

ARI3129 - Advanced Computer Vision for AI

By Kyle Demicoli and Nicholas Vella

Introduction

In recent years, the field of computer vision has witnessed significant progress, propelled by the development and utilisation of deep learning models in various applications. This paper will delve into the meticulous evaluation and comparison of pre-trained models within the Keras Applications API [1], dataset development and object detector training.

The first task involves exploring three trained architectures VGG16 [2], VGG19 [3] and ResNet50 [4] with ImageNet weights. The goal is to understand how these models perform in classifying pizza images with their complexities. This initial investigation helps us gain insights into the strengths and limitations of each model guiding us towards exploration.

The second task involves creating a curated dataset specifically designed for training custom object detectors. Alongside ImageNet pizza images we incorporate sources. Apply meticulous annotation and augmentation processes to enhance the datasets quality. This task focuses on considerations such as planning, format, configuration and content choices that lay the groundwork for developing object detection models.

The crux of the project unfolds in task three which involves training and evaluating two object detection methods specifically designed for identifying pizza and its toppings. Using TensorFlow PyTorch the models are carefully examined, resulting in a comparison of their performance. The paper also incorporates analytics to reveal insights into the detection process, such as listing the toppings and other relevant information obtained from analysing the images.

Background on the techniques used

Task 1:

Keras Applications API [1]:

The Keras Applications API [1] from the Keras deep learning library provides a set of pre-trained models for diverse computer vision tasks. It makes it easier to apply cutting-edge neural networks, allowing users to benefit from advanced models with little effort. Keras Applications allows you to apply transfer learning on specific applications using models that have been pre-trained on large datasets such as ImageNet.

VGG16 [2] and VGG19 [3]:

VGG16 [2] and VGG19 [3] were introduced by the Visual Geometry Group (VGG) from Oxford University to be known for its simplicity in design. VGG16 and VGG 19 contain 16 layers of weight each, having a small receptive field (3x3). These models are characterized by their simplicity and performance in image classification assignment. VGG architectures played an important role in gaining insights into deep convolutional neural networks and have found popularity among the researchers of computer vision.

ResNet50 [4]:

The ResNet50 [4], from the family of Residual Network (ResNet), solves this problem- training very deep networks. ResNet50 as introduced by Microsoft Research utilises skip-type connections for residual learning to train very deep neural networks. The residual connections solve the issue of vanishing gradient, making it possible for one to train networks with hundreds of layers. The 50-layer ResNet is the standard for a wide range of computer vision applications because it works well and can easily be trained.

These models found in Keras Applications API act as strong instruments for different goals including image recognition, object detection and feature extraction etc. Researchers and practitioners often treat them as a starting point for building customised deep learning models.

VGG16, VGG19 and ResNet50 were used in Task 1 for evaluation and comparison.

Task 3:

YOLO:

The YOLO architecture [7] is a popular object detection model, well-known for its efficiency and speed. YOLO stands for (You only look once) which accurately describes how this algorithm functions. Traditional object detection frameworks typically require more than one step in the process: first, the image must be scanned multiple times to identify different regions of interest. Each region must then be classified to ascertain whether it contains an object of interest. This method may require a lot of time and computational resources.

By only examining the image once, YOLO, on the other hand, streamlines this procedure. Here's how it functions:

- YOLO processes the entire image using a single neural network. This network creates a grid of cells out of the image and simultaneously predicts bounding boxes and probabilities for each cell.
- The algorithm predicts every bounding box and its class probability in a single pass. This differs from other approaches that might need multiple passes in order to identify objects.

- YOLO is faster than conventional two-step models by nature because it examines the entire image during training and testing. It is thus especially well-suited for tasks involving the real-time detection of objects.
- By examining the entire image at once, YOLO considers the image's global context in addition to the local areas that are taken into account for object detection. Accurate object identification can be aided by this, particularly in complex scenes.

