Wellness Wingman

Syncing Food Insights with Fitness Goals

By Chirag Lakhanpal, Tejas Rawal and Yashwant Bhaidkar

Table of contents

01 Introduction

Introduction

Overview

- **Goal:** Enable accurate food intake tracking through image recognition.
- **Significance:** Shift from traditional, imprecise food frequency questionnaires to precise image-based analysis.
- **Advancement:** Utilizes deep learning to address the complexities in recognizing diverse food items.

Problem Statement

- **Objective:** Train models to accurately identify individual food items in images, using a novel dataset.

Key Features

- **Annotations:** Includes segmentation, classification (mapping to Swiss Food items), and weight/volume estimation.
- **Dynamic Nature:** Continuously expands with new data, enhancing model accuracy and adaptability.

The Challenge

- **Core Challenge:** Accurate detection of a wide range of food items in diverse, real-world images.
- **Dataset Complexity:** Employs a comprehensive dataset from COCO, featuring various food items in real-life settings with detailed annotations for advanced model training.

02 Project Scope

Data Description & Features

Data Structure

- We have the images and the annotations for the bounding boxes as well as coordinates to segment the actual objects in each image.

Dataset components

- Training set: 54392 food images with instances.
- Validation set: 946 food images with 1576 instances.
- Debug test set: same images as the validation set for testing purpose

Data preprocessing

- Bounding box Issues: There are some of the bounding boxes which are not properly enclosing the targeted objects. We removed those images from the training.

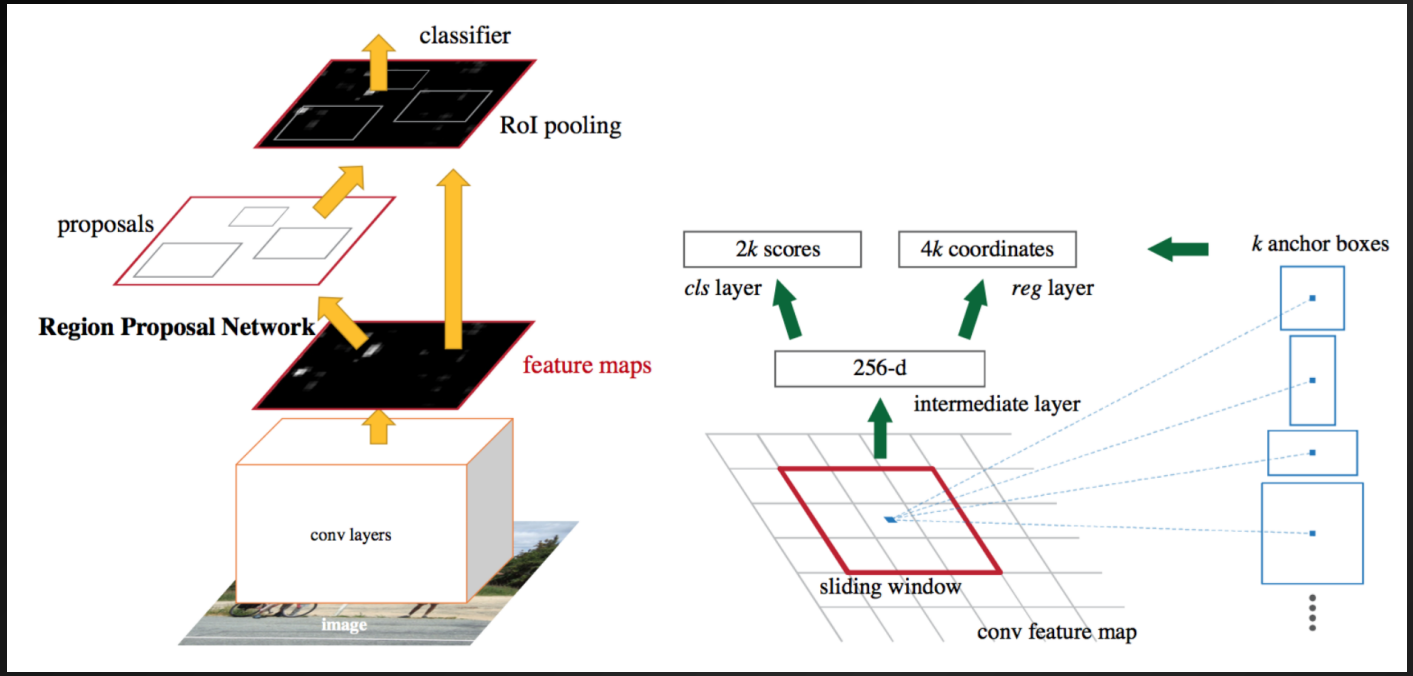
Section	Subsection	Description	Example
info	Various fields	General dataset information, typically metadata.	This Section is empty in the dataset.
categories	id	Unique identifier for each category.	2578 (for water)
	name	Programmatic name of the category in lowercase.	'water'
	name_readable	Human-readable 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]]

