THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Final Project Report

Yashwant Bhaidkar
The George Washington University
DATS 6303: Deep Learning
Professor: Amir Jafari

Table of Contents

Introduction

Project overview

Calorie tracking is one of the important aspects of the proper diet and it is very difficult task to measure the each and every component of the food. In this task, we are developing the model that identifies food items in images and estimates their calorie content using deep learning techniques. This project will enhance the dietary tracking for personal and medical purposes, simplifying food intake monitoring through image analysis. It is an object detection task which will identify the food item from the image and provides the name of the item.

Outline of work

As we are group of 3 members, we divided the initial processing equally and we explored the model architecture individually.

For model deployment, we developed the predict functions individually and combined the code in final application.

Below are the tasks as project contribution:

- **Data inspection**
- **Exploratory data analysis**
- **Annotation conversion from COCO to YOLO**
- **YOLOv8 Training and model optimization**
- **Predict function – YOLOv8 for Streamlit application**

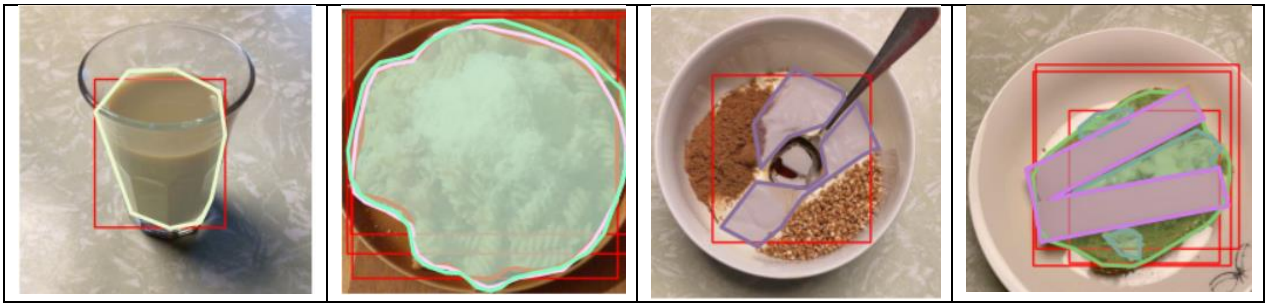
Data inspection

As a primary inspection, I explored the data manually and there were few images which are not related to the food. So, we removed those images as an outlier. Our task is related to object detection, so there is no way to inspect the data through script. There are very few images which are not related to the food.

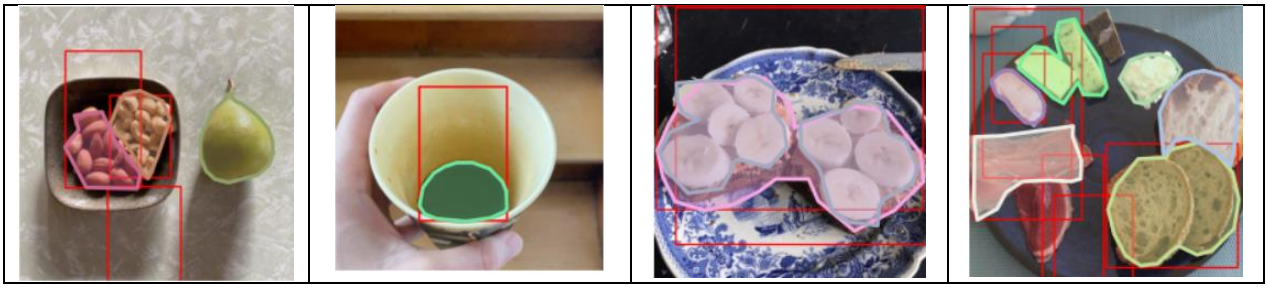
EDA for bounding boxes:

I 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.

Some of the proper bounding boxes samples. below are some images from the dataset after plotting the bounding boxes :



Some of the bounding boxes are not correctly enclosing the object:



There are some images where bounding box width and height is greater than the actual image width and height. We deleted those images while training the model as the normalized annotation coordinates are going above 1 after the normalization.

Data preprocessing:

COCO to YOLO Annotations:

As we have coco annotations, to apply YOLO on it, we need the annotation in the txt format.

There are two possible tasks we can perform using the available annotation:

1)Object detection

2)Instance segmentation

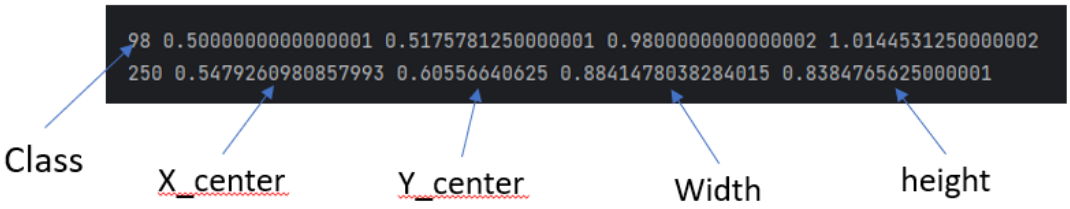
In object detection, we have Xmin, Ymin, width and height of the images. For YOLO, we need the annotation in normalized Xcenter, Ycenter, width and height format.

