THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Final Group Project Report

Chirag Lakhanpal, Tejas Rawal, and Yashwant Bhaidkar
The George Washington University
DATS 6303: Deep Learning
Professor Amir Jafari

Table of Contents

Introduction

Recognizing food from images is an extremely useful tool for a variety of use cases. There are a variety of manual food intake tracker applications available on the market, but how cool would it be if you could receive precise labels and calorie information for the food you are about to eat by simply taking a picture of it. Food tracking can be of medical relevance as well, specifically in the domain of dietary clinical studies where data collection is often inaccurate and difficult to obtain.

Our challenge essentially boiled down to solving a multi-instance segmentation and detection problem. As a group, we chose to explore three different well-known and well-researched neural network architectures in this area to observe their effectiveness at handling the task. This report aims to compose and reflect the journey we each took in tuning and training this dataset.

Dataset

The dataset described is structured in the COCO (Common Objects in Context) data format, which is widely used in computer vision for object detection, segmentation, and image captioning tasks which can be downloaded here. The key components of this dataset are:

- Info: This section appears to be empty in the provided dataset, but typically it contains metadata about the dataset such as version, description, and contributors.

- Categories: This is a list of categories, each with an ID, name, readable name, and a supercategory. For example, in the provided snippet, items like 'water', 'pear', and 'peach' are listed under the 'food' supercategory. Each category is uniquely identified by an ID.

- Images: This section consists of a list of images. Each image is described by its file name, dimensions (height and width), and a unique ID. For instance, an image file named '065537.jpg' with dimensions 464x464 pixels is identified by the ID 65537.

- Annotations: The annotations provide detailed information about objects in the images. Each annotation includes an area (size of the object), bounding box coordinates, category ID (linking to the Categories section), a unique ID, the ID of the image it belongs to, a flag indicating if the object is a single instance or part of a crowd (iscrowd), and segmentation information for precisely outlining the object.

Training Data Set

Set of 54,392 (as RGB images) food images, along with their corresponding 100,256 annotations in MS-COCO format

Test Data Set

Set of 946 (as RGB images) food images, along with their corresponding 1708 annotations in MS-COCO format

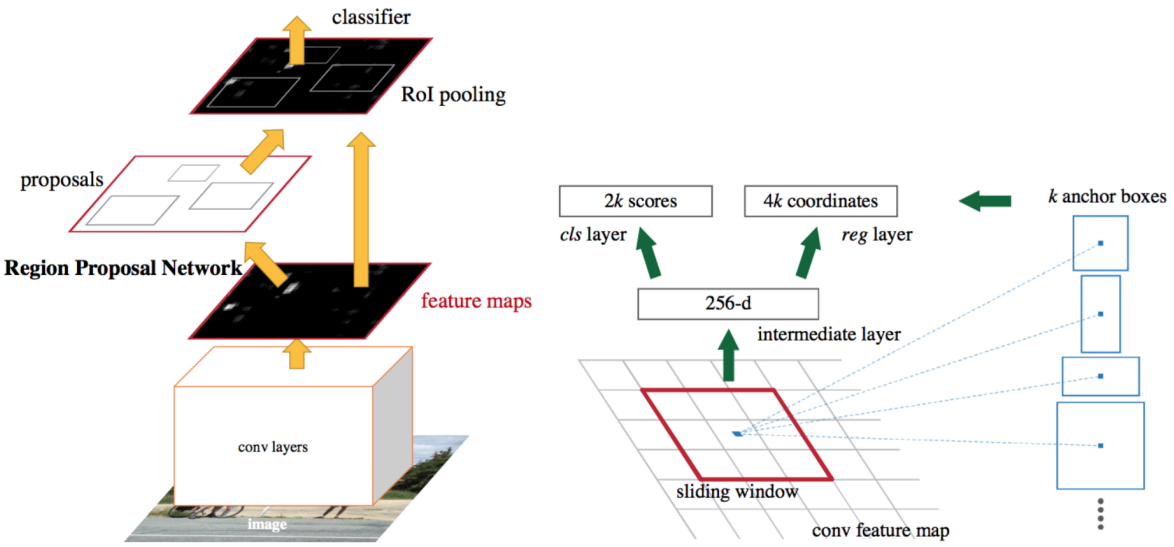
Section	Subsection	Description	Example
info	Various fields	General dataset information, typically metadata.	This Section is empty in the dataset.
categories	id	Unique identifier for each category.	2578 (for water)
	name	Programmatic name of the category in lowercase.	'water'
	name_readable	Human-readable name of the category.	'Water'
	supercategory	Group to which the category belongs.	'food'
images	file_name	Name of the image file.	'065537.jpg'
	height	Height of the image in pixels.	464
	id	Unique identifier for each image.	65537
	width	Width of the image in pixels.	464
annotations	area	The area of the annotated object.	44320
	bbox	Bounding box coordinates for the object.	[86.5, 127.5, 286.0, 170.0]
	category_id	The category identifier for the annotated object.	2578 (for water)
	id	Unique identifier for each annotation.	102434
	image_id	The identifier of the image associated with the annotation.	65537
	iscrowd	Indicates if the annotation contains a single object (0) or multiple objects (1).	0 (single object)
	segmentation	Polygonal segmentation coordinates for the object.	[[235.99999999999997, 372.5, 169.0, 372.5, ..., 368.5, 264.0, 371.5]]

Deep Learning Networks

Faster R-CNN

Faster R-CNN (Ren et al., 2016) is one of the most popular deep learning object detection algorithms and an innovative iteration in the Region-Based Convolutional Neural Network (R-CNN) family of models first introduced by Girshick et al. (2013).

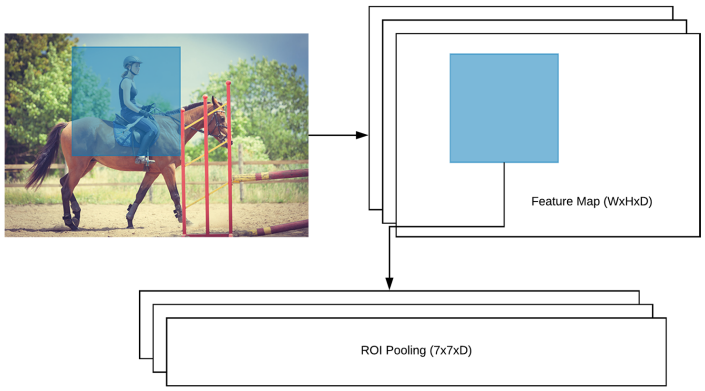
Building on top of the previous iterations of the R-CNN architectures (R-CNN and Fast R-CNN), Faster R-CNN introduced an additional module called the Region Proposal Network (RPN) to generate a set of proposed bounding boxes for an image to determine where in an image a potential object is located.