03 Architecture

Faster R-CNN

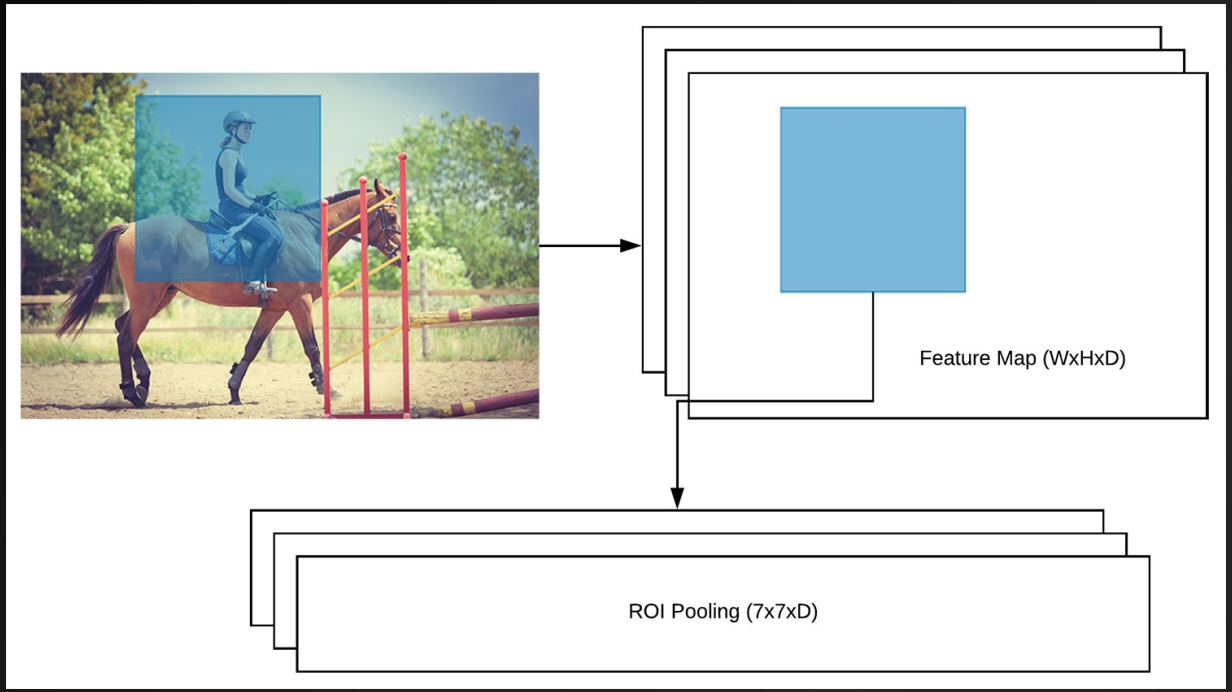
- One of the most popular deep learning-based object detection algorithms is the family of R-CNN algorithms, originally introduced by Girshick et al. (2013).
- Fast R-CNN (Girshick et al. (2015)) improved upon the original algorithm by unifying three independent models into one jointly trained framework and increasing shared computation results
- Faster R-CNN (Ren et al., 2016) is an intuitive speedup solution to Fast R-CNN that integrates integrate the region proposal algorithm into the CNN model

Faster R-CNN: Architecture



- Pre-trained CNN (base network) extract features from the input image
- Features are sent in parallel to two different components:
- Region Proposal Network (RPN): determines *where* a potential object could be in an image
- The final is the Region-Based Convolutional Neural Network (R-CNN)
- 2 fully-connected heads: 1 for bounding box predictions and 1 for class predictions

Faster R-CNN: Workflow



- Pre-train a CNN network on image classification tasks.
- Fine-tune the RPN (region proposal network) end-to-end for the region proposal task, which is initialized by the pre-train image classifier.
- Slide a small n x n spatial window over the conv feature map of the entire image.
- At the center of each sliding window, we predict multiple regions of various scales and ratios simultaneously.
- An anchor is a combination of (sliding window center, scale, ratio). For example, 3 scales + 3 ratios => k=9 anchors at each sliding position.
- Train a Fast R-CNN object detection model using the proposals generated by the current RPN
- Faster R-CNN implements a multi-task loss function that combines the losses of classification and bounding box regression

Mask R-CNN R-50 FPN 3x

Overview of Mask R-CNN:

A state-of-the-art framework for object instance segmentation that extends Faster R-CNN by adding a branch for predicting segmentation masks.

