MANGO FRUIT DETECTION FROM AERIAL IMAGE

by

MARAWAN ASHRAF FAWZY ELDEIB

1181102334

Session 2023/2024

The project report is prepared for

Faculty of Engineering

Multimedia University

in partial fulfilment for

Bachelor of Engineering (Hons) Electronics

majoring in Computer

FACULTY OF ENGINEERING

MULTIMEDIA UNIVERSITY

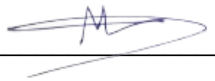
JULY 2024

DECLARATION

I hereby declare that this work has been done by myself and no portion of the work contained in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

I also declare that pursuant to the provisions of the Copyright Act 1987, I have not engaged in any unauthorised act of copying or reproducing or attempt to copy / reproduce or cause to copy / reproduce or permit the copying / reproducing or the sharing and / or downloading of any copyrighted material or an attempt to do so whether by use of the University's facilities or outside networks / facilities whether in hard copy or soft copy format, of any material protected under the provisions of sections 3 and 7 of the Act whether for payment or otherwise save as specifically provided for therein. This shall include but not be limited to any lecture notes, course packs, thesis, text books, exam questions, any works of authorship fixed in any tangible medium of expression whether provided by the University or otherwise.

I hereby further declare that in the event of any infringement of the provisions of the Act whether knowingly or unknowingly the University shall not be liable for the same in any manner whatsoever and undertakes to indemnify and keep indemnified the University against all such claims and actions.

Signature: 

Name: MARAWAN ASHRAF FAWZY ELDEIB

Student ID: 1181102334

Date: 4th July 2024

ACKNOWLEDGEMENT

I express my profound gratitude to my supervisor, Mr. Mohd Haris Lye Bin Abdullah, for his invaluable assistance, guidance, and support throughout this project. His extensive expertise in agriculture and artificial intelligence, combined with his practical insights into model development and implementation, has been instrumental in shaping the direction of my research. His contributions have significantly enriched this project and facilitated my professional development.

I am also deeply indebted to Ir. Prof. Dr. Hezerul Bin Abdul Karim, thank you for your invaluable advice and guidance throughout the research process. His expertise in agricultural technology and machine learning has dramatically enhanced the methodologies employed in this project. His insightful feedback was crucial in addressing the complexities of image processing techniques and their application in agriculture.

Furthermore, I sincerely thank Dr Lim Sin Liang for her comprehensive guidance and support throughout the FYP 1 and FYP 2 courses. Her structured approach and clear instructions were essential in achieving the project milestones and adhering to the high standards expected in these final-year projects. Her guidance on effective communication and presentation skills was vital in articulating the findings and significance of my work.

Finally, I express my heartfelt appreciation to my parents for their unwavering support, encouragement, and belief in me throughout my academic journey. Their constant motivation and sacrifices have been fundamental to my success.

ABSTRACT

This thesis explores the application of deep learning models for mango detection using aerial imagery, focusing on two primary models: Faster R-CNN and YOLO variants. Experiments were conducted using datasets from the University of Sydney, CQUniversity, and a locally prepared dataset.

The Faster R-CNN model was evaluated with VGG-16 and ResNet-50 backbones. The ResNet-50 model, enhanced with the Detectron2 framework and AdamW optimizer, achieved the best performance with an AP50 of 93.02 on the Sydney dataset and 80.52 on the local dataset.

YOLO models, particularly YOLOv8, demonstrated superior performance across different configurations and datasets. The best configuration for YOLOv8 on the CQUniversity dataset, early stopping with SGD, achieved a mAP@0.5 of 0.992. On the Sydney dataset, YOLOv8 with early stopping and augmentation achieved a mAP@0.5 of 0.959. In comparison, the best performance for YOLOv10 was a mAP@0.5 of 0.970 on the CQUniversity dataset with early stopping and augmentation.

Overall, YOLOv8 outperformed both YOLOv10 and traditional Faster R-CNN implementations, with the highest performance seen in the YOLOv8 model on the CQUniversity dataset.

The development of MangoVision, a user-friendly GUI application, integrated these advanced object detection models with features such as image and video processing, GPS coordinates extraction, and annotation, providing a practical solution for real-time mango detection in agricultural applications.

This research demonstrates that YOLOv8, particularly with early stopping and SGD, offers the highest performance for mango detection, making it suitable for real-world agricultural monitoring. Future work should focus on enhancing model performance through additional data augmentation, exploring other state-of-the-art models, and expanding the application to other types of fruit detection.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AdamW	Adaptive Moment Estimation
AI	Artificial Intelligence
AP	Average Precision
BCE	Binary Cross-Entropy
CIOU	Complete Intersection Over Union
CNN	Convolutional Neural Network
Conv	Convolutional Layers
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CUDA	Compute Unified Device Architecture
CV	Computer Vision
C2f	Cross-Stage Partial Network
DL	Deep Learning
FC	Fully Connected
FCN	Fully Convolutional Networks
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Networks
FPS	Frames Per Second
FYP	Final Year Project
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IoU	Intersection over Union
mAP	Mean Average Precision
ML	Machine Learning
MMU	Multimedia University
PAN	Path Aggregation Network
PIL	Python Imaging Library
PL	Pooling Layer

ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
R-CNN	Region-based Convolutional Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
TN	True Negative
TP	True Positive
TXT	Text File Document
UAV	Unmanned Aerial Vehicle
VSCode	Visual Studio Code
YOLO	You Only Look Once

CHAPTER 1 INTRODUCTION

1.1 Background

The agricultural sector, particularly mango cultivation and harvesting, is vital due to the fruit's popularity for its juiciness, delicious taste, and nutritional value. Mango is one of the most important tropical fruits commercialised and consumed worldwide, serving as an excellent source of fibre, provitamin A carotenoids, vitamin C (ascorbic acid), and phenolics [1]. Mango farming plays a crucial role in Malaysia's agriculture, providing a steady income for farmers and those involved in marketing and thereby contributing to rural development and environmental sustainability. Mango trees prevent soil erosion, mitigate climate change effects, and maintain fertile land. However, traditional mango production is labour-intensive and error-prone.