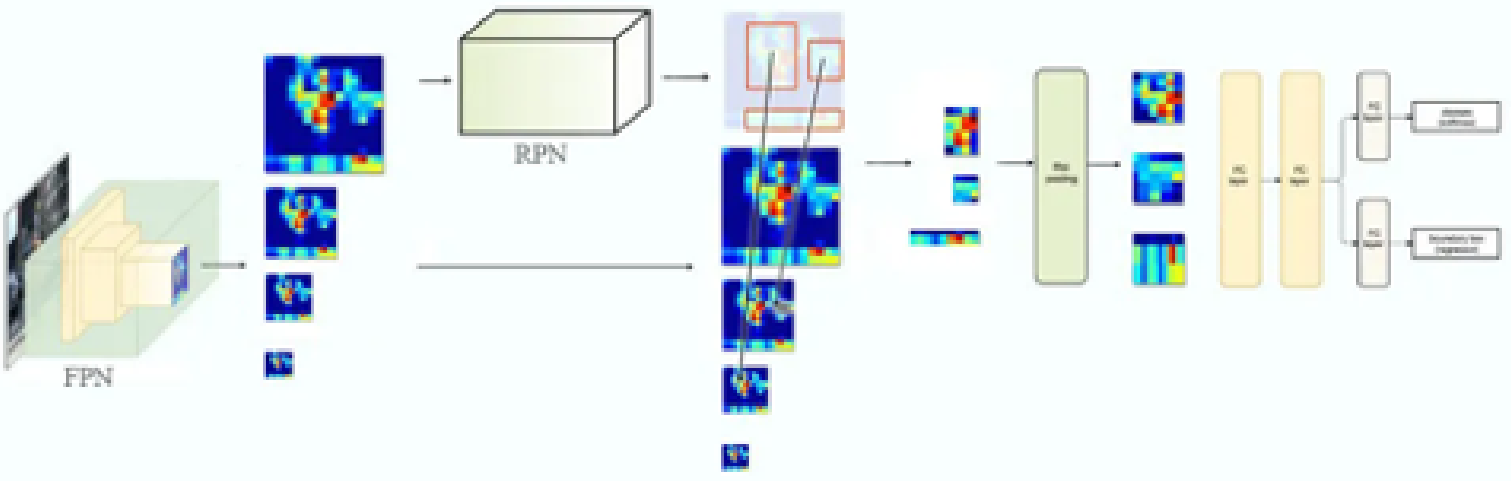
In general, the Faster R-CNN architecture is composed of:

- A backbone CNN layer (usually a pretrained layer like ResNet)
- The RPN
- The Region of Interest (ROI) pooling layer
- Two linear classifier heads, one for bounding box coordinates and one for class labels

Proposals are generated using anchors, which create bounding at different scales across multiple sample points along the image grid. The proposals are then pruned based on their Intersection over Union metric against the ground truth labels and sent to the ROI layer, along with the feature maps extracted by the base CNN.



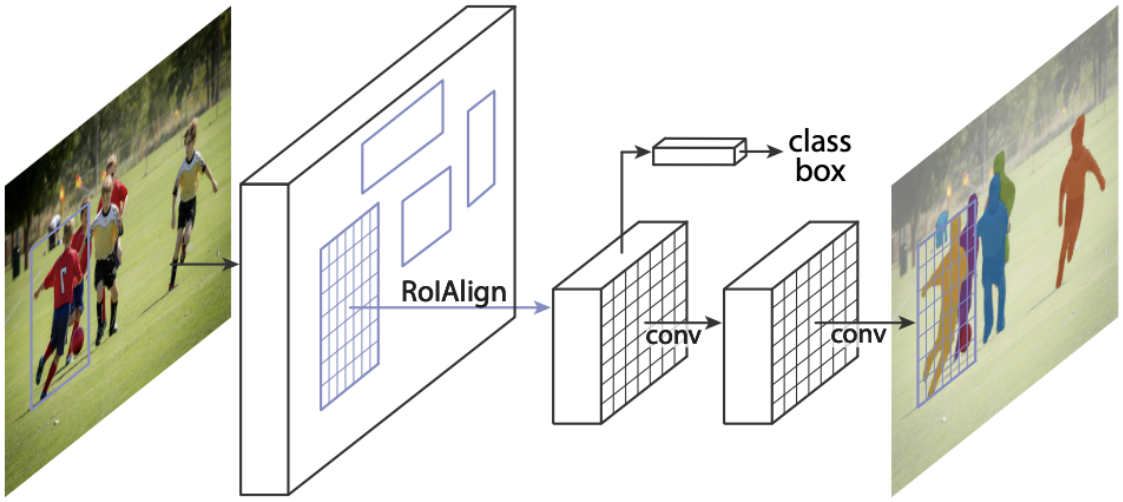
For our implementation, we trained 2 different versions of the pretrained model: fasterrcnn_resnet50_fpn_v2 and fasterrcnn_mobilenet_v3_large_fpn. The difference between these models resides in their choice of backbone CNN, but each includes a modification to the original model architecture - the inclusion of a Feature Pyramid Network (FPN). The FPN module helps by generating multiple feature map layers for a single image using pathways composed of downsampling and upsampling CNN layers.



Mask R-CNN

Two variants of the Mask R-CNN model with different backbones - mask_rcnn_R_50_FPN_3x and mask_rcnn_R_101_FPN_3x were trained for this project. A few additional models were experimented with namely, MMdetection and HTC models. However, due to installation issues, we were unable to make them work, hence, we focused on the Mask R-CNN models, tuning them for the best performance on the given dataset. The average precision (AP) metric was chosen to compare the results of the models.

Working and Architecture



The two models chosen were Mask R-CNN R-50 FPN 3x and Mask R-CNN R-101 FPN 3x. Mask R-CNN is an extension of the Faster R-CNN model, which is designed for object detection. Mask R-CNN adds an additional branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression. The key features of Mask R-CNN are

- Region Proposal Network (RPN): Generates object proposals.
- RoI Align: Accurately extracts features from each proposed region, solving the misalignment issue present in the Faster R-CNN's RoI Pooling.
- Classification and Bounding Box Regression: Determines the class of each object and refines their bounding boxes.
- Segmentation Mask Prediction: Generates a binary mask for each instance in the image.

ResNet-50 & ResNet-101 Backbone

The backbone of a neural network is the initial part that is responsible for extracting features from the input images. Mask R-CNN uses ResNet (Residual Networks) as a backbone. Specifically, ResNet-50 is a variant with 50 layers. It introduces residual learning to ease the training of deep networks. The key idea is to introduce identity shortcut connections that skip one or more layers.

Feature Pyramid Network (FPN)

FPN improves the standard feature extraction pipeline by building a pyramid of features at multiple scales. This is particularly useful for detecting objects of different sizes.

- Multi-Scale Feature Extraction: FPN generates a set of feature maps at different scales, each corresponding to different layers of the ResNet.
- Top-Down Pathway and Lateral Connections: After reaching the top of the network, the FPN constructs higher-resolution features by upscaling spatially coarser but semantically stronger feature maps from higher levels and enhancing them with features from lower levels.

3x Training Schedule

The "3x" in "R-50 FPN 3x" refers to the training schedule. The 3x schedule triples the training time compared to the standard schedule. This allows the model to potentially achieve better performance as it learns from the training data for a longer period.