The Significance of R-50 FPN 3x:

- R-50 refers to the use of a ResNet-50 backbone, a 50-layer deep network known for its efficiency and effectiveness in feature extraction.
- FPN stands for Feature Pyramid Network, which enhances the feature extraction capability by integrating low-resolution, semantically strong features with high-resolution, semantically weak features.
- 3x indicates an extended training schedule, tripling the number of training iterations to improve model accuracy.

Applications

Widely used in various fields such as autonomous driving, medical image analysis, and object tracking in videos due to its exceptional ability in precise object localization and segmentation.

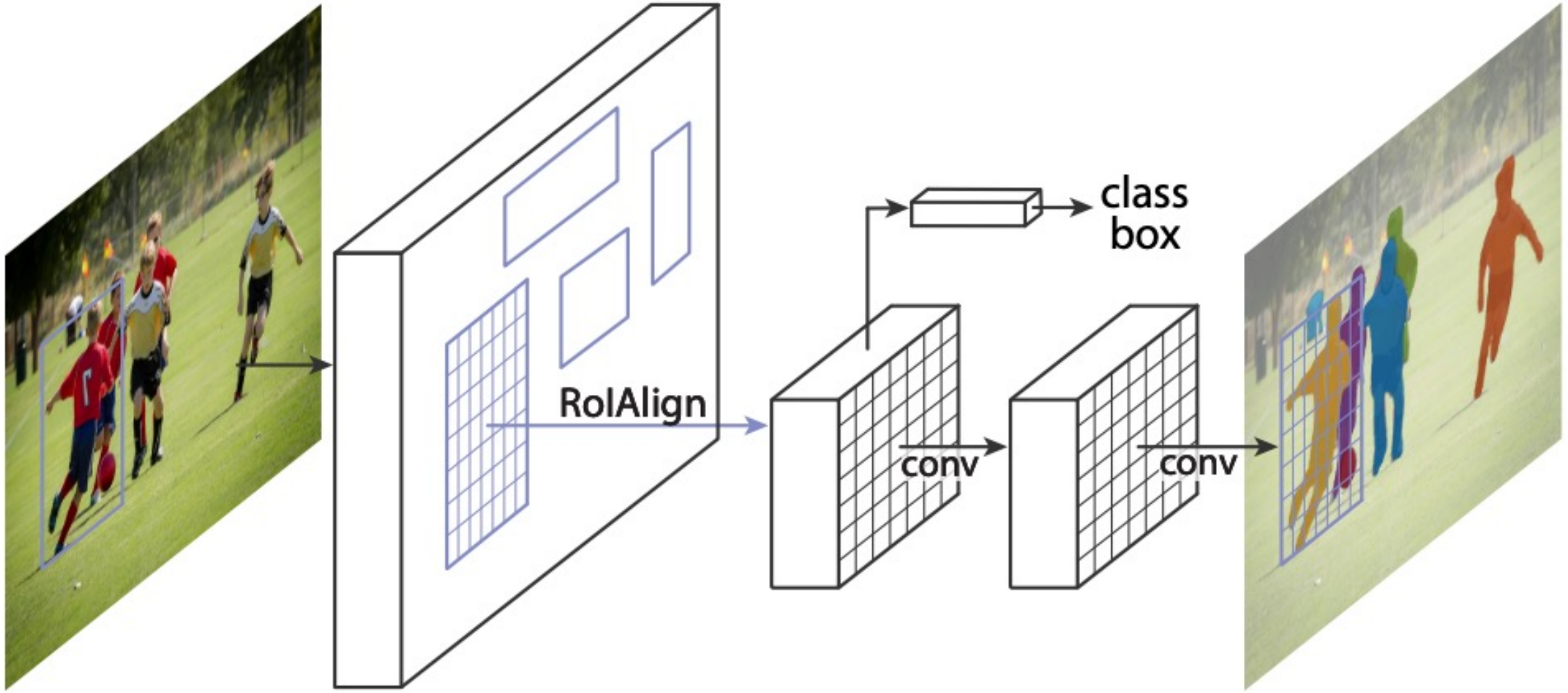
Model Architecture

Core Components

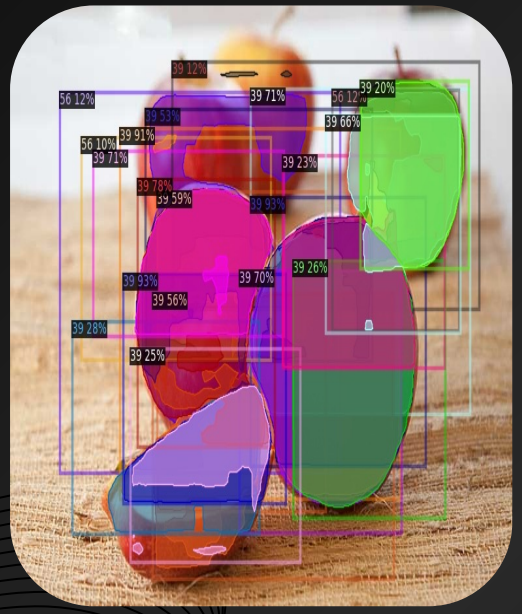
- ResNet-50 Backbone: Extracts a rich set of features from the input image. Known for "skip connections" that help in alleviating the vanishing gradient problem in deep networks.