Recent advancements in agricultural technology, particularly aerial imaging and artificial intelligence (AI), present new opportunities for innovation [2]. These technologies, especially drones with high-resolution cameras, have revolutionised agricultural practices. These drones capture detailed images of mango orchards, which are processed using advanced deep-learning frameworks like PyTorch. Object detection models such as YOLO (You Only Look Once) and Faster R-CNN are employed. YOLO is known for its real-time processing speed, and Faster R-CNN, implemented using Detectron2, offers higher accuracy through a two-stage detection process.

Integrating AI and drone technology in agriculture enables precision farming, where inputs like water, fertilisers, and pesticides are applied precisely where needed, reducing waste and minimising environmental impact. AI-driven systems analyse data from aerial images, providing real-time insights and automating tasks such as fruit counting and yield prediction, thereby increasing efficiency and productivity in mango farming [3], [4]. This project aims to leverage these technologies to automate the detection of mango fruits, enhancing accuracy and reducing manual labour in mango orchard management. AI and drones can transform traditional farming practices,

making them more efficient and scalable and contributing to economic growth and environmental sustainability.

1.2 Problem Statements

The primary challenge addressed by this project is accurately detecting mango fruits in large-scale orchards using aerial imagery and deep learning frameworks. The problems include:

- Manual inspection of mango trees is labour-intensive and time-consuming, leading to high labour costs and lower productivity.
- Human inspectors are prone to errors, resulting in inconsistencies.
- Various factors, such as dense foliage, varied fruit appearances, the altitude and angle of the drone, fruit clustering, different camera viewpoints, and occlusion by leaves and branches, complicate the detection process.
- Scalable and efficient methods that integrate modern technology into agricultural practices are needed to improve fruit detection and harvesting accuracy.

1.3 Project Scope

This project's scope encompasses developing and evaluating an AI-based system for mango detection using aerial imagery. Specifically, the project includes:

- **Data Collection and Labelling:** Drones will be used to collect high-resolution aerial images of mango orchards. The mangoes, characterised by their oval shape and green, yellow, or red skin, will be annotated to create a comprehensive dataset for model training and evaluation.
- **Model Design and Training:** Designing and training deep learning models, specifically YOLO and Faster R-CNN variants, to detect mango fruits from aerial images. This process will involve using the PyTorch framework to implement and optimize these models for the specific task of mango detection.

- **Performance Evaluation:** Evaluating the trained models' performance to ensure accuracy and reliability using standard metrics such as precision, recall, and mean Average Precision (mAP).
- **GPS Coordinates Extraction:** Developing a method to extract the GPS coordinates of detected mangoes to facilitate efficient orchard management by providing precise location information for each detected fruit.
- **Comparative Analysis:** Conducting a comprehensive comparison of different model architectures and training strategies to determine the most effective approach for mango detection from aerial imagery.

The project does not include:

- **Extended Field Testing:** Long-term field testing and deployment in diverse environmental conditions beyond initial proof-of-concept demonstrations.

An additional component was added later in the project:

- **Graphical User Interface (GUI) Development:** Creating a user-friendly GUI that incorporates multiple features, including GPS integration, to support the detection process and enhance practical usability.

This section clearly defines the project's scope and delineates the boundaries of the research, ensuring a focused approach to achieving the project's objectives and contributions.

1.4 Project Objectives and Contributions

The main objectives of this project are:

- Evaluate public datasets on mango fruit detection and collect a new dataset from aerial drone images.

- Design and train object detection models to detect mango fruits from these aerial images.
- Assess the performance of these models using the newly collected dataset.

The project successfully developed an automated system leveraging aerial images and deep learning models to enhance the precision and effectiveness of mango detection.

The key contributions of this work are:

- Evaluated the suitability of public datasets for training deep learning models for mango detection.
- Collected and labelled a comprehensive dataset of mango fruits from aerial drone images, providing valuable data for model training and evaluation.
- Designed and trained state-of-the-art object detection models, including YOLO and Faster R-CNN, to accurately detect mango fruits from aerial images.
- Evaluated various models, including the latest versions, to identify the best speed and accuracy, demonstrating that YOLO outperforms Faster R-CNN for this project.
- Developed a graphical user interface (GUI) with multiple features to facilitate the use of the detection system, including GPS integration for detected mangoes, enhancing the practical utility of the system for precision agriculture.

This project significantly impacts food security and agricultural productivity, supporting healthier diets through optimal harvesting and promoting sustainable farming practices. It also contributes to economic growth by increasing farm productivity and creating new job opportunities in high-tech fields.

1.5 Report Outline

The remainder of this thesis is organised as follows:

- **Chapter 2: LITERATURE REVIEW**- Provides a comprehensive overview of relevant background on object detection techniques, deep learning models, and their applications in agriculture. Reviews previous work in fruit detection, discussing the evolution of techniques from traditional computer vision methods to modern deep learning approaches. Identifies gaps in current research that this thesis aims to address.
- **Chapter 3: METHODOLOGY**- Details of the experimental setup, including the data collection process, dataset preparation techniques, and annotation methods. Describes the architectures of the employed models (YOLO variants and Faster R-CNN), the implementation details using PyTorch and Detectron2, and the evaluation metrics used in this study. Explains the development process of the MangoVision interface.
- **Chapter 4: EXPERIMENTS AND RESULTS -** provides a comprehensive model performance analysis across different datasets and configurations. It includes ablation studies on various aspects, such as data augmentation, transfer learning, and the effects of different hyperparameters on model performance. It also offers detailed comparisons between YOLO and Faster R-CNN models, discussing their respective strengths and weaknesses in the context of mango detection.
- **Chapter 5: CONCLUSIONS -** summarises the overall achievements, significant findings, and conclusions drawn from the research. It provides future work recommendations, suggesting model enhancements and further research directions.

Reference Materials: The final section includes all the references cited throughout the report, adhering to the IEEE reference style. It also contains appendices that provide project code and a few model curves.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

Detecting mango fruits from aerial images involves multiple fields of study, including computer vision, deep learning, and agricultural technology. This chapter reviews the theoretical background and existing literature related to these areas. The review is organised into several sections, each focusing on different aspects of the project: the fundamentals of computer vision and object detection, advancements in deep learning models, integration of these technologies in agriculture, GPS integration for efficient orchard management, and data annotation.