A Closer Look:

YOLOv5:

YOLOv5 was released on the 6th of January 2020. It combines the ideas of YOLO with CNN, emphasising accuracy, speed, and simplicity. YOLOv5, which is renowned for its real-time processing ability, provides a balance between accuracy and speed. Because it is based on the PyTorch framework, it is simple to use and deploy. Its object detection capabilities are almost at the cutting edge.

YOLOv7:

YOLOv7 was released on July 6th, 2022. CNN is combined with YOLO techniques in YOLOv7, just like in YOLOv5. Its goal is to increase speed and accuracy over the previous models. When it comes to object detection tasks, YOLOv7 is an excellent choice because it can achieve higher throughput on specific hardware than previous YOLO versions.

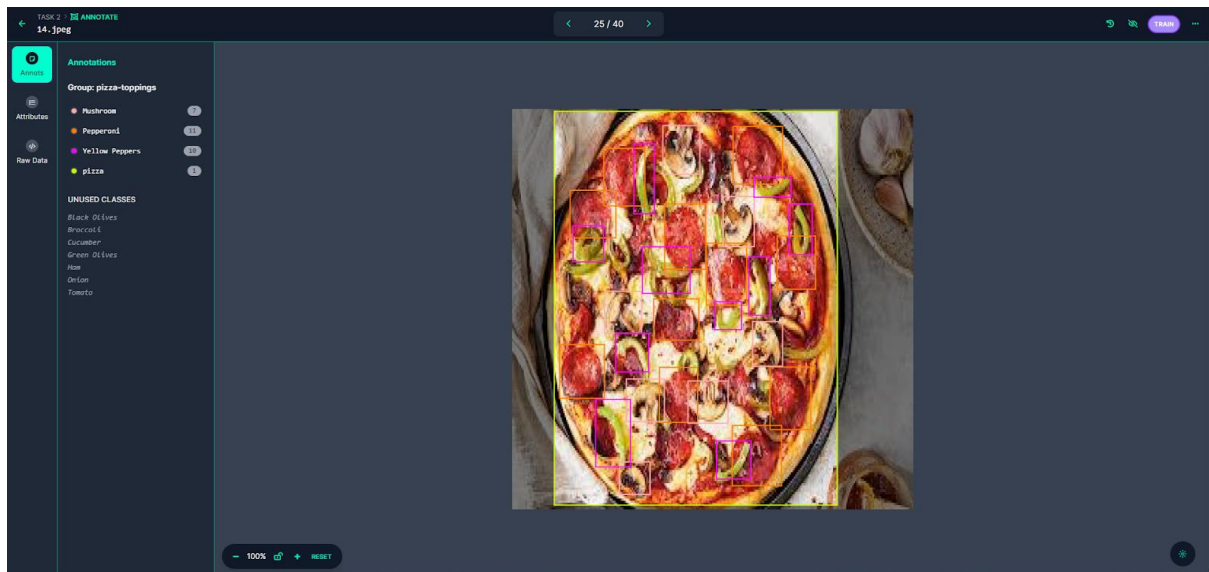
YOLOv8:

Lastly, YOLOv8 was introduced on the 10th of January 2023. In terms of the architecture used, YOLOv8 does away with the need for manually defined anchor boxes by introducing an anchor-free design. In addition to making training easier, this improves the model's capacity to identify objects with different sizes and aspect ratios. Additionally, it includes multi-scale prediction, which enables the model to successfully represent objects at various scales. Thanks to architectural improvements, YOLOv8 has proven to be faster and more accurate than both YOLOv5 and YOLOv7. Because of its optimization for embedded platforms, such as the RTX 4070 Ti and NVIDIA Jetson AGX Orin, performance is dependable and effective.

Data Preparation

A significant stage in the design of strong and effective object detection models is curation and preparation for this dataset. In this project, an inclusive dataset comprising various pizza images was compiled from a GitHub repository [5]. This selection was motivated by the fact that so many pizza composition and toppings are included in this dataset. The need for diversity in the dataset was considered vital to a comprehensive assessment of object detection models because they are designed to function effectively in different real-life situations.

