Coursework Submission Cover Sheet

LIVERPOOL JOHN MOORES UNIVERSITY

To be completed by the student:

LJMU ID NO:	967350
Student Name Full Name:	Wannaka Waththa Waduge Raneesha Dilakshan Fernando
Delivery Mode:	Distance Learning
Location:	Colombo
Module Code:	6613ELEICB
Module Title:	Autonomous Systems and Machine Learning
Coursework Title:	Develop a deep learning model to detect objects in images
Name of Tutor:	H.M.R.T. Bandara
Due Date:	25/04/2024
Word Count:	2694
Marker (If different from tutor):	

Acknowledgement

This assignment would not have been possible without the support and contributions of several resources. I would like to express my gratitude to the following:

Roboflow: I am thankful to the team at Roboflow for providing the high-quality, annotated food dataset used in this assignment. Their platform and resources were invaluable in streamlining the data preparation process and enabling seamless integration with the YOLOv8 model.

Ultralytics: I extend my appreciation to the developers of the YOLOv8 object detection model at Ultralytics. Their state-of-the-art model and comprehensive documentation facilitated the implementation and training process within this assignment.

Open-Source Community: I acknowledge the immense contributions of the open-source community, whose collective efforts have led to the development and dissemination of numerous libraries, tools, and resources that were instrumental in the successful completion of this assignment. I am grateful for the sharing of knowledge and collaboration within this community.

Academic Resources: I would like to thank the authors of the various research papers, tutorials, and online resources that provided valuable insights and guidance throughout the course of this assignment. Their work has significantly contributed to my understanding of object detection techniques and deep learning methodologies.

Faculty: I express my gratitude to my faculty members for their support, feedback, and insightful discussions, which have helped shape and refine my approach to this assignment. Their valuable inputs have played a crucial role in my learning and growth.

Computing Resources: Finally, I acknowledge the computational resources provided by Google Colab, which enabled me to efficiently train and evaluate the deep learning models.

The successful completion of this assignment would not have been possible without the collective efforts and contributions of all these resources. I am profoundly grateful for their support and the opportunity to work on this exciting and challenging task.

Table of Contents

Table of Figures

Introduction

In recent years, object detection has become a crucial task in computer vision with a wide range of applications, including autonomous vehicles, surveillance systems, and image/video analysis. The ability to accurately identify and localize objects in images or video streams is a fundamental step towards understanding and interpreting visual data. Deep learning techniques, particularly convolutional neural networks (CNNs), have revolutionized the field of object detection, achieving state-of-the-art performance on various benchmark datasets.

The objective of this assignment was to develop a deep learning model capable of detecting objects in images. Specifically, I focused on the food dataset provided by Roboflow, which contains a diverse collection of food items across multiple categories. This dataset posed an interesting challenge due to the varying appearances, textures, and compositions of different food items, as well as the potential occlusions and clutter present in real-world scenarios.

To tackle this task, I employed the YOLOv8s (You Only Look Once) model, a state-of-the-art real-time object detection system developed by Ultralytics. YOLOv8 is known for its exceptional speed and accuracy, making it well-suited for real-time applications. The 's' variant of the model is a scaled-down version, designed for efficient inference on resource-constrained devices while maintaining competitive performance.

The main steps followed in this project were:

Data Preparation: Exploring the food dataset, understanding its format, and preprocessing the data by resizing, normalizing, and splitting it into training, validation, and test sets.

Model Selection and Training: Modifying the YOLOv8s model to accommodate the specific object classes in the food dataset and training the model on the prepared data while fine-tuning hyperparameters for optimal performance.

Evaluation and Visualization: Assessing the trained model's performance on the test set using appropriate metrics and visualizing the model's predictions on sample images to qualitatively analyse its capabilities.

Reporting and Reflection: Documenting the approach, results, challenges encountered, and potential future improvements for the object detection model.

Data Preparation

For this project, I utilized the food dataset provided by Roboflow (Roboflow, 2024), focusing specifically on the categories of fast foods, beef bowls, salads, drinks, and soups. Roboflow offers high-quality, annotated datasets in the widely used COCO format (Lin, 2014), which includes image files and corresponding annotations containing object bounding boxes and class labels.

