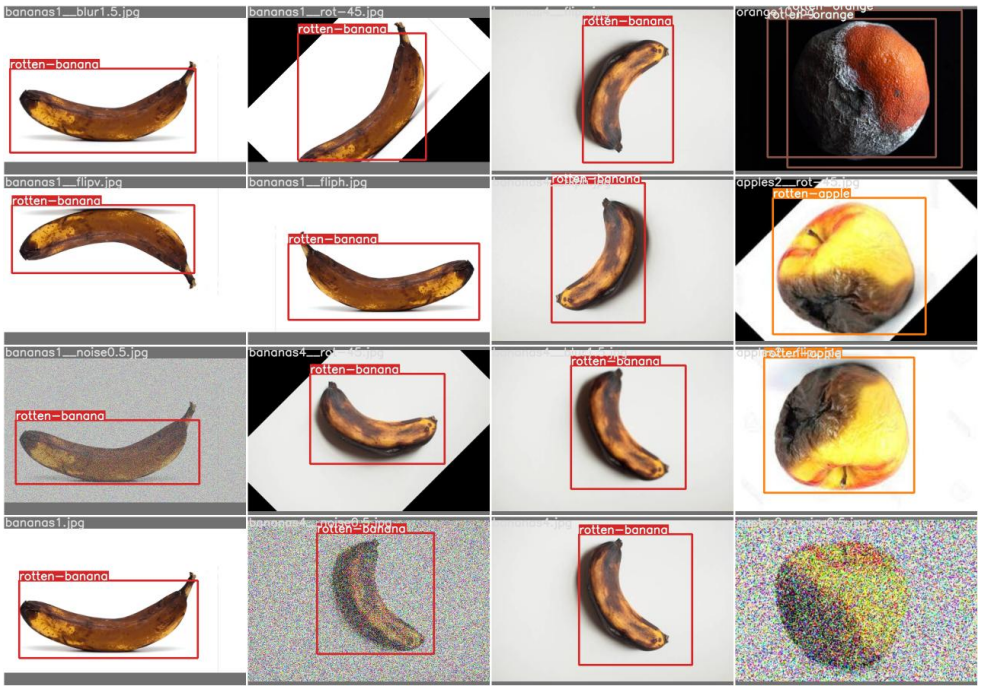
EXP5 Results

0/19	9.66G	0.07998	0.05373	0.04467	0.1784	19	640	0.2072	0.5362	0.2973	0.1314	0.02436	0.01634	0.01794
1/19	9.53G	0.06234	0.04245	0.02186	0.1267	33	640	0.2993	0.5863	0.475	0.2407	0.02262	0.0124	0.01267
2/19	9.53G	0.06058	0.03765	0.01412	0.1123	36	640	0.4446	0.5498	0.4713	0.2417	0.02128	0.0116	0.01157
3/19	9.53G	0.05797	0.03757	0.0126	0.1081	28	640	0.3421	0.5996	0.5234	0.2813	0.02061	0.01106	0.01119
4/19	9.53G	0.05587	0.03557	0.0121	0.1035	22	640	0.4477	0.6501	0.4517	0.2003	0.02779	0.009632	0.01389
5/19	9.53G	0.05194	0.03814	0.01235	0.1024	18	640	0.4661	0.632	0.5449	0.2823	0.01845	0.0111	0.01057
6/19	9.53G	0.04841	0.03847	0.01185	0.09873	36	640	0.4995	0.6953	0.615	0.3772	0.01843	0.01141	0.01164
7/19	9.53G	0.04543	0.03979	0.01064	0.09586	21	640	0.449	0.6303	0.6148	0.3695	0.01764	0.0114	0.01035
8/19	9.53G	0.04506	0.03946	0.01094	0.09545	29	640	0.4768	0.7158	0.6525	0.3934	0.01733	0.01099	0.009259
9/19	9.53G	0.0425	0.03895	0.009518	0.09097	21	640	0.5333	0.6242	0.6035	0.3824	0.01655	0.01091	0.009358
10/19	9.53G	0.0407	0.03711	0.008248	0.08606	19	640	0.4612	0.6802	0.6487	0.4284	0.01516	0.01087	0.01013
11/19	9.53G	0.03884	0.03506	0.006983	0.08088	52	640	0.4738	0.7603	0.6382	0.4221	0.01525	0.01025	0.007811
12/19	9.53G	0.03812	0.03574	0.006115	0.07997	53	640	0.4815	0.781	0.682	0.4561	0.01401	0.01025	0.008395
13/19	9.53G	0.03763	0.03408	0.005803	0.07751	39	640	0.5125	0.7478	0.6797	0.4494	0.01503	0.01046	0.00781
14/19	9.53G	0.03488	0.03314	0.005333	0.07336	30	640	0.5293	0.8078	0.7085	0.4643	0.01477	0.01055	0.006754
15/19	9.53G	0.03412	0.03229	0.004775	0.07119	42	640	0.5625	0.7758	0.6897	0.4606	0.01379	0.01059	0.006486
16/19	9.53G	0.03353	0.03124	0.004556	0.06932	32	640	0.558	0.7984	0.6884	0.4779	0.01371	0.01055	0.006195
17/19	9.53G	0.03242	0.03098	0.004042	0.06745	34	640	0.5808	0.7029	0.6827	0.4592	0.0139	0.01055	0.00736
18/19	9.53G	0.0313	0.03106	0.003727	0.06608	29	640	0.5277	0.728	0.6819	0.4617	0.01427	0.01084	0.008272
19/19	9.53G	0.03062	0.02899	0.003712	0.06332	36	640	0.53	0.7781	0.696	0.4701	0.0137	0.01048	0.007352

EXP5 - Test Batch-0 Predictions



EXP5 - Train Batch-0



EXP5 - Test Batch-0 Labels