The Roboflow [6] platform helped orchestrate the annotation and organisation of the dataset. While well-known for its convenient interface and robust annotation functions, Roboflow [6] aided in the normalisation of these images into specific classes based on diverse pizza toppings. Pictorially, the annotated classes were Black Olives, Broccoli, Cucumber, Green Olives, Ham, Mushroom, Onion, Pepperoni, Tomato, Yellow Peppers and Pizza.



Annotation Process

Subsequently, the dataset underwent a judicious partitioning into three subsets: training, testing, and validation. The arrangement was deliberately set at 81%, 11%, and 8% to ensure enough data for the model learning while also leaving other sets distinct for rigorous testing and validation.

During the preprocessing stage in Roboflow [6], a series of data augmentation techniques was carefully used to strengthen the model resilience and generalisation capacity. Each training example was subjected to a triad of horizontal and vertical flips, minimal and maximal zoom crops, and rotations within a range of -15° to $+15^{\circ}$. Additionally, adjustments to hue and exposure, ranging from -25° to $+25^{\circ}$ and -10% to $+10\%$, respectively, were introduced.

These augmentations were carefully tailored together to introduce variation into the dataset, mimicking different opinions and lighting conditions that models could see in practice use. Through leveraging this augmented dataset of meticulously curated, balanced-class samples with variations added to it, the foundation for training and evaluating optimised models targeted at solving nuanced tasks such as pizza recognition can be established firmly.

After a careful process of dataset preparation and annotation, the model training within in Roboflow [6] platform was another important validation step. This procedure evaluated annotation accuracy and the appropriateness of a dataset for subsequent object detection tasks. Using the Roboflow [6] infrastructure, testing was also merged with annotation-based workflows to form a unified development cycle. The iterative testing verified the accurate detection of various pizza toppings, claiming that the dataset constructed was usable and credible.



Try on mobile


This is the QR code for testing the Roboflow [6] model with the prepared dataset.

roboflow

ProjectsUniverseDocumentationForum

Nicholas Vella

CV GROUP PROJECT



Task 2

object detection

Sharing Page

Classes

Upload

Unassigned

Annotate

Images298

Generate

Versions2

Deploy

Health Check

Upgrade

Switch Model:

task-2-ln6yj/2

Trained On: task-2-ln6yj 436 ImagesView Version

Model Type: Roboflow 3.0 Object Detection (Fast)

Checkpoint: COCO

mAP45.3%Precision47.5%Recall44.1%

View Model Graphs

Samples from Test Set

View Test Set

Upload Image or a Video File

Drop files here or


Select File

Paste YouTube or Image URL

Paste a link

Try With Webcam

Try On My Machine



21 objects detected

Confidence Threshold: 50%
Overlap Threshold: 50%
Label Display Mode: Draw Labels

```
{
  "predictions": [
    {
      "x": 392,
      "y": 364.5,
      "width": 88,
      "height": 57,
      "confidence": 0.934,
      "class": "Pepperoni",
      "class_id": 7
    },
    {
      "x": 655,
      "y": 535,
      "width": 76,
      "height": 82,
      "confidence": 0.93,
      "class": "Pepperoni",
      "class_id": 7
    }
  ]
}
```


Copy

roboflow

ProjectsUniverseDocumentationForum

Nicholas Vella

CV GROUP PROJECT



Task 2

object detection

Sharing Page

Classes

Upload

Unassigned

Annotate

Images298

Generate

Versions2

Deploy

Health Check

Upgrade

Switch Model:

task-2-ln6yj/2

Trained On: task-2-ln6yj 436 ImagesView Version

Model Type: Roboflow 3.0 Object Detection (Fast)

Checkpoint: COCO

mAP45.3%Precision47.5%Recall44.1%

View Model Graphs

Samples from Test Set

View Test Set

Upload Image or a Video File

Drop files here or


Select File

Paste YouTube or Image URL

Paste a link

Try With Webcam

Try On My Machine



18 objects detected

Confidence Threshold: 50%
Overlap Threshold: 50%
Label Display Mode: Draw Labels

```
{
  "predictions": [
    {
      "x": 419.5,
      "y": 543.5,
      "width": 55,
      "height": 47,
      "confidence": 0.877,
      "class": "Black Olives",
      "class_id": 8
    },
    {
      "x": 445,
      "y": 449.5,
      "width": 110,
      "height": 97,
      "confidence": 0.847,
      "class": "Mushroom",
      "class_id": 5
    }
  ]
}
```