YOLOv8

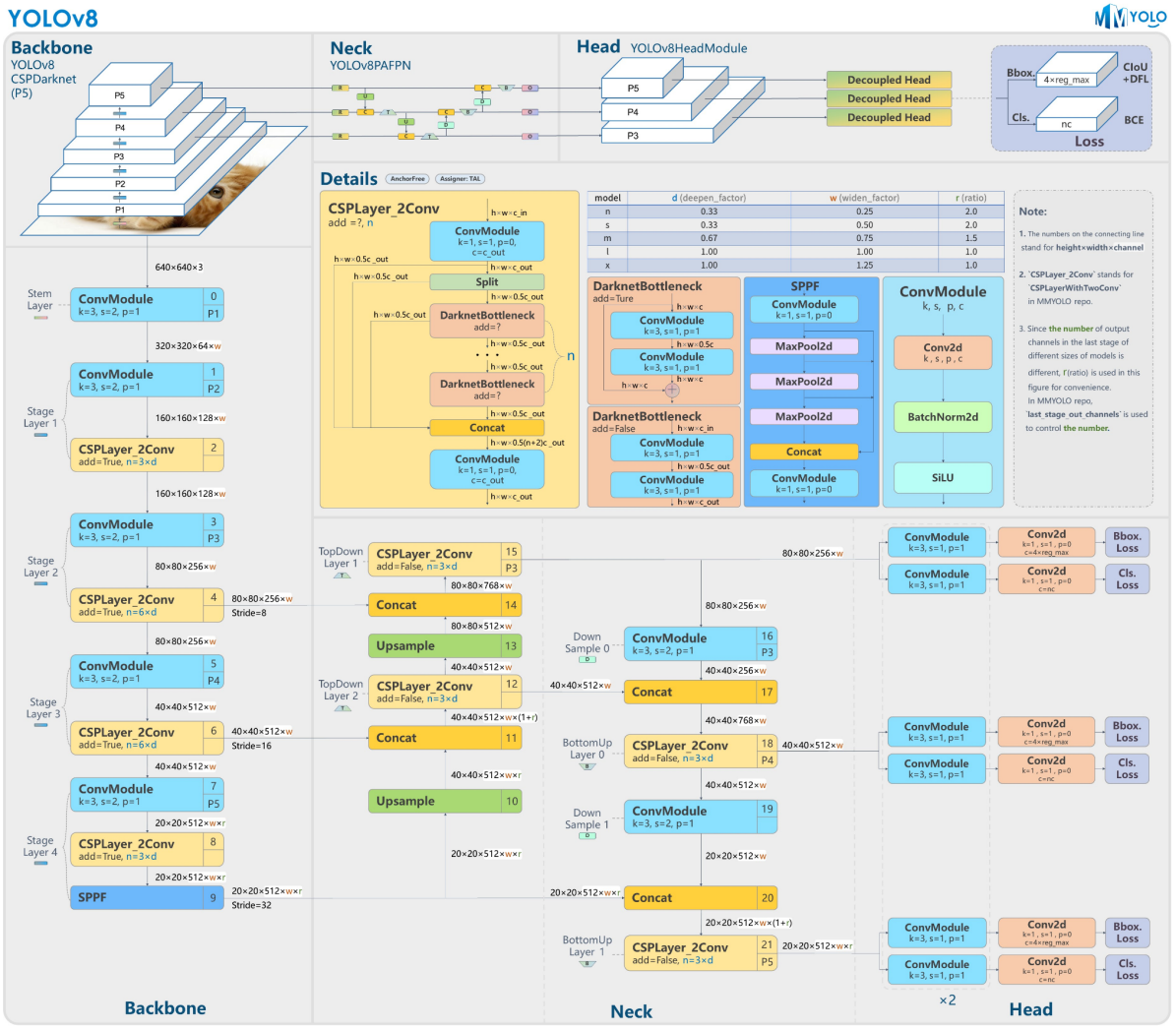
- YOLOv8 is a real time object detection model developed by Ultralytics. It is the 8th version of YOLO and is an improvement over the previous versions in terms of speed, accuracy and efficiency.
- The model is created in PyTorch and can run on both CPU and GPU. YOLOv8 is highly efficient and supports numerous formats such as TF.js or coreML. Like YOLOv7, YOLOv8 can be used for object detection, segmentation and image classification.
- There are few main changes in the YOLOv8 model compared to the previous versions.
- YOLOv8 has a new backbone network which extracts the features from the images.
- A new anchor-free detection head and a new loss function, makes it a perfect choice for a wide range of object detection and image segmentation tasks.

Here are some key features of YOLOv8:

- Advanced Backbone and Neck Architectures: YOLOv8 employs state-of-the-art backbone and neck architectures, resulting in improved feature extraction and object detection performance.
- Anchor-free Split Ultralytics Head: YOLOv8 adopts an anchor-free split Ultralytics head, which contributes to better accuracy and a more efficient detection process compared to anchor-based approaches.
- Optimized Accuracy-Speed Trade Off: With a focus on maintaining an optimal balance between accuracy and speed, YOLOv8 is suitable for real-time object detection tasks in diverse application areas.

- Variety of Pre-trained Models: YOLOv8 offers a range of pre-trained models to cater to various tasks and performance requirements, making it easier to find the right model for your specific use case.

Architecture



(The architecture image is taken from mmlab Github.)

Data Preprocessing and EDA

Data Acquisition

The data was acquired directly from the challenge's website. There were 2 zipped files: one for the training and one for the test dataset. Extracting the files revealed the images folder and the corresponding annotations JSON file. Conveniently, the annotations were provided in the COCO data format. This is a widely-used standard used for evaluating datasets and models in object detection and image segmentation tasks. There are several libraries available for parsing and analyzing COCO annotations which helped with data preparation for training.

Data inspection

Primary inspections of the training images dataset revealed a few images that were not related to food at all. This evaluation process was done manually and was cumbersome, but we were able to extract several outlier images. We did not get the time to explore automating this process but would ultimately like to expand our pipeline to include this cleaning.

As is standard in the COCO format, the annotations JSON could include multiple ground truth labels for a given image. While developing visualizations to describe the data, the annotations had to be remapped so each image's entry would contain all labels, bounding boxes, and segmentations associated with its features.

The COCO format for the bounding box coordinates was also different from what our model architectures required. In COCO, ground truth bounding boxes are in the [Xmin, Ymin, Width, Height] format. During preprocessing, these were converted to the [Xmin, Ymin, Xmax, Ymax] format.

The annotated segmentation mask arrays consisted of polygon coordinates relating to the image's grid space. These would need to be converted into binary masks for consumption by the model during training and inference.

The libraries and frameworks we used provided nice, out-of-the-box functionality for dealing with this data structure.

Exploratory Data Analysis

1. Category Distribution

Findings: The findings from the data visualization indicate that the dataset contains various food categories with differing image counts. The category 'Water' has the highest count, and the category "Veggie Burger" has the lowest, suggesting a potential class imbalance.