One of the advantages of using Roboflow is that it allows for directly importing the dataset into the YOLOv8 format, which automatically handles the splitting of data into training, validation, and test sets (Roboflow documentation, 2024). I leveraged this functionality to obtain the three subsets required for our object detection task.

Before feeding the data to the model, several preprocessing steps were performed. First, I resized the images to a uniform resolution of 640x640 pixels to ensure consistent input dimensions for the YOLOv8s model. This step is essential as deep learning models often require fixed input sizes for efficient processing (Krizhevsky, 2012).

While the dataset splitting was managed automatically by Roboflow, I did need to edit the paths for the training, validation, and test sets within the data.yaml file. This step ensured that the model had access to the appropriate data subsets during training and evaluation.

With the dataset properly prepared and organized, I could proceed to the next step of model selection and training.

Model Selection and Training

For this object detection task, I employed the YOLOv8s (You Only Look Once) model developed by Ultralytics (Ultralytics, 2024). YOLOv8 is a state-of-the-art real-time object detection system that builds upon the success of its predecessors, offering exceptional speed and accuracy. The 's' variant, which I chose, is a scaled-down version designed for efficient inference on resource-constrained devices while maintaining competitive performance.

The YOLOv8s model is a convolutional neural network (CNN) architecture pre-trained on the COCO dataset (Lin, 2014), which consists of eighty common object categories. To adapt the model to our specific task of detecting fast foods, beef bowls, drinks, and soups, I modified the final layers of the network to accommodate the relevant object classes present in our dataset.

The training process involved feeding the pre-processed images and corresponding annotations into the YOLOv8s model. I utilized the following hyperparameters during training:

- Batch size: 8
- Optimizer: NAdam (Nesterov-accelerated Adaptive Moment Estimation) (Dozat, 2016)
- Initial learning rate: 0.000417
- Momentum: 0.8

These hyperparameters were chosen based on empirical evidence and experimentation to strike a balance between training stability and convergence speed. The NAdam optimizer, a variant of the popular Adam optimizer, incorporates Nesterov momentum, which can potentially lead to faster convergence and better performance for certain types of problems (Sutskever, 2013).

The model was trained for 20 epochs, with the input images resized to 640x640 pixels. During training, I monitored the validation metrics, such as mean Average Precision (mAP), to assess the model's performance and guide the hyperparameter tuning process. Fine-tuning the hyperparameters, including the learning rate and momentum, can potentially lead to improved performance and faster convergence (Bengio, 2012).

After training the model on the food dataset, I proceeded to evaluate its performance on the test set and visualize its predictions.

Evaluation and Visualization

To assess the performance of the trained YOLOv8s model, I evaluated it on the held-out test set and analysed various metrics and visualizations.