- Feature Pyramid Network: A key innovation that builds a top-down architecture with lateral connections, allowing the network to leverage multi-scale features.

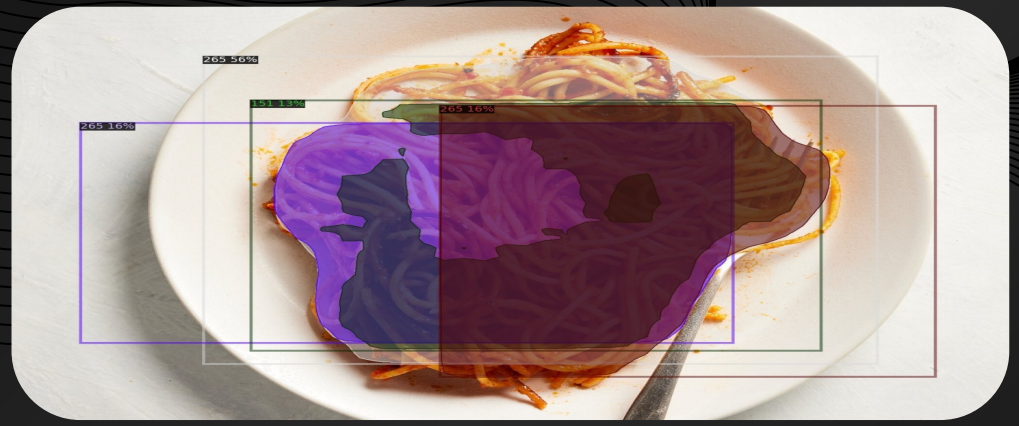
Workflow

- The image is processed through the ResNet-50 backbone, generating a feature map.
- FPN refines these feature maps at various scales, ensuring robust feature extraction across different object sizes.
- The refined features are then used for two parallel tasks: object classification/detection (via bounding boxes) and instance segmentation (creating a pixel-wise mask for each instance).


RoIAlign
class
box
conv
conv

Performance Metrics





Metric	Highest Score
AP	10.583
AP50	16.585
AP75	11.470
APs	0.000
APm	2.242
APl	11.246

YOLOV8 Overview

- 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.

- YOLOv8 has a new backbone network, 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.

Coco to Yolo Annotations

JSON file


```
{'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)