2.2 Integration of AI and Drone Technology in Agriculture

The integration of AI and drone technology is transforming agricultural practices. Drones equipped with high-resolution cameras can capture detailed images of crops, which are then processed using AI algorithms to extract valuable insights. This approach enables precision farming, where resources are applied precisely where needed, improving efficiency and reducing waste.

2.2.1 Applications of Drones in Agriculture

Drones are used in various agricultural applications, including crop monitoring, soil analysis, irrigation management, and pest control. They provide a cost-effective and efficient way to collect data over large areas, which can be analysed to make informed decisions about crop management.

2.2.2 AI-Driven Crop Monitoring

AI algorithms analyse drone data to monitor crop health, predict yields, and identify issues such as pest infestations or nutrient deficiencies. This real-time monitoring allows farmers to ensure optimal crop growth and reduce losses proactively.

2.3 Importance of Mango Detection and Scope of the Review

Mango detection is crucial due to its impact on agricultural productivity and the efficiency of farming practices. Accurate detection can enhance yield prediction, improve resource allocation, and reduce labour costs. This review will cover various approaches and technologies used for object detection in agriculture, focusing on mango detection. The review spans recent advancements in deep learning and their application in agricultural settings, with a particular emphasis on aerial imagery and automated detection systems.

2.4 Fundamentals of Computer Vision and Object Detection

Computer vision is a field of artificial intelligence that enables computers to interpret and make decisions based on visual data. Object detection, an essential task in computer vision, involves identifying and locating objects within an image. The advent of deep learning has revolutionised object detection, mainly using Convolutional Neural Networks (CNNs).

Object detection generally includes object localisation and classification. Precisely, deep learning models for this task predict where objects of interest are in an image by applying the bounding boxes around these objects (localisation). Furthermore, these models also classify the detected objects into types of interest (classification). There are two primary approaches to object detection: one-stage and two-stage object detection.

2.4.1 Convolutional Neural Networks (CNNs)

In image processing, Convolutional Neural Networks (CNNs) are considered one of the most popular types of neural networks. Their structure is modelled on the visual cortex of the human brain. CNNs are a class of deep neural networks commonly used for analyzing visual imagery. They are designed to automatically and adaptively learn spatial hierarchies of features from input images.

Critical Components of CNNs:

- **Convolutional Layers:** Apply a convolution operation to the input, passing the result to the next layer.

- **Pooling Layers:** Reduce the dimensionality of each feature map but retain the most essential information.

- **ReLU Activation Function:** Applies a non-linear activation function to increase the network's ability to learn complex patterns. The ReLU (Rectified Linear Unit) function is defined as:

$$f(x) = \max(0, x) \qquad 2.1)$$

- **Fully Connected Layers:** Combine all the features learned by the previous layers across the image to identify the larger patterns.

- **SoftMax Activation Function**: Used in the output layer of classification networks to convert the logits into probabilities. The SoftMax function is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j j} \qquad 2.2)$$

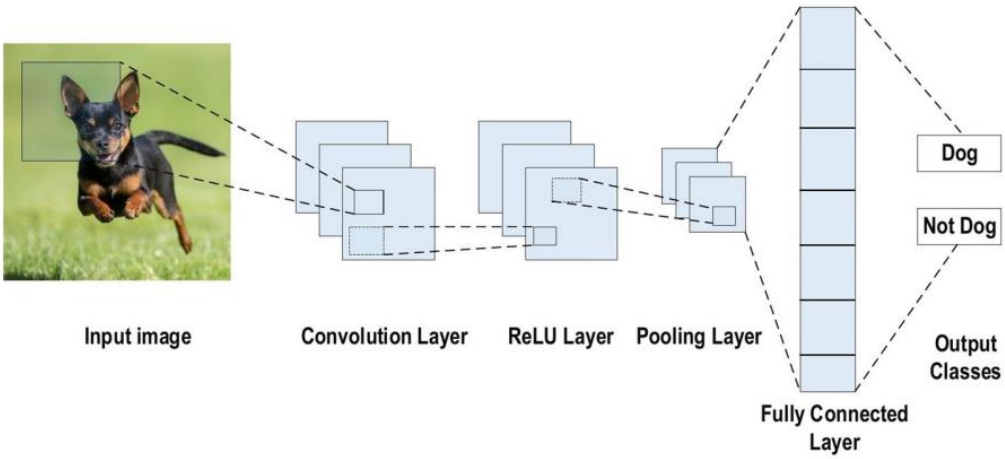


Figure 2.1: An example of CNN architecture for image classification [6].

2.4.2 Feature Extraction and Challenges

Features are specific patterns or information extracted from input images through convolutional layers. These features include edges, textures, shapes, and more complex patterns. In the context of mango detection from aerial photos, challenges in feature extraction arise due to:

- **Varying Lighting Conditions:** Different times of day and weather conditions can affect image quality, leading to inconsistencies in feature extraction.

- **Occlusions:** Mangoes may be partially hidden by leaves or branches, complicating detection.

- **Similarity in Color and Texture:** Mangoes and leaves often share similar colours and textures, making differentiation challenging.

These factors necessitate the development of robust feature extraction techniques to improve the accuracy and reliability of mango detection models.

2.4.3 PyTorch Framework

PyTorch is a widely used machine learning library based on the Torch library, designed for applications such as computer vision and natural language processing. Initially developed by Meta AI and now part of the Linux Foundation, PyTorch is valued for its flexibility, ease of use, and dynamic computational graph, facilitating intuitive model building and debugging. It is famous for research and industrial applications, especially rapid experimentation and prototyping.

Key Features of PyTorch:

- **Ease of Use:** PyTorch's syntax and structure resemble those of Python's scientific computing library, NumPy, making it accessible and easy to learn.
- **Comprehensive Support and Ecosystem:** PyTorch supports a variety of deep learning tasks, including neural networks, computer vision, natural language processing, and reinforcement learning. It also has a large and active community, with extensive documentation, tutorials, and third-party libraries that enhance its ecosystem and provide robust support for developers.

2.5 Deep Learning Models for Object Detection

Several deep learning models have been developed for object detection, each with strengths and weaknesses. This section reviews the two primary models used in this project: YOLO (You Only Look Once) and Faster R-CNN. These models can be broadly categorised into one-stage and two-stage detectors.