Confusion Matrix

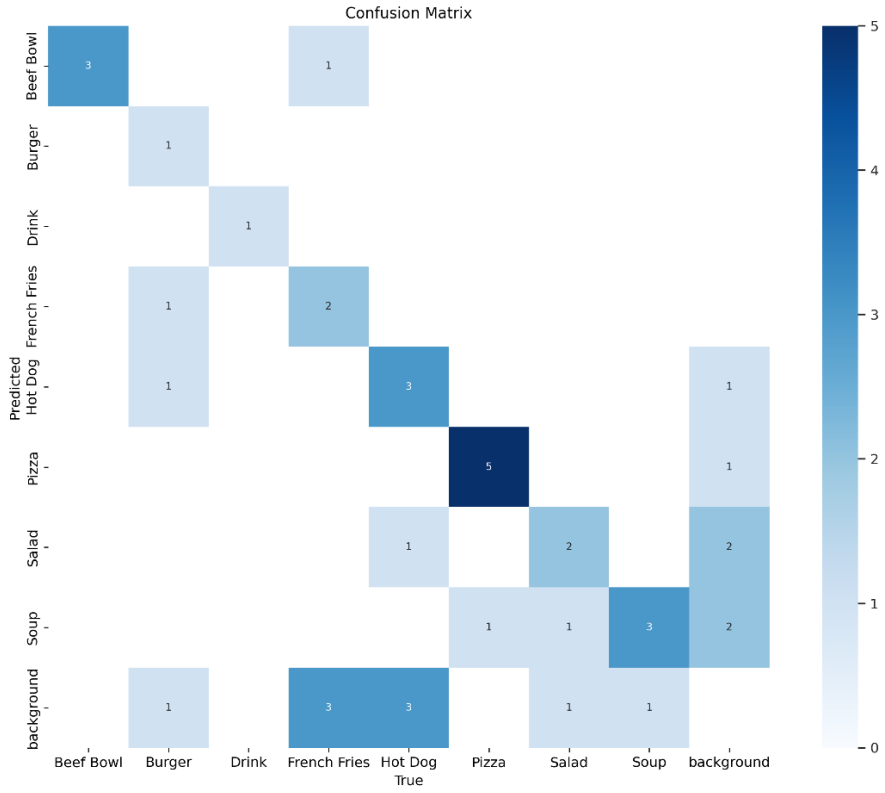

Figure 1: Confusion Matrix

The confusion matrix (above image) provides a comprehensive view of the model's performance across different classes. Each row represents the actual class, while each column represents the predicted class. The diagonal elements indicate the number of correctly classified instances for each class.

From the confusion matrix, I can observe that the model performed well in detecting certain classes, such as "Beef Bowl" and "Pizza," with few misclassifications for “Pizza.” However, there were some instances where the model struggled, such as confusing "Burger" with "French Fries", “French Fries” with “Beef Bowl,” “Hot Dog” with “Salad” and “Salad” with “Soup.”

This insight can guide future efforts in improving the model's performance, potentially by augmenting the training data or fine-tuning the model's architecture for better discrimination between similar classes.

Training Curves

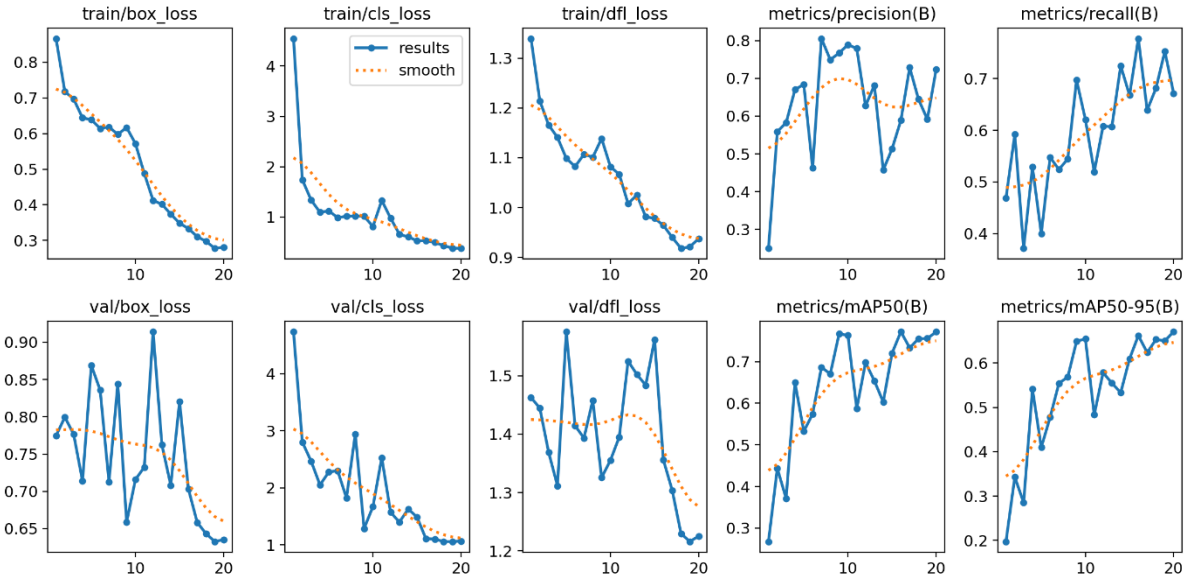

Figure 2: Training Curves

Above image presents various training curves that provide valuable insights into the model's learning behaviour. The "train/box_loss" and "val/box_loss" plots show the bounding box regression loss, which measures the model's ability to accurately predict object locations.