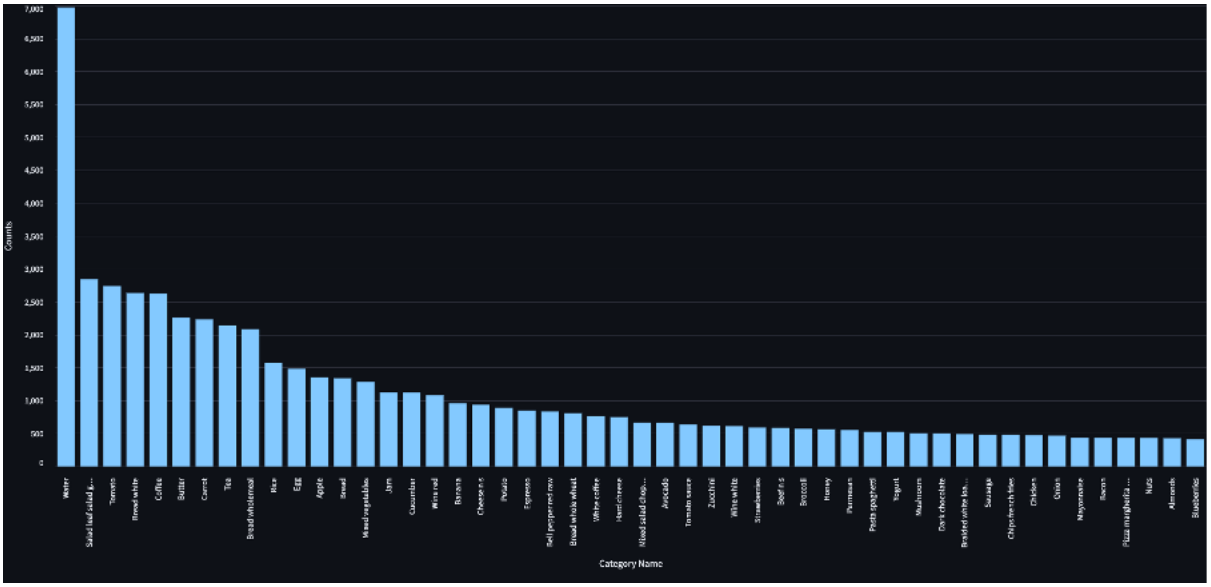


Figure 1: Frequency of Food Categories. The chart shows the number of images for each food category in the dataset, with 'Water' being most common.

2. Distribution of heights and widths of images

Findings: The scatter plot reveals a positive correlation between image width and height, indicating that as images get wider, they also tend to get taller. There are few potential outliers. This is for setting up image sizes as a parameter to the model.

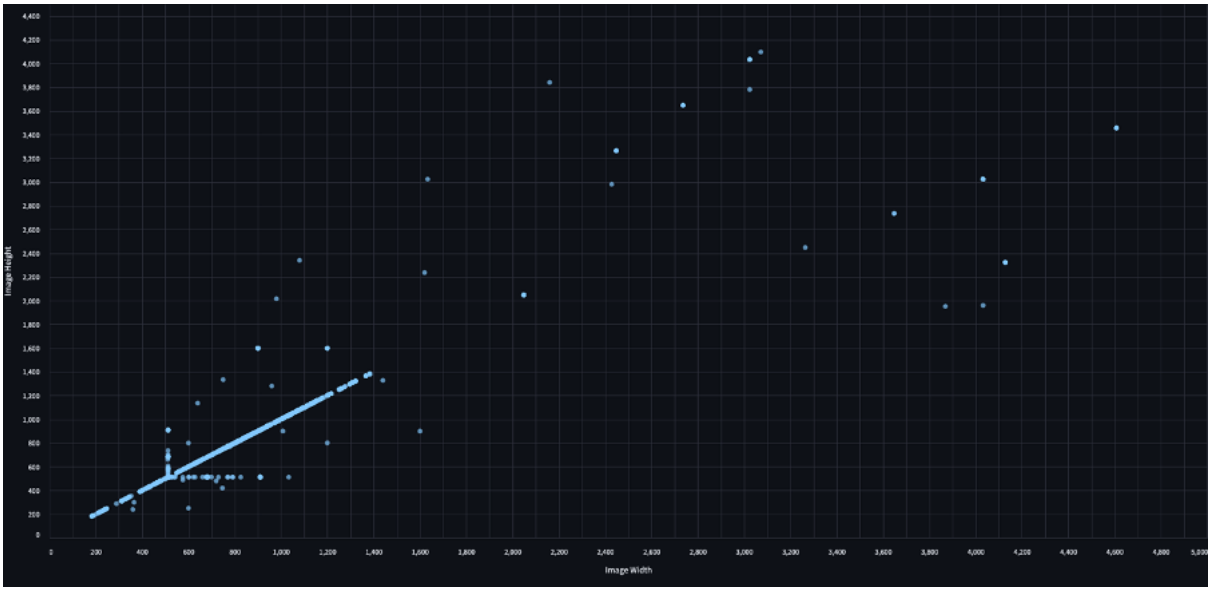


Figure 2: Scatter plot of Image Dimensions. This plot shows the relationship between image width and height in the dataset, with a trend indicating a proportionate increase in height with width.

3. Distribution of heights and widths of images

Findings: The density plot exhibits a sharp peak at an aspect ratio of 1, indicating a large number of square images in the dataset, with fewer rectangular images.

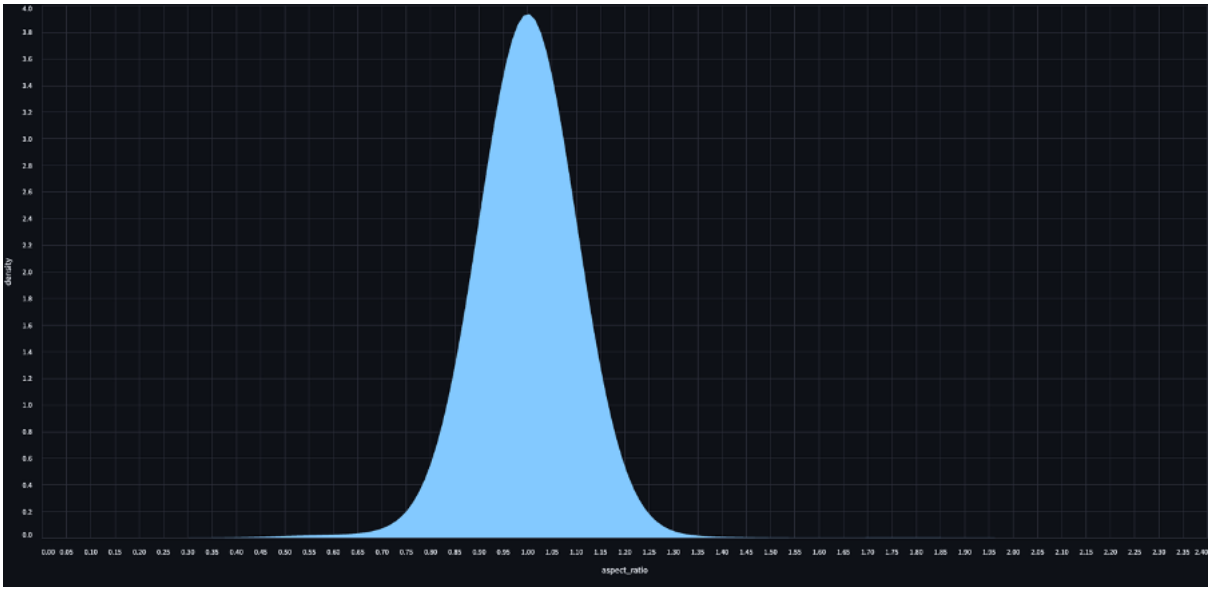


Figure 3: Aspect Ratio Distribution. The density plot showcases the predominance of square images within the dataset.