2.5.1 One-Stage Detectors

One-stage detectors, such as YOLO, perform object localisation and classification in a single forward pass through the network [5]. These models divide the input image into a grid and simultaneously predict each grid cell's bounding boxes and class probabilities. One-stage detectors' main advantage is their speed, making them suitable for real-time applications [6].

YOLO (You Only Look Once)

You Only Look Once (YOLO) is an object-detection algorithm introduced in 2015 by Joseph Redmon and Ali Farhadi in a research paper. YOLO's architecture was a significant revolution in the real-time object detection space, surpassing its predecessor – the Region-based Convolutional Neural Network (R-CNN). YOLO is a single-shot algorithm that directly classifies an object in a single pass by having only one neural network predict bounding boxes and class probabilities using a full image as input. YOLO models are implemented using the Ultralytics library.

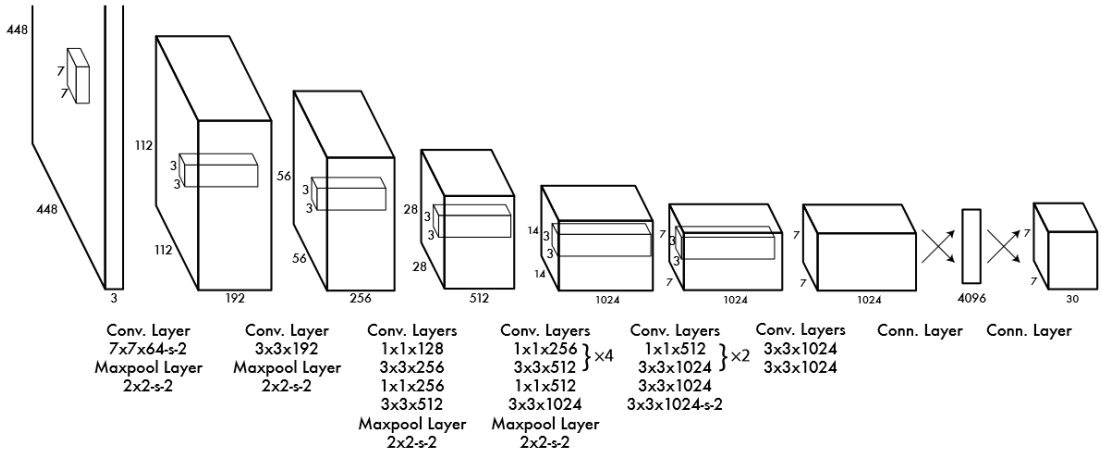


Figure 2.2: YOLO Architecture [5].

The YOLO object detection network comprises 24 convolutional layers AND two fully connected layers. The input image has a resolution of 448x448 with three colour

channels (RGB). The network uses a series of convolutional layers with 1x1 and 3x3 filters to reduce the feature space and capture spatial hierarchies. Maxpool layers are interspersed to downsample the feature maps, reducing their spatial dimensions while retaining essential features. The first layer uses a 7x7 filter with 64 filters and a stride of 2, followed by layers with 3x3 filters and increasing filter counts. The convolutional layers are pretrained on the ImageNet classification task at a resolution of 224x224, and the resolution is doubled for detection. The fully connected layers have 4096 and 30 units, respectively, predicting bounding boxes and class probabilities. YOLO achieves real-time object detection by dividing the image into a grid and making predictions for each cell, with its architecture optimised for both speed and accuracy.

Strengths of YOLO:

- Real-time processing capability
- Simple and efficient architecture
- Good performance on large-scale datasets

Weaknesses of YOLO:

- Lower accuracy for small objects
- Coarser predictions due to grid-based approach

YOLO Variants:

YOLO (You Only Look Once), developed by Joseph Redmon and Ali Farhadi at the University of Washington, emerged in 2015 as a viral object detection and image segmentation model due to its remarkable speed and accuracy.

- **YOLOv2 (2016):** Introduced enhancements such as batch normalisation, anchor boxes, and dimension clusters to improve the original model.
- **YOLOv3 (2018):** Improved performance with a more efficient backbone network, multiple anchors, and spatial pyramid pooling.

- **YOLOv4 (2020):** Added innovations like Mosaic data augmentation, an anchor-free detection head, and a new loss function.
- **YOLOv5:** Further enhanced performance with features like hyperparameter optimization, integrated experiment tracking, and automatic export to widespread formats.
- **YOLOv6 (2022):** Meituan open-sourced this version, which is used in the company's autonomous delivery robots.
- **YOLOv7:** Introduced tasks such as pose estimation on the COCO key points dataset.
- **YOLOv8:** The latest model from Ultralytics incorporates advanced features for enhanced performance, flexibility, and efficiency. It supports tasks like detection, segmentation, pose estimation, tracking, and classification.
- **YOLOv9:** Features innovations like Programmable Gradient Information (PGI) and the Generalized Efficient Layer Aggregation Network (GELAN).
- **YOLOv10:** Developed by researchers at Tsinghua University using the Ultralytics Python package, this version advances real-time object detection with an End-to-End head that eliminates the need for Non-Maximum Suppression (NMS).

YOLOv8 Architecture Overview

1. **Backbone**

The backbone of YOLOv8 is designed to extract features from the input image through a series of convolutional layers, max-pooling, and other operations. The backbone uses a CSPDarknet model that enhances learning capabilities and reduces computation. Key components include:

- **Convolutional Layers (Conv):** These layers are used for feature extraction. They apply filters to the input image to detect features like edges, textures, etc.

- **C2f (Cross-Stage Partial Network):** This module improves gradient flow and reduces computational load by splitting the input into two parts, one passing through the Bottleneck layers and the other through a shortcut connection, then concatenating the results.
- **SPPF (Spatial Pyramid Pooling-Fast):** This module pools features at multiple scales, which helps detect objects at different scales.