```
98 0.5000000000000001 0.5175781250000001 0.9800000000000002 1.0144531250000002
250 0.5479260980857993 0.60556640625 0.8841478038284015 0.8384765625000001
```


Class X_center Y_center Width height

Normalized segmentation coordinates(Yolo Format)

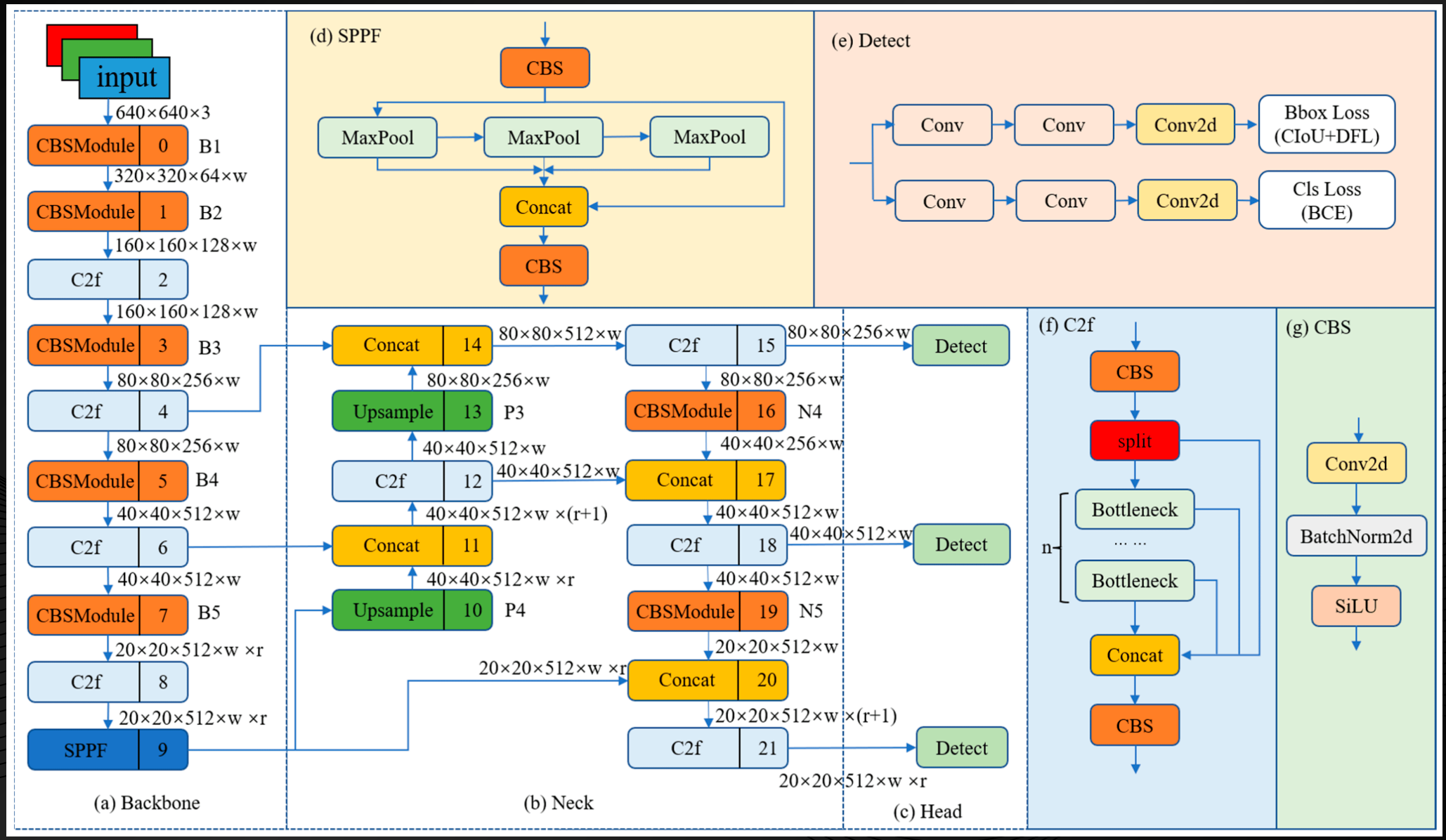
Class x1 y1 x2 y2