The "train/cls_loss" and "val/cls_loss" plots depict the classification loss, reflecting the model's performance in assigning correct class labels to the detected objects.

The "train/dfl_loss" and "val/dfl_loss" plots represent the combined loss, which is a weighted sum of the bounding box regression and classification losses.

By analysing these curves, I can observe a steady decline in the training losses as the model continues to learn from the data. However, it is crucial to monitor the validation losses to ensure the model does not overfit to the training set.

Data Augmentation

After the first 10 epochs, I observed a drop in the mAP values, as evident from the "metrics/mAP50(B)" and "metrics/mAP50-95(B)" plots. To improve the model's generalization capability and prevent overfitting, I introduced various data augmentation techniques using the Albumentations library (Albumentations: A Python library for image augmentation, 2024). Specifically, I applied random blurring, median blurring, grayscale conversion, and contrast-limited adaptive histogram equalization (CLAHE) to a small fraction of the training images.

After incorporating these augmentations, I observed a temporary dip in the mAP values, but subsequently, the curves began to rise again. This behaviour is expected, as the model initially struggles to adapt to the augmented data, but eventually learns to generalize better, leading to improved performance.

Data augmentation is a widely used technique in deep learning to introduce variations in the training data, mimicking real-world scenarios and helping the model become more robust to different image transformations and conditions (Shorten, 2019).

Evaluation Metrics:

The graphs also highlight several evaluation metrics calculated on the validation set during training. The "metrics/precision(B)" and "metrics/recall(B)" plots represent the model's precision and recall, respectively, which are important measures of its overall performance.

Additionally, the "metrics/mAP50(B)" and "metrics/mAP50-95(B)" plots depict the mean Average Precision (mAP) at different Intersection over Union (IoU) thresholds. As mentioned in the report, the final mAP50 and mAP50-95 values achieved by the model on the test set are 0.771 and 0.671, respectively.

These mAP values indicate that the model has achieved an elevated level of performance in detecting objects from the given food categories. However, there is still room for improvement, and further fine-tuning or architectural modifications could potentially enhance the model's accuracy.