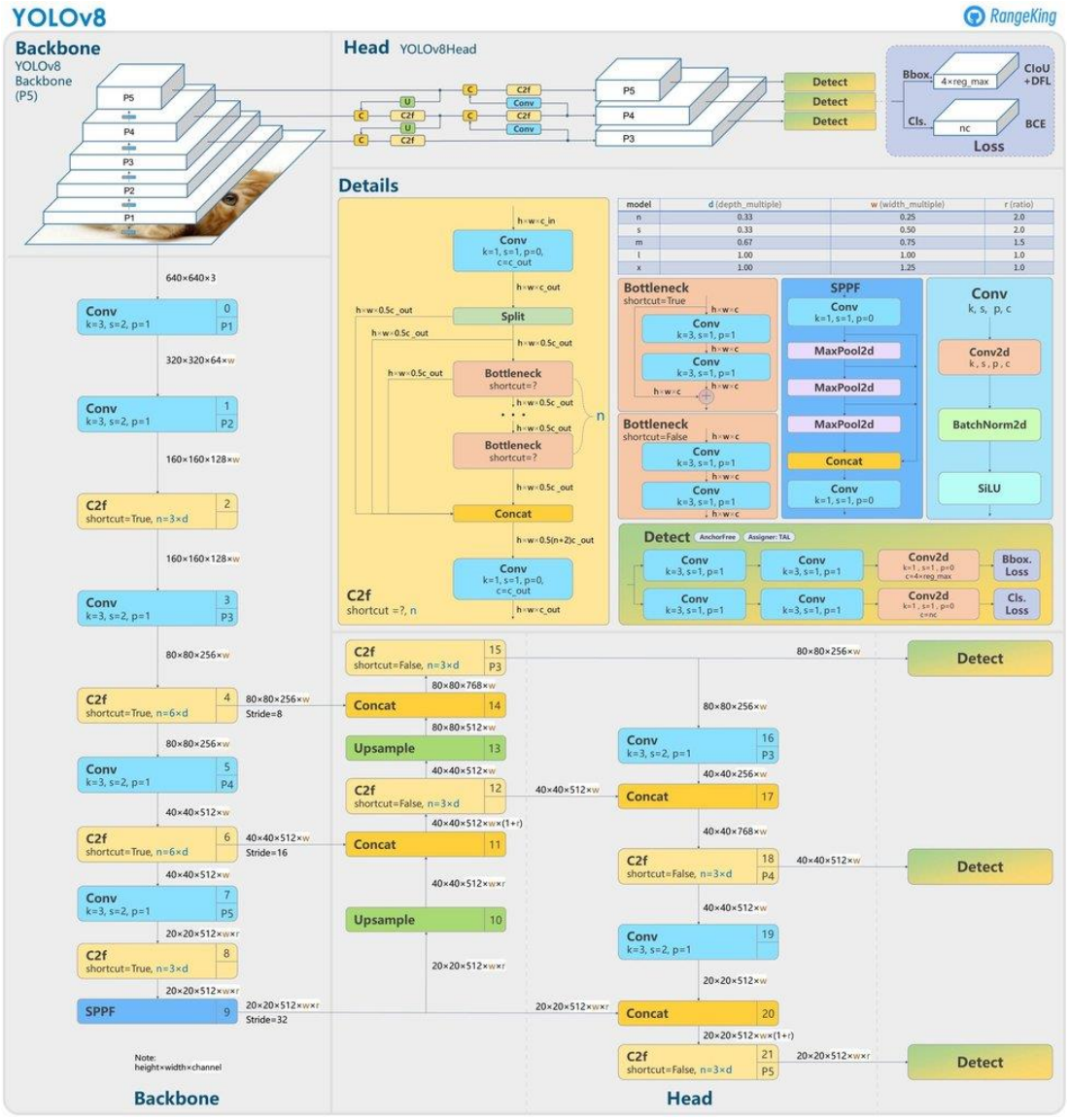


Figure 2.3: YOLOV8 Architecture [7].

2. Head

The head of YOLOv8 generates the final predictions, including the bounding boxes and class probabilities. The head combines features from different layers to improve detection accuracy. Key components include:

- **Convolutional Layers:** These layers further process the features extracted by the backbone.
- **Upsample and Concat:** These operations combine features from different layers, ensuring the model can detect objects at various scales.
- **Detection Layers:** The final layers predict the bounding boxes and class probabilities. These predictions are made at three scales (P3, P4, P5), allowing the model to handle objects of various sizes.

3. Detailed Modules

- **Bottleneck:** Consists convolutional layers with residual connections, enabling the model to learn complex features without gradient vanishing issues.
- **SPPF Module:** Aggregates features at different scales to enhance the model's ability to detect objects of varying sizes.
- **Loss Function:** The loss function used in YOLOv8 includes BCE (Binary Cross-Entropy) and CIOU (Complete Intersection Over Union) to ensure accurate bounding box predictions.

4. Key Features of YOLOv8

- **Improved Feature Extraction:** CSPDarknet53 and combining FPN and PAN architectures enhance the model's ability to extract detailed image features.
- **Multi-Scale Detection:** The model makes predictions at three different scales, improving its ability to detect small and large objects accurately.
- **Efficient Computation:** The architectural improvements reduce computational load while maintaining high detection accuracy and speed.

This detailed architecture ensures that YOLOv8 can achieve high accuracy and real-time performance, making it suitable for various real-world object detection tasks. The YOLOv8 model performs better detecting small and occluded objects, achieving a mAP50 of 99.1% and mAP50-95 of 83.5%, with an average inference speed of 50 fps on 1080p videos. [8].

YOLOv10 Architecture

YOLOv10, the latest version, builds upon the strengths of YOLOv8 with additional refinements to enhance accuracy and speed. Paper [9] Describe YOLOv10's architecture incorporating advanced techniques such as multi-scale feature pyramids and improved loss functions to handle diverse object sizes and shapes more effectively.

1. **Backbone:** Enhanced CSPDarknet with deeper layers and more residual connections for improved feature extraction.
2. **Neck:** Advanced FPN and PAN configurations for better feature aggregation and localisation precision.
3. **Head:** Refined anchor-free detection mechanism, leveraging advanced post-processing techniques like Soft-NMS to improve detection accuracy.

YOLOv10 significantly improves both mAP and inference speed, with reported mAP50 of 99.5% and mAP50-95 of 85.2% and an inference speed of 60 fps on 1080p videos. [9].