4. Distribution of annotation counts per image (log scale)

Findings: The logarithmic distribution graph indicates most images have a low number of annotations, with a steep decline as the annotation count per image increases.

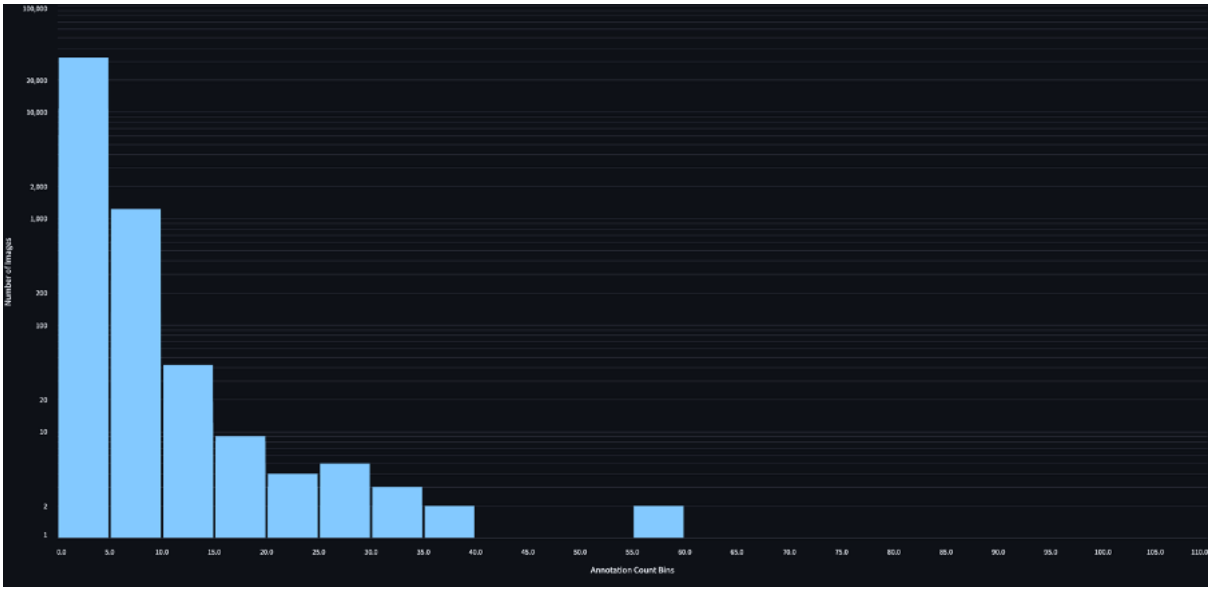


Figure 4: Logarithmic Distribution of Annotations per Image. The graph shows the skewed distribution towards images with fewer annotations.

5. Distribution of area of annotations per category

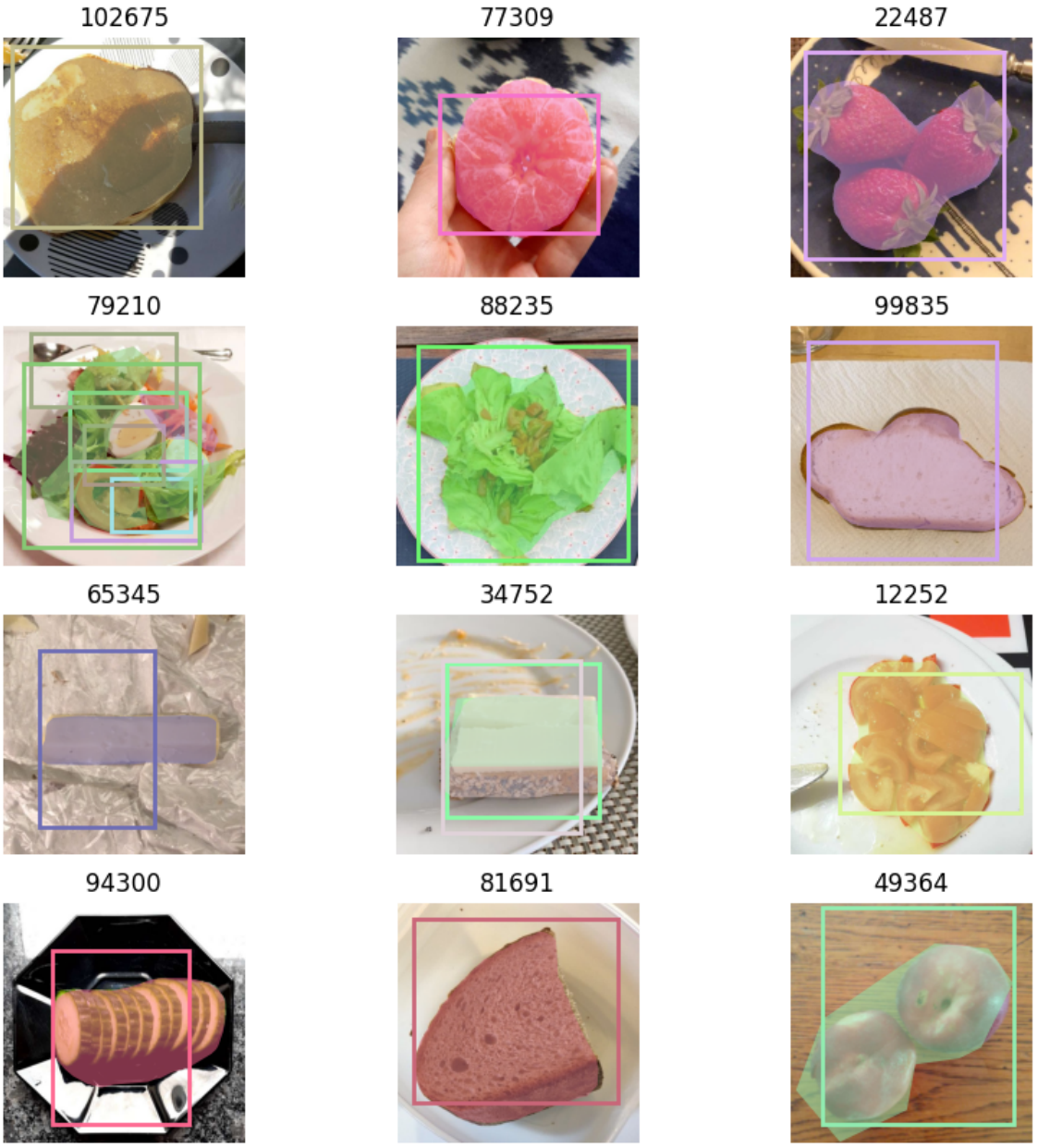
Findings: The treemap depicts the proportion of annotation regions across several dietary categories, with 'Water' taking up the most space, indicating its widespread presence in the dataset.