Copy

roboflow

ProjectsUniverseDocumentationForum

Nicholas Vella

CV GROUP PROJECT



Task 2

object detection

Sharing Page

Classes

Upload

Unassigned

Annotate

Images298

Generate

Versions2

Deploy

Health Check

Upgrade

Switch Model:

task-2-ln6yj/2

Trained On: task-2-ln6yj 436 ImagesView Version

Model Type: Roboflow 3.0 Object Detection (Fast)

Checkpoint: COCO

mAP45.3%Precision47.5%Recall44.1%

View Model Graphs

Samples from Test Set

View Test Set

Upload Image or a Video File

Drop files here or

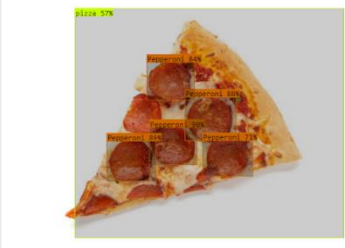
Select File

Paste YouTube or Image URL

Paste a link

Try With Webcam

Try On My Machine



6 objects detected

Confidence Threshold: 50%
Overlap Threshold: 50%
Label Display Mode: Draw Labels

```
{
  "predictions": [
    {
      "x": 382.5,
      "y": 244.5,
      "width": 83,
      "height": 85,
      "confidence": 0.985,
      "class": "Pepperoni",
      "class_id": 7
    },
    {
      "x": 229,
      "y": 271,
      "width": 86,
      "height": 70,
      "confidence": 0.89,
      "class": "Pepperoni",
      "class_id": 7
    }
  ]
}
```

Copy

Implementation of the object detectors

For this section of the task, intensive research was done to find 3 computer vision model architectures that would be ideal for the detection of pizzas and their toppings. The idea was to use three different architectures mainly:

- Faster R-CNN: In the field of computer vision, Faster R-CNN (Region-based Convolutional Neural Networks) is a well-liked and significant model. It offers better performance in terms of accuracy and speed compared to earlier models such as R-CNN and Fast R-CNN. Faster R-CNN presents a region proposal network that allows almost cost-free region proposals by sharing full-image convolutional features with the detection network. A fully convolutional network, or RPN, predicts object bounds and objectness scores at each position simultaneously. The model has two phases of operation. The RPN carries out the first step, which suggests potential object bounding boxes. These suggestions are used by the Fast R-CNN detector in the second stage to further classify and improve the bounding boxes.
- SSD: Another well-known architecture in the field of computer vision object detection is the Single Shot MultiBox Detector (SSD). SSD completes both tasks in a single pass, in contrast to two-stage models like R-CNN, where one stage proposes regions and the other classifies them. SSDs become faster and more efficient as a result. In order to help the model predict the presence of objects, SSD presents the idea of default boxes, which are a collection of fixed-size boxes. There are several default boxes with varying aspect ratios used for each feature map cell. Finally, SSD is based on common CNN architectures for feature extraction, such as VGG16 (explained in the Background section). Convolutional layers are added to the model in later layers, especially for object detection.
- YOLO: (explained in the Background section)

However, finding models that are up to date was a difficult task, so we had to settle for 3 variations of the YOLO architecture.

- YOLOv5:
- YOLOv7-tiny:
- YOLOv8:

The following are the instructions carried out to train such models:

1. Cloning the GitHub repository.

```
git clone https://github.com/WongKinYiu/yolov7.git
cd yolov7
```

- o The first line clones the github repository and places the folder in C:/Users/*username/yolov5/7/8

- The second line simply moves the command prompt to the correct directory.
2. Installing all necessary Python dependencies.

```
pip install -r requirements.txt
```

- The above command installs all necessary dependencies for the model to be able to work correctly. In most cases, the cloned repository had a specific requirements.txt file that listed all the information. In rare cases where this was not provided, it was essential to follow the readme.md file to check and install all libraries used.