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

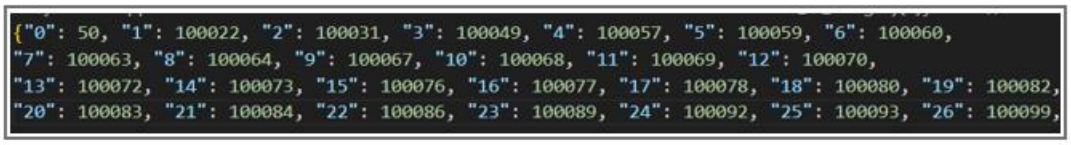
JSON file structure


```
{'id': 184123,
 'image_id': 131072,
 'category_id': 101246,
 'segmentation': [[169.0,
   350.5,
   383.5,
   277.0,
   360.0,
   303.5,
   327.0,
   331.5,
   308.0,
   343.5,
   216.0,
   373.5]],
 'area': 71393.0,
 'bbox': [61.5, 61.5, 318.0, 322.0],
 'iscrowd': 0}
```

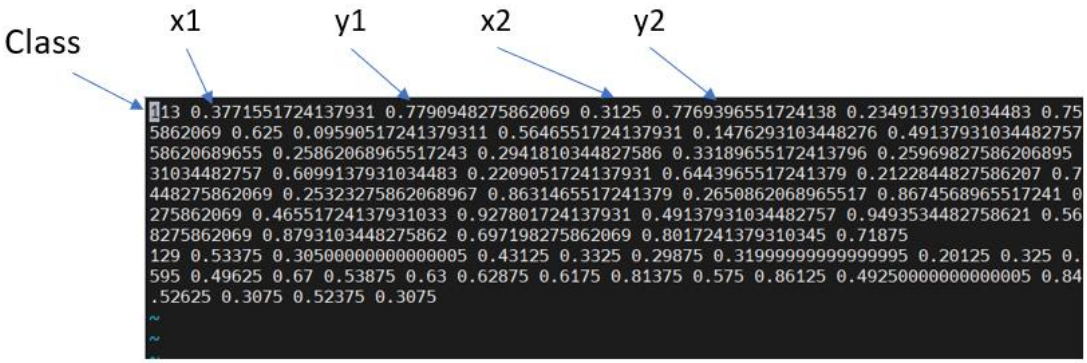
Normalized bounding box details(Yolo Format)



Category to class mapping



Normalized segmentation coordinates(Yolo Format)



YOLOv8 – Details

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.

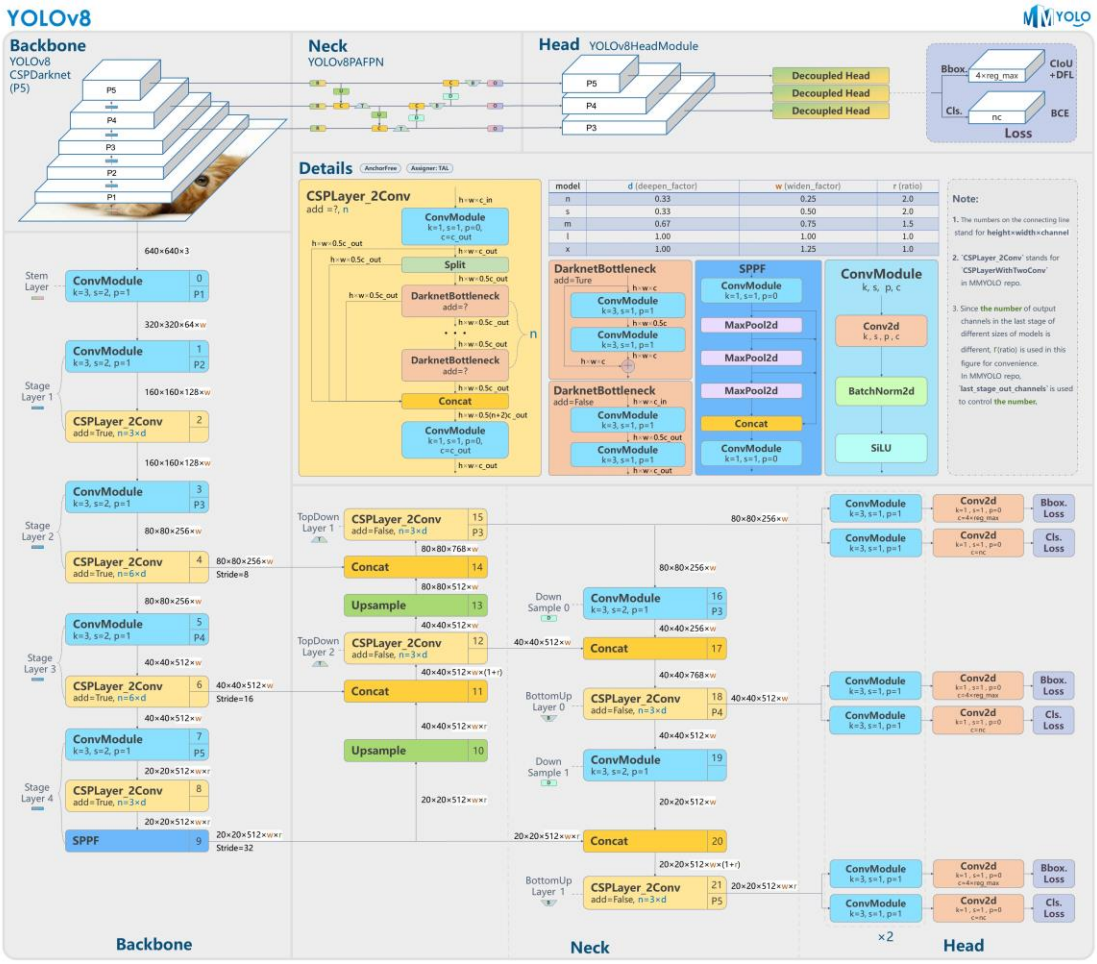
The model is created in PyTorch and can run on both CPU and GPU. YOLOv8 is highly efficient and supports numerous formats such as TF.js or core ML. Like YOLOv7, YOLOv8 can be used for object detection, segmentation and image classification.