Figure 5: Treemap of Annotation Areas by Category. This visualization represents the relative size of annotations for each food category, highlighting the dominance of certain categories like 'Water' and 'Salad Leaf Salad Green'.

6. Annotated Images

We plotted the bounding boxes on the images and it inferred that some of the bounding boxes are accurate but there are some of them which are not even enclosing the objects or they are not properly enclosing the object.

Below are some images from the dataset after plotting the bounding boxes:



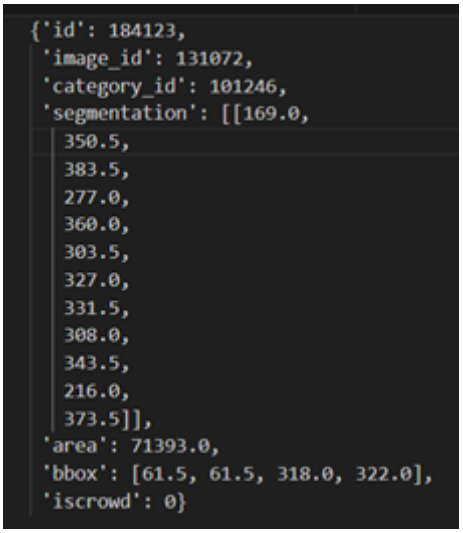
Data Preprocessing

We ensure the correct image dimensions and adjust bounding boxes from XYWH to XYXY format. Rotated annotations and non-food items are removed to focus the dataset on relevant images.The below table summarizes all the necessary steps taken to preprocess the data. The processed data is saved in new JSON annotation files.

Issue Identified	Solution Implemented
Misaligned segmentation and bounding boxes	Adjusted bounding boxes based on segmentation data
Incorrect image dimensions	Updated image dimensions using actual image files
Rotated images	Removed rotated annotations from the dataset
Non-food item annotations	Excluded annotations not related to food items
Small bounding boxes	Small bounding boxes retained to assess model performance (Did not improve the model)
Rotated annotations	Removed annotations with a rotated bounding box mode
Images with aspect ratio > 2	Choose to remove them

In COCO annotations, we have Xmin, Ymin , width and height of the images. For YOLO, we need the annotation in normalized Xcenter, Ycenter, width and height format.

For that, we have developed the script that creates the folder structure and stores the labels in .txt format in the required folder structure.

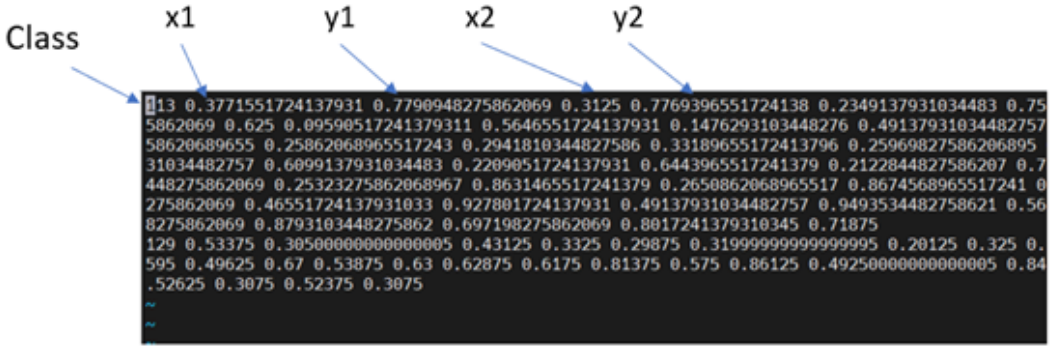


Using JSON coordinates, we developed a script which converts the coordinates into JSON

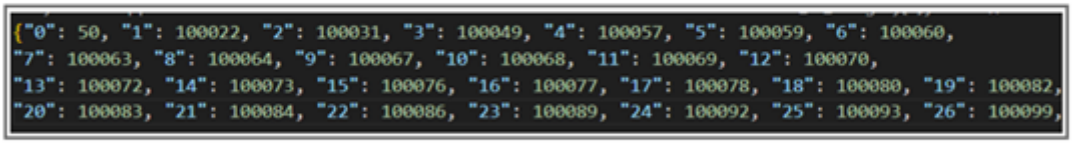
Normalized bounding box details(Yolo Format)

For segmentation, we have X and y coordinates of all the points. We have normalized those coordinates using the respective image width and height.

Normalized segmentation coordinates(Yolo Format)



Category to class mapping



Models

Faster R-CNN

The two variations of the model were both constructed using the PyTorch framework, specifically the torchvision library. A custom dataset was created for processing each image and related annotations. For images, this included transformations to resize, randomly flip, randomly rotate, and normalize the pixel values. Due to memory constraints, the images were scaled down to 256x256 pixels and a minibatch size of 16 was used.

For loss optimization, we used the Stochastic Gradient Descent optimizer with various parameter combinations to help address overfitting. These included momentum, weight decay, and gradient clipping. The learning rate was connected to a scheduler that reduced its value by a factor of 10 every 5 epochs.

For quick iterations, while experimenting, there were only 5 epochs run. Once an optimal set of parameters were determined, the training epochs were increased. There were 20 epochs of training for the MobileNet-based model and 10 for the ResNet-based model. Because the ResNet model was quite large in terms of depth and parameters, it took much longer (and more memory) to converge.

During inference, a detection threshold was used to filter out bounding box predictions with a weak score. The threshold was set at 0.5.

Masked R-CNN

Network Implementation

1. Framework Choice: The project will be implemented using FastAI and Detectron2 frameworks.
2. Data Loading and Augmentation: Data loaders are created with custom transformations to augment the dataset. Transformations include rotation, zoom, lighting adjustments, and random erasing. The batch size for the data loader is set to 9 for efficient memory usage.

Model Architecture

1. The Mask R-CNN architecture with ResNet-50 backbone from the COCO-InstanceSegmentation model zoo is selected. The model is configured with Detectron2 to accommodate the specifics of our dataset, like the number of classes.

Training Parameters

1. Mini-batches: Mini-batches are used with a batch size of 8. Batch size is determined based on memory constraints and the complexity of the model.
2. Learning Rate: A base learning rate of 0.00025 is set. Learning rate adjustments will be made using the "WarmupMultiStepLR" scheduler.

Training Iterations:

1. The model will train for 50,000 iterations.
2. Checkpointing is enabled every 20,000 iterations.

Performance Evaluation

1. Validation Metrics: Performance is assessed using the mean Average Precision (mAP) on the validation dataset. A COCO evaluator is employed for detailed evaluation.

2. Overfitting Prevention: Regular checkpoints and monitoring of validation loss will help in detecting overfitting. Data augmentation and a diverse dataset help in generalizing the model.
3. Extrapolation Detection: Rigorous testing on a separate set not seen during training.Regular validation checks during training.