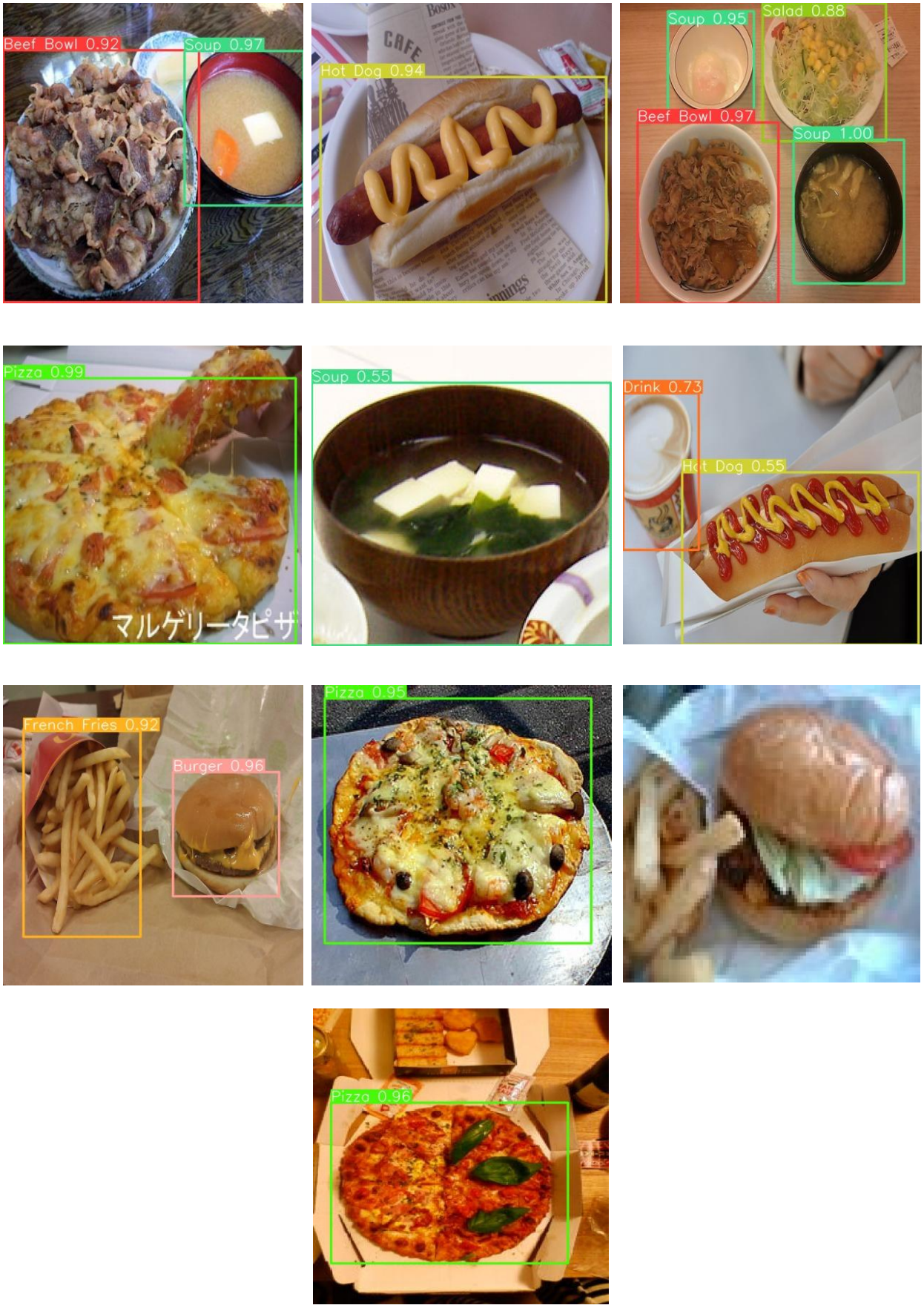

Figure 3: The ten predicted images.

By tuning the hyperparameters I handpicked the best combo for this by running and checking the results multiple times and from the best I got the model managed to detect the objects of nine images out of ten including boundary boxes and with high confidence scores.

Overall, the visualizations provided valuable insights into the model's training process, performance, and areas for potential improvement. In the next section, I will discuss the specific challenges encountered during the project and suggest potential future enhancements.

Challenges and Potential Improvements

Throughout the course of this project, I encountered several challenges and identified areas for potential improvement, which are discussed below:

Challenges

Computational Resources: Training deep learning models, especially for object detection tasks, can be computationally intensive and resource demanding. In this assignment, I faced challenges related to limited access to GPU resources. Running the model on Google Colab's free version without GPU (T4) was often slow and prone to crashes due to memory limitations. This constraint made the training process time-consuming and difficult to manage efficiently.

Hyperparameter Tuning: Determining the optimal set of hyperparameters for training the YOLOv8 model was a significant challenge. I extensively explored different combinations of hyperparameters, such as learning rates (0.0008,0.0004, 0.0002, and more), momentum values (0.8, 0.85, 0.9, and more), optimizers (Adam, AdamW, NAdam, RAdam, RMSProp, SGD), YOLO model variants (n, s, m, l, and x), also the batch size as well (4, 8, 16, 32 and more) and the number of epochs. Finding the right balance between these hyperparameters was crucial to achieve optimal performance and prevent issues like overfitting or underfitting.

For instance, this below image shows the training curves for YOLOv8n, epochs=20, batch size=16, optimizer=AdamW, Learning rate=0.000833 and momentum=0.9. with this approach out of the ten images three are not detected and the confidence scores were low as well.