There are few main changes in YOLOv8 model compared to the previous versions

YOLOv8 has a new backbone network which extract the features from the images.

A new anchor-free detection head and a new loss function, makes it a perfect choice for wide range of object detection and image segmentation tasks.

YOLOv8 Architecture



Architecture image is taken from mmlab Github. It is based on the official code of YOLOv8

Details: In YOLOv8, the backbone network fetches the features from the images and feeds it to the neck and head detects the object.

There are some complex blocks used in this architecture. Some of them are:

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 Tradeoff:** 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.

Implementation:

Yolo file structure:

For YOLOv8 object detection, we have to create the structure of the dataset and it goes in 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)
```

(above structure is taken from medium)

.yaml config file for execution details:

path: (dataset directory path)

train: (Complete path to dataset test folder)

test: (Complete path to dataset test folder)

val: (Complete path to dataset test folder)

#Classes

nc: 2 #replace according to your number of classes

#classes names

#replace all class names list with your classes names

names: [all Classes in list format]

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 pretrained model for YOLOv8 Segmentation as we have some of the boxes 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:

```
##with yolov8m
model = YOLO('yolov8m-seg.pt')


# Training.
results = model.train(
   data='/home/ubuntu/term_project/data/food.yaml',
   imgsz=640,
   epochs=1,
   batch=40,
   save=True,
   pretrained = True,
   name='yolov8m_segmentation',
   plots = True,
   device = 0,
   workers = 8
   )
```

So, we first loaded the pre trained yolov8 medium model and then using the train method, we can pass the remaining parameter.

For data, we passed the path of .yaml file which contains the information about the entire data hierarchy and class information.

We can set rest of the parameter to tune the model. worker = 8 will use the all cores from the GPU.

Image size is one of the important parameter. The all standard results published for YOLO models are with image size 640X640.

Results:

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

Predict function for streamlit App:

For streamlit app, I have developed the function which is accepting the user image input and detect the food objects from that image and shows them with bounding boxes.

Here is the main method used in the predict function:

```
from ultralytics import YOLO
model = YOLO("best.pt")
model.predict(source = '/home/ubuntu/term_project/food/images/train/131145.jpg',
show = True,save = True,hide_labels = True, hide_conf = True,conf = 0.1,
save_txt = True,save_crop = False,line_thickness = 2)
```

Once the model is ready, we will get the best.pt file.

Using YOLO object, we can load the model.

Using predict method, we can pass the required parameters and it will detect the objects from the provided image.

Result using YOLOv8 model:

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)

Summary:

YOLOv8 is one of the best object detection models and we applied the pre trained YOLO model to customized it for the food object detection. Using the same, we got around 0.53 of mAP score which is a sensible detection.

In the food object detection, it is clear that we can easily detect the separated food items but in case of complex food dishes, it will be difficult to detect the overlapping object and the confidence of detection will be low.

With YOLOv8, we got the best results so far and it detected some of the mix dishes really well.

Code percentage:

Code taken from internet: approx.170 lines

Modified lines of code: approx.100 lines

Code written by me: approx. 150 lines

percentage of the code found or copied from the internet is: 20%

references

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

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

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

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

https://github.com/ultralytics/ultralytics