3. Modifying or creating files to match our database.

```
/yolov7
  /datasets
    /dataset_name
      /images
        /train
          (all training images here)
        /val
          (all val images here)
      /labels
        /train
          (all train annotations.txt here)
        /val
          (all val annotations.txt here)
```

- The above shows the correct structure that the files had to follow in order for the model to be trained correctly.
- Each annotation.txt file stores annotations for a specific image in a list, each annotation having the following format:

```
<class_id> <x_center_norm> <y_center_norm> <width_norm> <height_norm>
```

- Where:
 - <class_id> shows the specific pizza/topping in the image.
 - <x_center_norm> shows the x-coord of the centre of the bounding box.
 - <y_center_norm> shows the y-coord of the centre of the bounding box.

- <width norm> holds the width of the bounding box.
- <height norm> holds the height of the bounding box.

4. Implementing/modifying a configuration file.

```
train: C:\Users\path\to\training\images\file
val: C:\Users\path\to\val\images\file

# Number of classes
nc: 10

# Class names
names: ['pizza', 'Mushroom', 'Pepperoni',
'Yellow Peppers', 'Black Olives', 'Onion',
'Ham', 'Tomato', 'Broccoli', 'Green Olives']
```

- This is a very simple implementation of a configuration file, which was enough to train the model. The content of the configuration file is very self explanatory.
 - train: holding the path to training images.
 - val: holding the path to validation images.
 - nc: holding the number of classes to be detected.
 - names: holding a list of names of all the classes.

5. Training the model.

```
python train.py --img 640 --batch 16 --epochs 50 -
-data pizza_toppings.yaml --weights yolov7-tiny.pt
```

- The above code was used to run the train.py file, which handles the training of the model. The variables passed after holding specific information according to the exact training we want to produce are as follows.
 - --img: holds the size of the images in the dataset
 - --batch: holds the batch size we want.
 - --epochs: holds the number of epochs we want the training to run for.
 - --data: holds the specific config file we want to use.
 - --weights: holds the weights that are to be used according to which version of YOLO is being used.

Evaluation of the object detectors

Different implementations had to be implemented as although similar, all models used had slightly different data handling methods.

The below are the steps taken to evaluate the models:

Load the models:

To load the models, the weights that were produced after training the model were used. Several weight files were produced, one for each checkpoint. The best.pt weight file was used, which was supposed to contain the best weights produced during training. The path to these trained weights was specified when loading the model with the torch.hub.load method.

Data Processing:

Class labels and object bounding boxes are retrieved from the text files. Here, normalization is done. These converts normalized coordinated (concerning image dimensions) to absolute pixel coordinates for accurate bounding box positioning.

Model Predictions:

The model predicts object classes and bounding boxes for every image in the testing dataset. This would provide class IDs and bounding box coordinates for each prediction.

Evaluation Metrics:

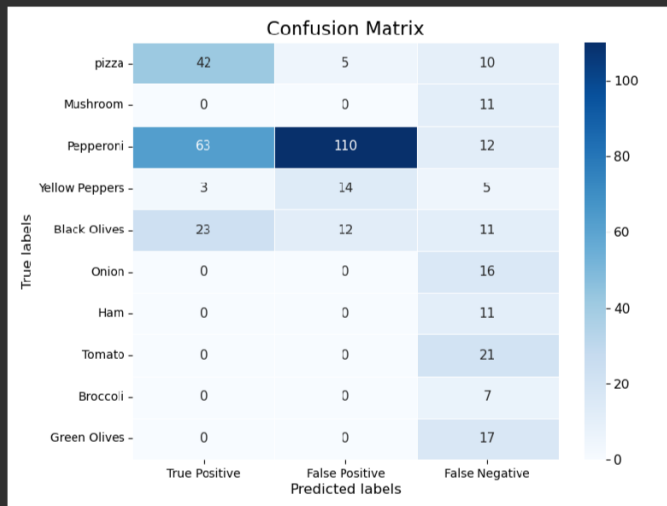
This section starts by using the Intersection over Intersections (IoU). This function calculates the bounding box overlap between the actual and predicted boundaries. The next step is the use the IoU threshold and class labels to determine matches. This step is essential to distinguish between accurate predictions (True Positives) and inaccurate predictions (False Positives/Negatives). With this information, it was now possible to calculate some evaluation metrics and figure out how our model was performing.