Post-Training Analysis

1. Threshold Adjustment: Post-training, the detection threshold is set to 0.1 to filter out low-confidence predictions.
2. Category Mapping: A JSON file mapping class IDs to category IDs is created for interpretability. This mapping aids in understanding the model's predictions in terms of actual food categories.
3. Model Deployment: The trained model is saved as 'model_final.pth'.It can be loaded for inference on new images using the configuration and weight files.

YOLOv8

For YOLOv8 object detection, we have to create the structure of the dataset and it goes in the following way.

```
├─ yolov8
## └─ train
####└─ images (folder including all training images)
####└─ labels (folder including all training labels)
## └─ test
####└─ images (folder including all testing images)
####└─ labels (folder including all testing labels)
## └─ valid
####└─ images (folder including all testing images)
####└─ labels (folder including all testing labels)
```

(The above structure is taken from medium)

Below is the screenshot of the actual .yaml file:

```
train: ../food/images/train/
val: ../food/images/val/
nc: 323
names:
  ['beetroot-steamed-without-addition-of-salt', 'green_bean_steamed_without_addition_of_salt', 'watermelon_fresh',
```

We have to pass the train, val data details path and nc is the number of classes.

In names, we have to pass all the names of classes based on their mapping in list format.

Train file:

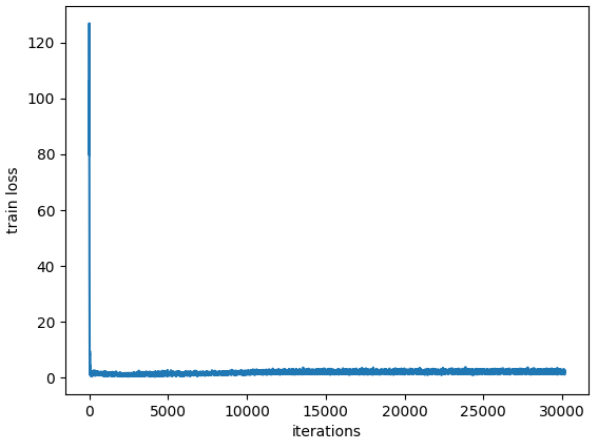
- While training, I have passed the pre-trained model for YOLOv8 Segmentation as we have some of the boxes that are not at the right position.
- For training, I have imported the Ultralytics library and used Yolov8 from it.

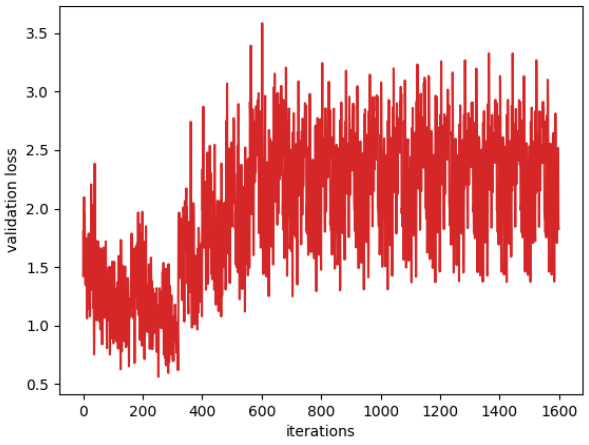
The training procedure is as follows:

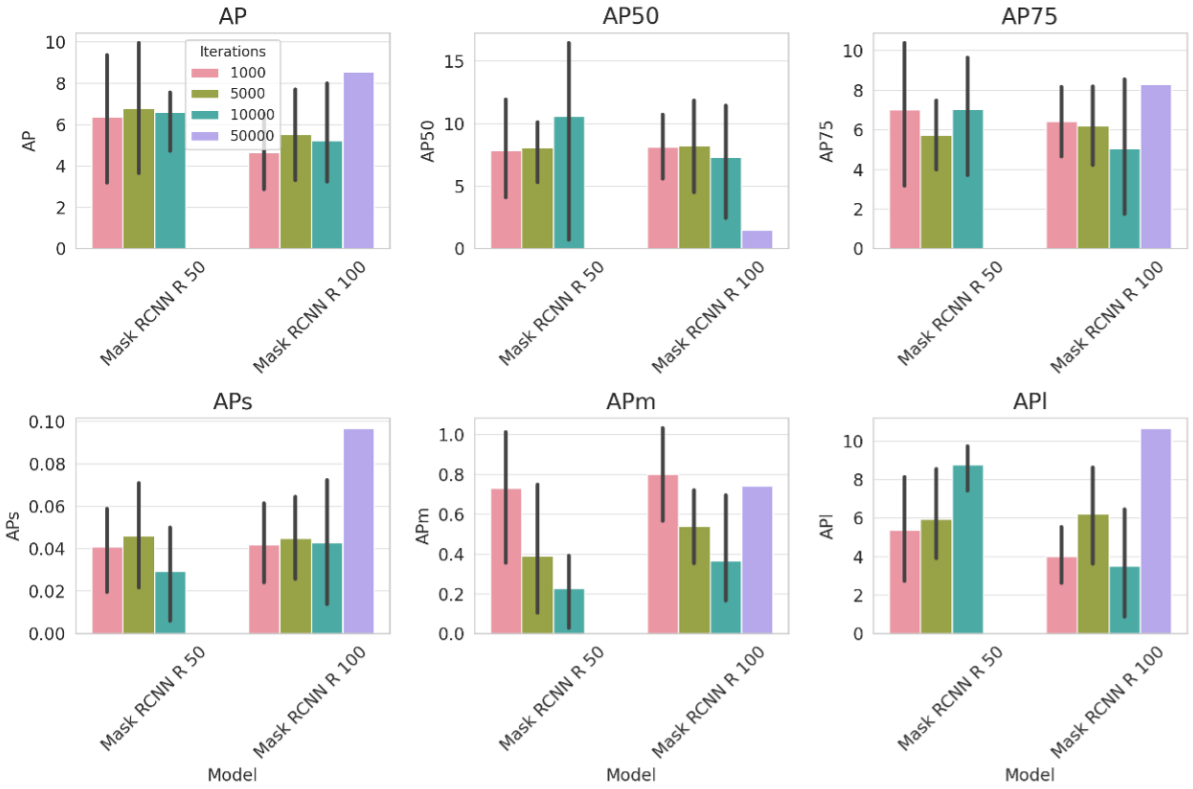
- So, we first loaded the pre-trained yolov8 medium model and then using the train method, we can pass the remaining parameters.
- For data, we passed the path of the .yaml file which contains the information about the entire data hierarchy and class information.
- We can set the rest of the parameters to tune the model. worker = 8 will use all cores from the GPU.
- Image size is one of the important parameters. All standard results published for YOLO models are with image size 640X640.

Results and Summary

For the Faster R-CNN set of models, we were unable to obtain any viable evaluation results to compare its effectiveness against the other trained models. We are aware that there is an issue with the training pipeline but weren't able to figure out the root cause yet. As you can see in the loss plots below, there is some inconsistent behavior as the number of iterations increases.