Comparative Analysis

The comparative analysis between YOLOv8 and YOLOv10 reveals several vital improvements and distinctions:

- **Accuracy:** YOLOv10 shows a slight edge over YOLOv8 regarding mAP, particularly for smaller and more complex objects.
- **Speed:** YOLOv10 achieves faster inference speeds, benefiting from architectural optimizations and improved post-processing techniques.

- **Feature Extraction:** Both models use CSPDarknet as their backbone, but YOLOv10 incorporates additional layers and connections, enhancing feature extraction capabilities.
- **Detection Mechanism:** YOLOv10's anchor-free mechanism and Soft-NMS contribute to more accurate and efficient object localisation than YOLOv8.

In summary, YOLOv10 builds upon the strong foundation of YOLOv8, introducing several key improvements that enhance accuracy and speed. The more profound and complex backbone of YOLOv10, coupled with refined detection mechanisms and advanced post-processing techniques, allows it to outperform YOLOv8 in both mAP and inference speed. These enhancements make YOLOv10 suitable for real-time applications requiring high precision and fast processing.

2.5.2 Two-Stage Detectors Faster R-CNN

Two-stage detectors, such as Faster R-CNN, generate and classify region proposals. The region proposal network (RPN) shares convolutional layers with the detection network, making it more efficient than its predecessors. Two-stage detectors are known for their high accuracy, particularly in detecting objects of varying sizes and cluttered scenes. [10].

Faster R-CNN (Region-based Convolutional Neural Networks)

Faster R-CNN is a two-stage object detection framework that generates region proposals and classifies each proposal [10], [11]. The region proposal network (RPN) shares convolutional layers with the detection network, making it more efficient than its predecessors [12].

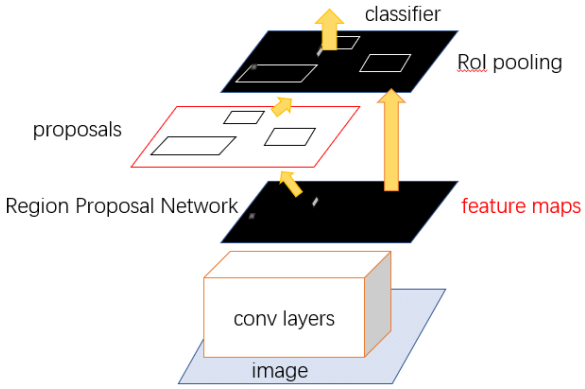


Figure 2.4: Faster R-CNN Architecture [10].

Strengths of Faster R-CNN:

- High detection accuracy
- Robust to variations in object size and background
- Effective in complex scenes

Weaknesses of Faster R-CNN:

- Slower processing speed compared to YOLO
- More computationally intensive

Faster R-CNN Backbones:

- **ZFNet:** Introduced by Zeiler and Fergus, ZFNet is an early deep-learning model that comprises eight layers, including five convolutional layers [13]. Despite its simplicity, ZFNet's innovations in visualisation techniques and understanding of convolutional layer activations provided valuable insights into CNNs' inner workings [14].

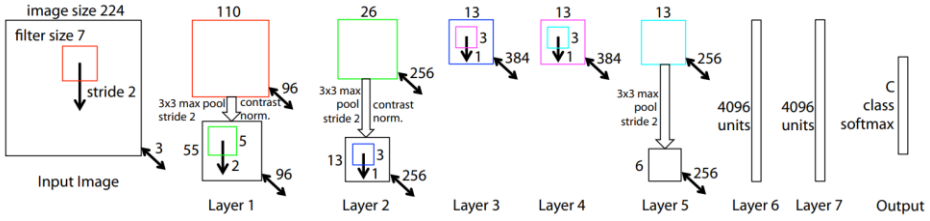


Figure 2.5: ZFNet Model Architecture [14].

- **VGG-16:** Developed by the Visual Geometry Group, VGG-16 consists of 16 layers, with thirteen convolutional layers and three fully connected layers. Each convolutional layer uses small 3x3 filters, which stack on top of each other to capture complex patterns. VGG-16 performs exceptionally well in extracting fine-grained features but demands high computational resources. When employing the VGG-16 backbone, the Faster R-CNN architecture relies on a selective search algorithm to enhance detection accuracy by focusing on potential areas of interest.

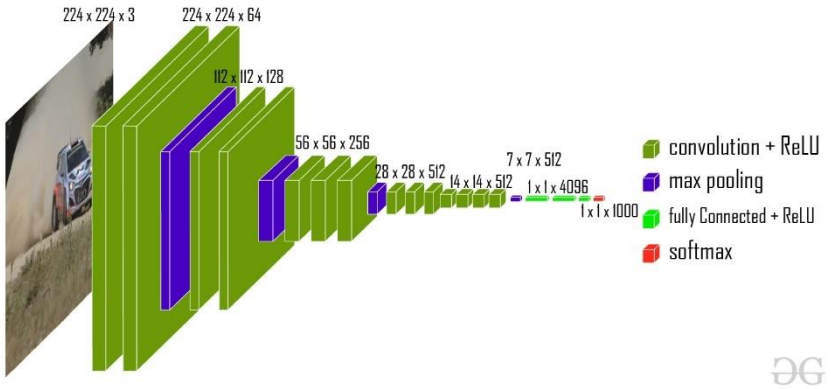


Figure 2.6: VGG16 Model Architecture [15].

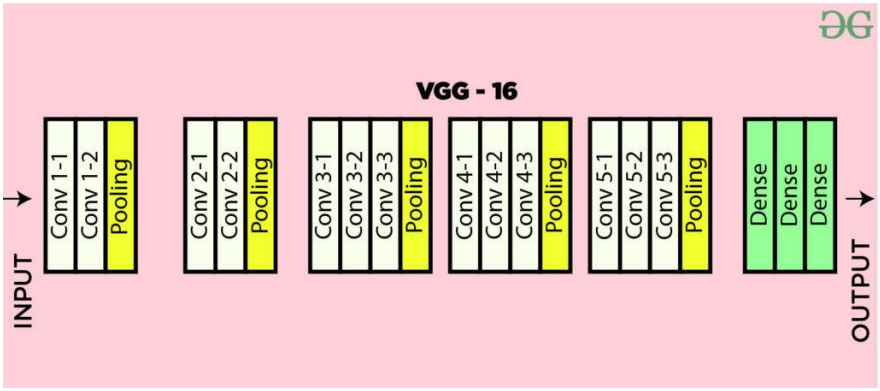


Figure 2.7: VGG16 Model Architecture Map[15].