- Confusion Matrix: In the fields of statistics and machine learning, a confusion matrix is an essential tool for illustrating how well a classification algorithm is working. It is especially helpful for figuring out how well a model is doing in terms of classifying instances correctly and incorrectly. In multi-class problems (like ours), the instances in a predicted class are represented by each column of the matrix, and the instances in an actual class are represented by each row. The number of accurate classifications for each class is displayed by the diagonal elements, while misclassifications are indicated by the off-diagonal elements.

YOLOv7 Confusion Matrix



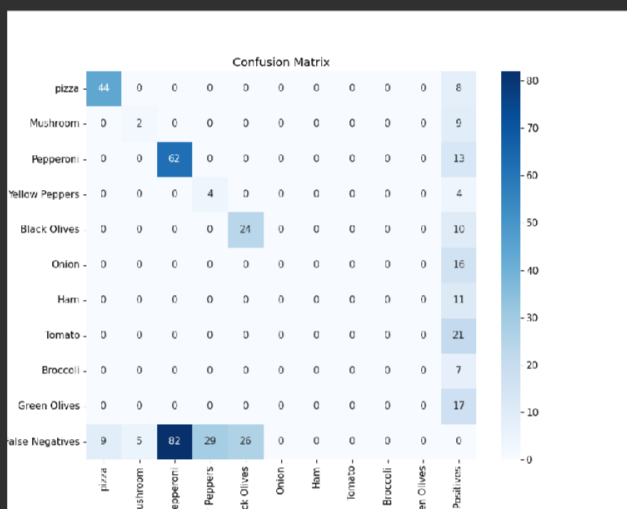
Step 0

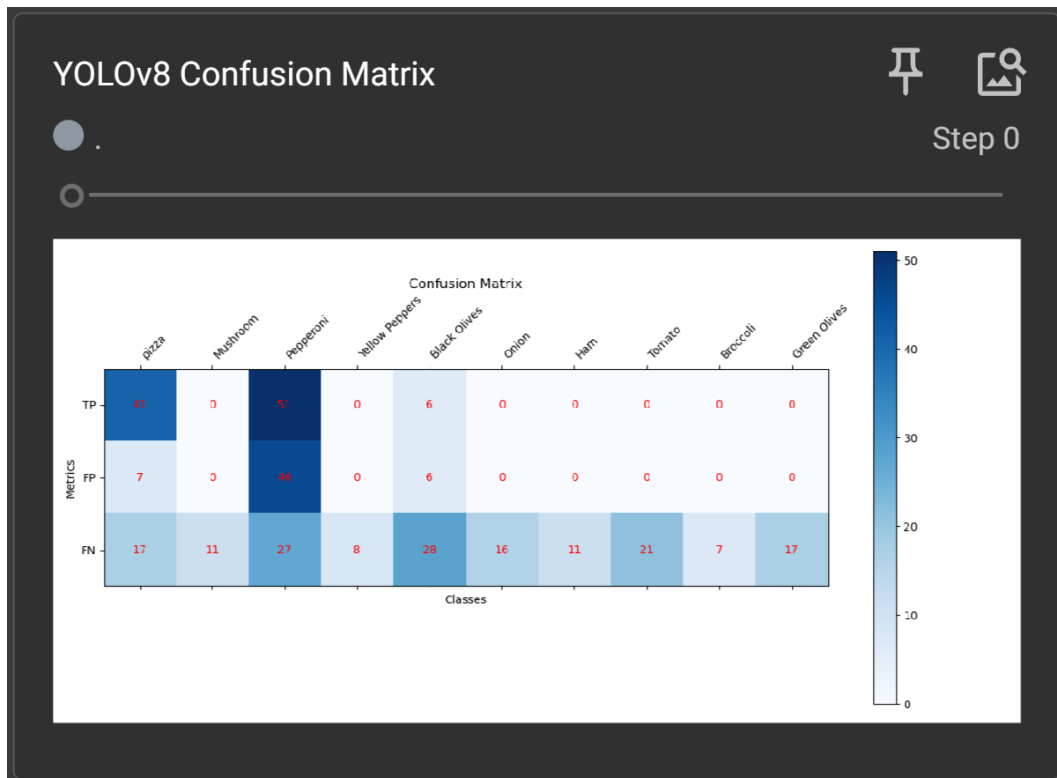


YOLOv5 Confusion Matrix



Step 0





- Precision: Precision is a metric for positive prediction accuracy. It is the percentage of pertinent instances found in the instances that were retrieved. A high precision means that the model produced a significant proportion of relevant results compared to irrelevant ones.
- Recall: Recall, which is a substitute for Sensitivity or True Positive Rate, quantifies a model's capacity to locate each and every pertinent case in a dataset. A high recall indicates that the majority of pertinent results were retrieved by the model.

The calculated metrics were all logged to tensorboard for a better, more organised comparison of the models as can be seen in the following page.

Comparing metrics:

- Precision:

```
YOLOv8: Precision = 0.789  
YOLOv7: Precision = 0.482  
YOLOv5: Precision = 0.2162
```

Notes:

When compared to the other two models, YOLOv8 has the highest precision, meaning that it makes more accurate positive predictions. This suggests that YOLOv8 has a higher chance of being accurate when predicting an object.

Compared to YOLOv8 and YOLOv5, YOLOv7 makes more positive predictions with moderate precision, but fewer accurate ones.

The fact that YOLOv5 has the lowest precision indicates that a sizable percentage of its positive predictions are false positives.