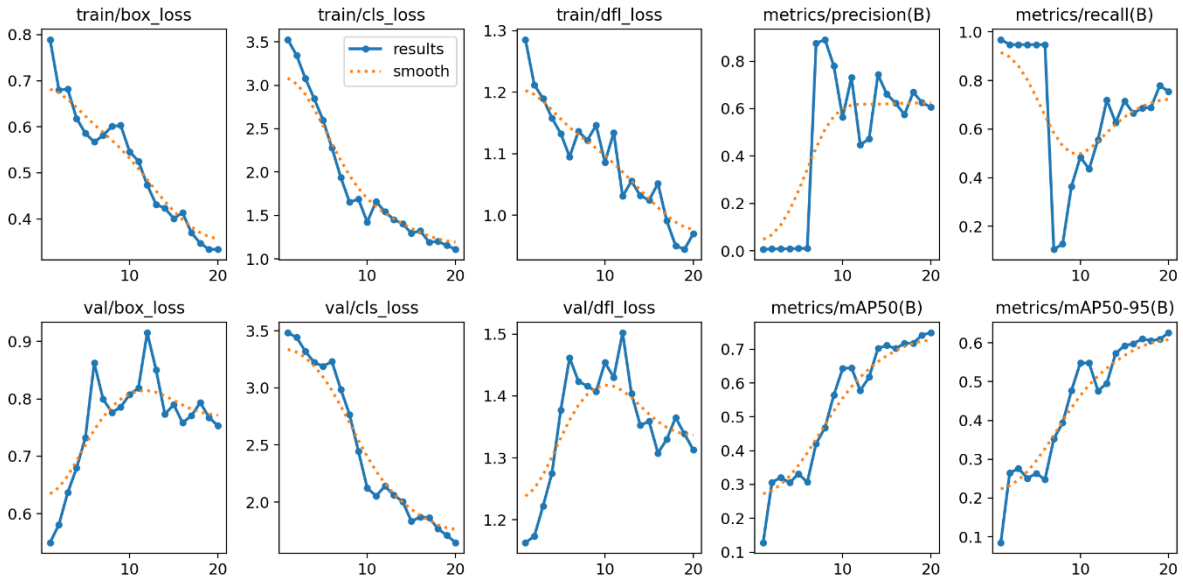

Figure 4: Tuning hyperparameters scenario1 training curves.

Another way I evaluated this was with their parameter's such as epochs=20, batch size=4, optimizer=Adam, Learning Rate0=0.0002 and momentum=0.9 with YOLOv8s with this approach out of the ten images only one is not detected but there were some images with detecting two food classes for only one and the confidence scores were low as well.

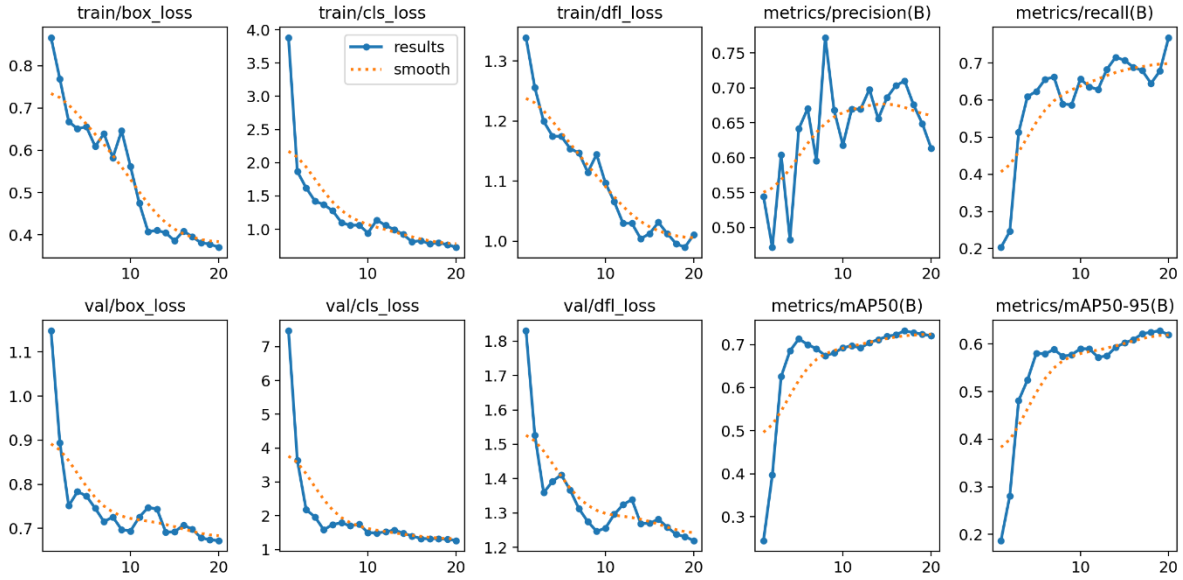

Figure 5: Tuning hyperparameters scenario2 training curves.