- **ResNet-50 and ResNet-101:** ResNet, introduced by He et al., addresses the vanishing gradient problem through residual learning. ResNet-50 and ResNet-101 use residual blocks with skip connections, allowing gradients to flow directly through the network. ResNet-50 balances depth and computational efficiency, while ResNet-101 offers higher accuracy but requires more computational resources.

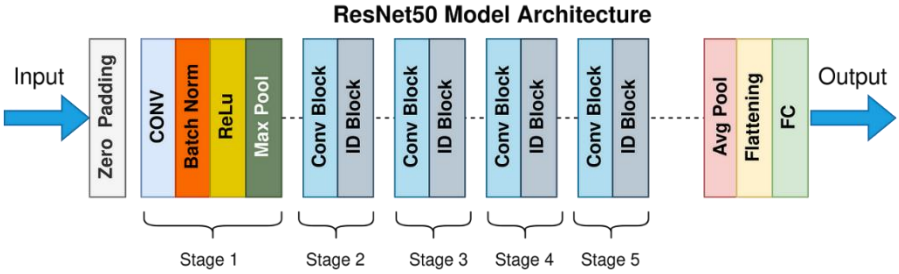


Figure 2.8: ResNet50 Model Architecture Map [16].

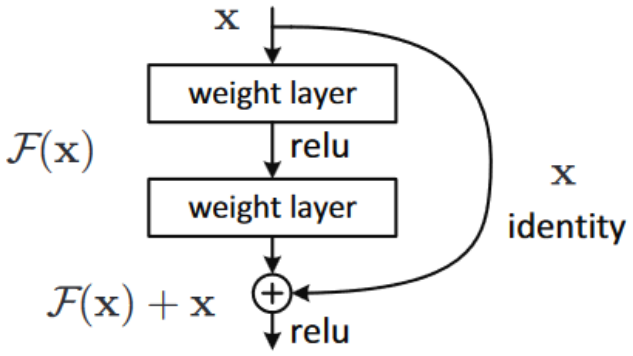


Figure 2.9: Residual learning: a building block [17].

This table compares the various backbones used for Faster R-CNN, evaluating them based on the number of layers, speed, accuracy, and computational cost. The backbones analyzed include ZFNet, VGG-16, ResNet-50, and ResNet-101:

Table 2.1: Comparison of Backbones for Faster R-CNN.

Backbone	Number of Layers	Speed	Accuracy	Computational Cost
ZFNet	8	Moderate	Moderate	Low
VGG-16	16	Slow	High	High
ResNet-50	50	Fast	Very High	Moderate
ResNet-101	101	Moderate	Highest	High

2.5.3 Detectron2 Implementation of Faster R-CNN.

Detectron2 is an open-source project developed by Facebook AI Research that provides state-of-the-art detection and segmentation algorithms. It is built on top of PyTorch and offers speed, accuracy, modularity, and customizability advantages [18].

Key Features of Detectron2:

- **Pre-trained Models:** Detectron2 provides pre-trained models through its Model Zoo, which can be used as a starting point for developing custom applications.

- **Feature Pyramid Network (FPN):** Some models in Detectron2 use the FPN technique, which extracts features at different layers of the model's architecture, enhancing accuracy.

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model | metrics
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model | metrics
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model | metrics
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model | metrics
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model | metrics
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model | metrics
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model | metrics

Figure 2.10: Pre-trained Faster R-CNN models for object detection [19].

This list details the model's name (which reflects its architecture), training and inference times, memory requirement, and accuracy (box AP or average precision). It links to download the model configuration file and weights.

Strengths of Detectron2:

- High flexibility and modularity for customisation.
- State-of-the-art pre-trained models available for various tasks.
- Optimized for performance on single and multiple GPUs, allowing for efficient training and inference.
- Supports a wide range of backbone architectures.

Weaknesses of Detectron2:

- Requires substantial computational resources for training.
- Complex configurations and fine-tuning may be needed for specific tasks.

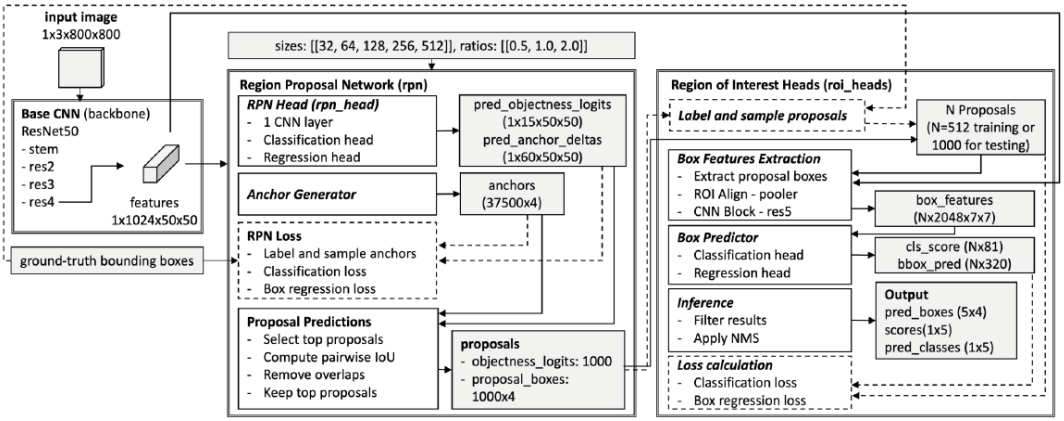


Figure 2.11: Architecture of Detectron2's implementation of Faster R-CNN [19].

The backbone network includes several convolutional layers that help to perform feature extraction from the input image. The region proposal network is another neural network that predicts the proposals with objectness and locations of the objects before feeding to the next stage. The region of interest heads have neural networks for object localisation and classification. However, the implementation details of Detectron2 are more involved. We should understand this architecture in depth to know what Detectron2 configurations to set and how to fine-tune its model.