- Recall:

```
YOLOv8: Recall = 0.492  
YOLOv7: Recall = 0.5198  
YOLOv5: Recall = 0.5333
```

Notes:

Even with its lowest precision, YOLOv5 has the highest recall. This suggests that although it causes more false positive errors, it is more accurate at detecting the majority of positive cases.

Recall for YOLOv7 is marginally lower than that of YOLOv5, indicating that it misses more real positives than YOLOv5 but fewer than YOLOv8.

YOLOv8 has the lowest recall despite having the highest precision. As a result, there are more missed actual positives (false negatives) and more cautious positive predictions.

Overall Comments:

Recall is frequently decreased by increased precision and vice versa. This is referred to as the trade-off between precision and recall. This is evident in our evaluation which makes a lot of sense and is a good sign that everything is working as it should be. In conclusion, every iteration of the YOLO model demonstrates proficiency in various facets of object detection functionality.

Analytics:

The following shows the output of the algorithm explained above.

References and List of Resources Used

[1] Keras, "Applications," Keras, 2024. [Online]. Available: <https://keras.io/api/applications/>. [Accessed: 14-Jan-2024].

[2] Keras, "VGG16 Function," Keras, 2024. [Online]. Available: <https://keras.io/api/applications/vgg/#vgg16-function>. [Accessed: 14-Jan-2024].

[3] Keras, "VGG19 Function," Keras, 2024. [Online]. Available: <https://keras.io/api/applications/vgg/#vgg19-function>. [Accessed: 14-Jan-2024].

[4] Keras, "ResNet50 Function," Keras, 2024. [Online]. Available: <https://keras.io/api/applications/resnet/#resnet50-function>. [Accessed: 14-Jan-2024].

[5] U. Dave, "PizzaDetector," GitHub, 2024. [Online]. Available: <https://github.com/utsavDave97/PizzaDetector>. [Accessed: 14-Jan-2024].

[6] Roboflow, "Roboflow," Roboflow, 2024. [Online]. Available: <https://roboflow.com/>. [Accessed: 14-Jan-2024].

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.