```
113 0.3771551724137931 0.7790948275862069 0.3125 0.7769396551724138 0.2349137931034483 0.75
5862069 0.625 0.09590517241379311 0.5646551724137931 0.1476293103448276 0.49137931034482757
58620689655 0.25862068965517243 0.2941810344827586 0.33189655172413796 0.25969827586206895
31034482757 0.6099137931034483 0.2209051724137931 0.6443965517241379 0.2122844827586207 0.7
448275862069 0.25323275862068967 0.8631465517241379 0.2650862068965517 0.8674568965517241 0
275862069 0.46551724137931033 0.927801724137931 0.49137931034482757 0.9493534482758621 0.56
8275862069 0.8793103448275862 0.697198275862069 0.8017241379310345 0.71875
129 0.53375 0.30500000000000005 0.43125 0.3325 0.29875 0.31999999999999995 0.20125 0.325 0.
595 0.49625 0.67 0.53875 0.63 0.62875 0.6175 0.81375 0.575 0.86125 0.49250000000000005 0.84
.52625 0.3075 0.52375 0.3075
~
~
~
```


Category to class mapping

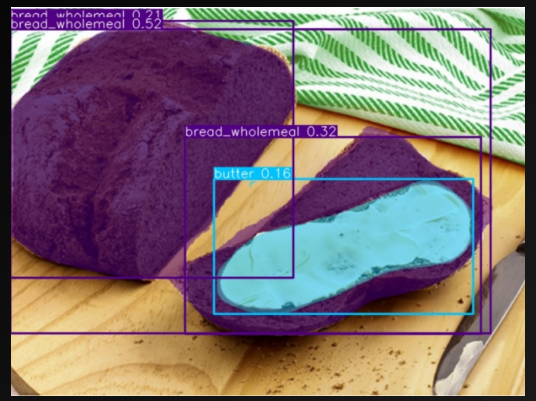

```
{"0": 50, "1": 100022, "2": 100031, "3": 100049, "4": 100057, "5": 100059, "6": 100060,
"7": 100063, "8": 100064, "9": 100067, "10": 100068, "11": 100069, "12": 100070,
"13": 100072, "14": 100073, "15": 100076, "16": 100077, "17": 100078, "18": 100080, "19": 100082,
"20": 100083, "21": 100084, "22": 100086, "23": 100089, "24": 100092, "25": 100093, "26": 100099,
```

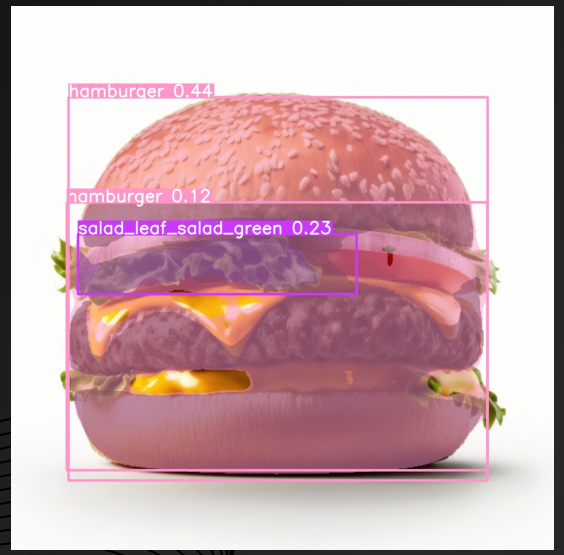
YoloV8– Architecture



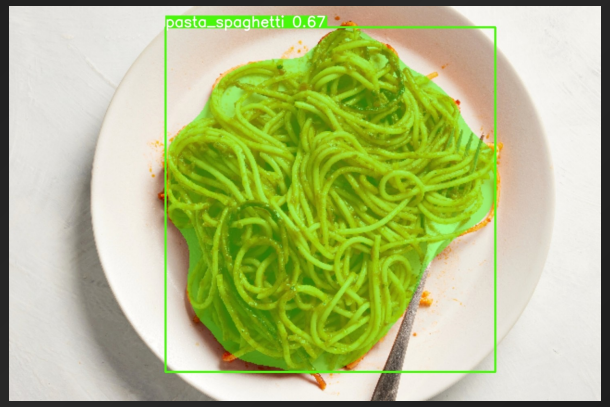
Yolov8 instance segmentation – prediction Sample

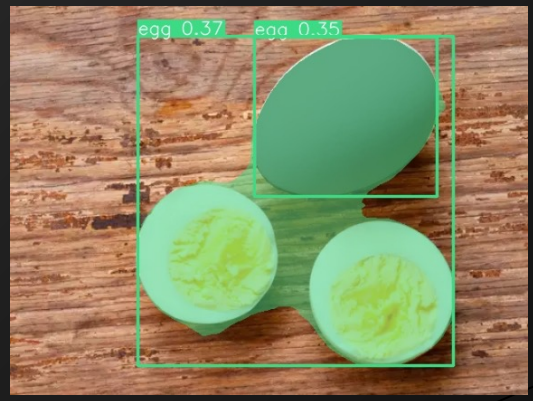
Prediction on Validation images





Prediction on real world images





Performance on validation dataset

mAP50	mAP50-95	R	Box(p)
0.532	0.444	0.487	0.577

04 Dashboard

Explore The App!

Thank you!