Like this I tinkered around with the hyperparameters values till I got the close to perfect score. And I achieved this with YOLOv8s, epochs=20, batch size=8, optimizer=Nadam, Learning rate0=0.000417 and momentum=0.8.

Potential Improvements

Data Augmentation Strategies: While I employed basic data augmentation techniques, exploring more advanced strategies, such as cutout regularization, random erasing, or adversarial data augmentation, could potentially improve the model's robustness and generalization capabilities.

Model Ensemble: Combining the predictions of multiple models trained on different subsets of the data or with different architectures could potentially improve the overall performance and robustness of the object detection system.

Transfer Learning and Fine-tuning: While I utilized a pre-trained model (YOLOv8s) and fine-tuned it on the food dataset, exploring different pre-trained models or fine-tuning strategies could lead to improved performance, especially for challenging object classes or scenarios.

Incorporating Context Information: Object detection models could benefit from incorporating contextual information, such as the relationships between objects, scene understanding, or semantic segmentation, to improve the accuracy of localization and classification.

Deployment Optimization: For real-world applications, optimizing the model's inference time and memory footprint could be crucial, especially for resource-constrained devices. Techniques like quantization, pruning, or specialized hardware acceleration could be explored to achieve efficient deployment.

By acknowledging these challenges and potential improvements, I can better understand the limitations of the current approach and identify avenues for future research and development in the field of object detection.

References

(2024). Retrieved from Roboflow: https://roboflow.com/

(2024). Retrieved from Ultralytics: https://www.ultralytics.com/

(2024). Retrieved from Albumentations: A Python library for image augmentation: https://github.com/albumentations-team/albumentations

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade* (pp. 437-478).

Dozat, T. (2016). *Incorporating Nesterov momentum into Adam.* ICLR Workshop.

Krizhevsky, A. S. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (p. 25).

Lin, T. Y. (2014). Microsoft coco: Common objects in context. *In European conference on computer vision* (pp. 740-755). Springer: Cham.

Roboflow documentation. (2024). Retrieved from Roboflow: https://docs.roboflow.com/

Shorten, C. &. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 1-48.

Sutskever, I. M. (2013). On the importance of initialization and momentum in deep learning. *International conference on machine learning* (pp. 1139-1147). PMLR.

Appendices

```
!pip install ultralytics
```

```
from ultralytics import YOLO
import os
from IPython.display import display, Image
from IPython import display
display.clear_output()
!yolo mode=checks
```

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="2xerwTYEFXEQQAnGEBah")
project = rf.workspace("school-doliz").project("foods-prkjq")
version = project.version(2)
dataset = version.download("yolov8")
```

```
!yolo task=detect mode=train model=yolov8s.pt
data={dataset.location}/data.yaml epochs=20 imgsz=640 batch=8
optimizer=NAdam lr0=0.000417 momentum=0.8
```

```
Image(filename=f'/content/runs/detect/train/confusion_matrix.png',
width=600)
```

```
Image(filename=f'/content/runs/detect/train/results.png', width=600)
```

```
!yolo task=detect mode=val
model=/content/runs/detect/train/weights/best.pt
data={dataset.location}/data.yaml
```

```
!yolo task=detect mode=predict
model=/content/runs/detect/train/weights/best.pt conf=0.5
source={dataset. location}/test/images
```

```
import glob
from IPython.display import Image, display

for image_path in glob.glob('/content/runs/detect/predict/*.jpg'):
    display(Image(filename=image_path, height=600))
    print("\n")
```

Here is the link for the code in colab :

https://colab.research.google.com/drive/1iW776PE7_mgTkARzQUHKo4kTjUV6nKXv?usp=sharing

Also, the downloaded .ipynb code file will be included separately.

Link for the GitHub Repository:

https://github.com/Raneeshafdo/A-Deep-Learning-Model-to-Detect-Objects-in-Images.git