The comparison between two models, r50 and r100, was conducted across various performance metrics, as depicted in the provided figure. These metrics include Average Precision (AP), AP at 50% Intersection over Union (AP50), AP at 75% Intersection over Union (AP75), AP small (APs), AP medium (APm), and AP large (APl). The models were evaluated at different iterations: 1,000; 5,000; 10,000; 50,000; 100,000 to understand the impact of training duration on model performance.

YOLOv8 Model(pre-trained):

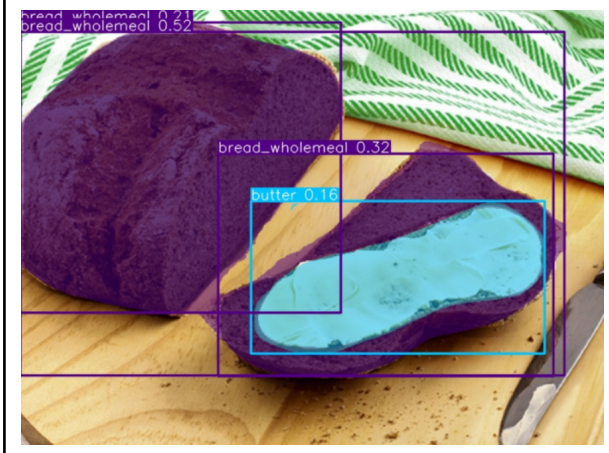
Below are the results we got on the validation dataset.

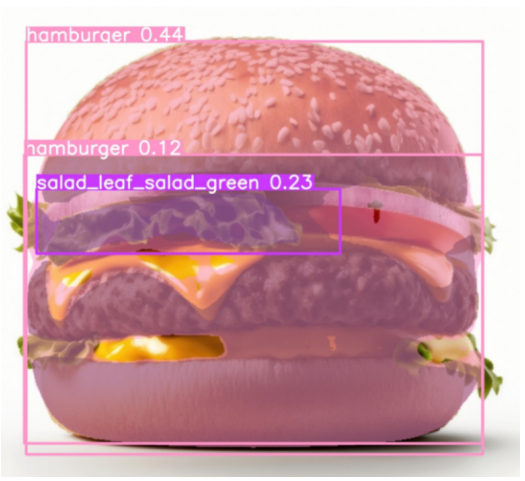
Classes	Images	Instances	Box(p)	R	mAP50	mAP50-95
323	946	1576	0.57	0.48	0.53	0.44

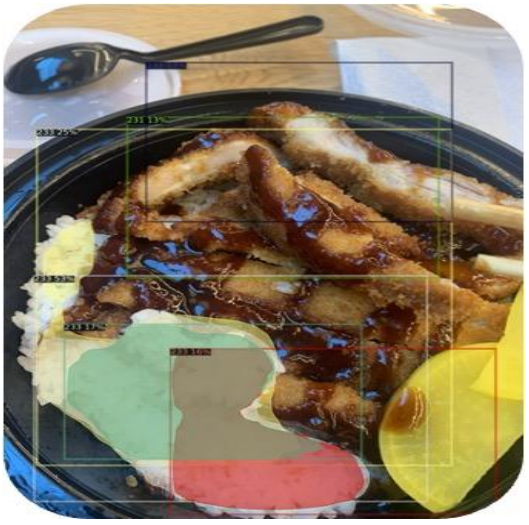
As per the results:

- mAP50 value is 0.53 which is good and indicates that the model is doing sensible prediction.
- Recall is 0.48 which is (true positive)/(true positive + false negative) based on instances

Below are the results of object detection on Validation images:







Below are the results of object detection on real images(sample images taken from www.google.com)


pasta_spaghetti 0.67
egg 0.37
egg 0.35
salad_leaf_salad_green 0.21
egg 0.24
cucumber 0.15
tomato 0.45
rice 0.57
curry_vegetarian 0.16

Conclusions

Multi-instance segmentation and object detection poses a difficult challenge when trying to build a generalized model. Obtaining and processing data can be tricky, and slight variations can have large negative effects on model performance. Therefore, extra consideration needs to be taken when preparing and processing the dataset.

In food object detection, we can easily detect separated food items, but in the case of complex food dishes it is difficult to detect overlapping objects. To make the models more robust, we need to train using more complex data that ranges in label and bounding box distribution.

The YOLOv8 model netted the best results. It even detected some of the mixed dishes with good confidence.

Some future considerations for our project:

- Data imbalance was present in the provided training dataset but not necessarily handled in the training pipeline. For a more comprehensive approach, we could use weighted sampling techniques to address this issue.
- We could take into account depth factors such as food item area to produce food item volume predictions and gather more accurate calorie information.

References

1. Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Liu, Y., Pham, H., Dong, X., Luong, T., Hsieh, C., Lu, Y., & Le, Q. V. (2023). Symbolic Discovery of Optimization Algorithms. ArXiv. https://arxiv.org/abs/2302.06675

2. Ren, S., He, K., Girshick, R., & Sun, J. (2016, January 6). Faster R-CNN: Towards real-time object detection with region proposal networks. arXiv.org. https://arxiv.org/abs/1506.01497

3. El Vaigh, C. B., Garcia, N., Renoust, B., Chu, C., Nakashima, Y., & Nagahara, H. (2021). GCNBoost: Artwork Classification by Label Propagation through a Knowledge Graph. arXiv preprint arXiv:2105.11852. Retrieved from https://arxiv.org/abs/2105.11852

4. Maheshwari, E., & Cao, M. Y. (Year). Building a Neural Historical Art Classifier with GLCM Texture Features and Attention Feature Extraction. Department of Computer Science, Stanford University. Retrieved from http://cs231n.stanford.edu/reports/2022/pdfs/80.pdf

5. Martinez, H. (2023, November 13). Faster R-CNNs. PyImageSearch. https://pyimagesearch.com/2023/11/13/faster-r-cnns/

6. ChatGPT by OpenAI

7. PyTorch. (n.d.). Torchvision object detection finetuning tutorial. TorchVision Object Detection Finetuning Tutorial. https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

8. https://blog.roboflow.com/whats-new-in-yolov8/

9. https://medium.com/@beyzaakyildiz/what-is-yolov8-how-to-use-it-b3807d13c5ce

10. https://stackoverflow.com/questions/68398965/coco-json-annotation-to-yolo-txt-format

11. https://medium.com/cord-tech/yolov8-for-object-detection-explained-practical-example-23920f77f66a

Image References

Papers with Code. (2020). Screenshot of method illustration. Retrieved from https://production-media.paperswithcode.com/methods/Screen_Shot_2020-05-23_at_7.44.34_PM.png

Screenshot for the YOLO file structure
https://medium.com/@beyzaakyildiz/what-is-yolov8-how-to-use-it-b3807d13c5ce

Yolo Architecture
https://github.com/open-mmlab/mmyolo/blob/dev/configs/yolov8/README.md/

Faster R-CNN Architecture
https://lilianweng.github.io/posts/2017-12-31-object-recognition-part-3/#faster-r-cnn

FPN Network
https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c

ROI pooling
https://pyimagesearch.com/2023/11/13/faster-r-cnns/