2.6 Evaluation Metrics.

The object detection task uses two main evaluation metrics: mAP@0.5 (or AP50) and F1-score (or F1). The former is the mean of average precisions (mAP) at the intersection over the union (IoU) threshold of 0.5 and is used to select the best models.

Standard Evaluation Metrics [9], [10]:

- **Precision:** Measures the accuracy of positive predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.3)$$

- **Recall:** Measures the completeness of the positive predictions.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.4)$$

Here, TP (True Positive) represents the number of predicted objects that are correctly classified and localised (based on the IoU threshold), FP (False Positive) represents the number of predicted objects that are incorrectly classified or localised, and FN (False Negative) represents the number of ground-truth objects that are not correctly predicted.

- **F1-score:** Balances the precision and recall and measures how a model performs on a specific dataset. It can be computed using the following formula:

$$\text{F1 score} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \quad (2.5)$$

- **mAP (Mean Average Precision):** mAP@0.5 is often used to select a general model that generalises well to unseen data.

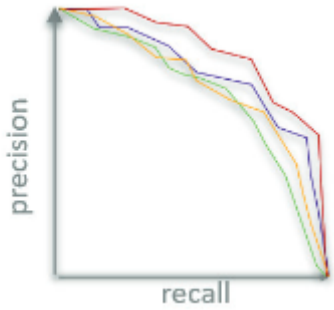


Figure 2.12: mean Average Precision at IoU=0.5 (mAP@0.5) [9].

- **IoU (Intersection over Union):** Determines the degree of similarity between a predicted region and the actual region by calculating the area of overlap

between both. An IoU threshold is typically set to determine whether a prediction is considered a true positive or a false positive.

Figure 2.13: Computation of Intersection over Union (IoU) [19].

- **Loss Curve:** The loss curve provided insight into how well the model was learning over time. A decreasing trend in the loss curve indicated improvement, showing that the model effectively minimised the loss function during training.

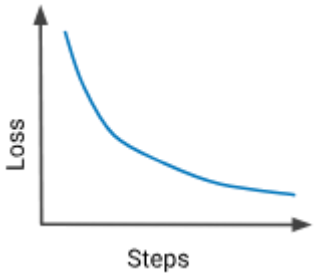


Figure 2.14: Loss Curve [20].

- **Confusion Matrix:** The confusion matrix is a tool used to evaluate a model's performance. It is visually represented as a table and provides a deeper insight into the model's performance, errors, and weaknesses. By highlighting true positives, true negatives, false positives, and false negatives, the confusion matrix allows data practitioners to analyse their model further by fine-tuning it.

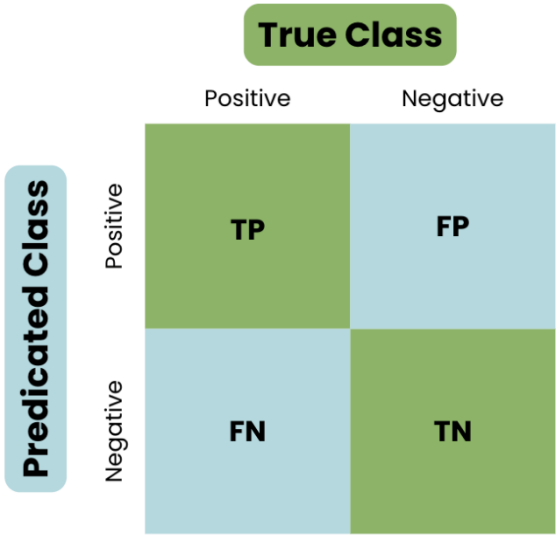


Figure 2.15: Confusion Matrix [21].

1. **True Positive (TP):** Your model correctly predicted the presence of a mango. For example, you identify a mango in an image containing a mango.
2. **True Negative (TN):** Your model correctly predicted the absence of a mango. For example, an image without a mango can be identified as not containing a mango.
3. **False Positive (FP):** Your model incorrectly predicted the presence of a mango. For example, an image without a mango can be identified as containing a mango.
4. **False Negative (FN):** Your model incorrectly predicted the absence of a mango. For example, they identify an image with a mango as not containing a mango.

2.7 Data Preparation

Data preparation is a crucial step in developing deep learning models. It involves collecting, annotating, and preprocessing data to ensure the model is trained on high-quality, relevant information. This section covers the tools and formats used for data annotation and preprocessing techniques.

2.7.1 Data Annotation

Data annotation is the process of labelling data for training machine learning models. Accurate annotation is critical for training robust object detection models.

Annotation Tools:

- **RoboFlow:** RoboFlow is the most advanced tool in the market, providing a comprehensive platform for labelling, preprocessing, and managing datasets for machine learning projects.
- **PychetLabeller:** PychetLabeller is a specialised tool used to annotate large-scale datasets efficiently.
- **labelImg and labelme:** Open-source, lightweight, fast, easy-to-use, Python-based applications.
- **VGG (Visual Geometry Group) Image Annotator (VIA):** A lightweight, standalone, browser-based web application.

Annotation Formats:

- **PASCAL VOC CSV Format:** Used for Faster R-CNN.
- **YOLO TXT:** Used for YOLO models.
- **COCO JSON:** Used for Faster R-CNN and Detectron2.

Table 2.2: COCO JSON Structure [19].

Section	Description
info	Provides general descriptions of the dataset.
licenses	A list of licenses applied to the images.
categories	A list of the available categories (or class labels) and supercategories for this dataset.
images	A list of image elements, each with information such as the id, width, height, and file_name.
annotations	A list of annotations, each with information such as segmentation and a bounding box (bbox).

Table 2.3: YOLO TXT Structure.

Element	Description
class ID	A unique identifier for each class label.
Bounding box coordinates	Normalised coordinates, including the bounding box's centre (x, y), width, and height.

Table 2.4: PASCAL VOC Format Structure.

Element	Description
file name	The name of the image file.
Bounding box coordinates	The coordinates include the bounding box's top-left corner (x1, y1) and bottom-right corner (x2, y2).

2.7.2 Data Preprocessing

Data preprocessing involves preparing raw data for the training process. Ensuring all images are scaled uniformly and normalised can improve the model's performance.

Preprocessing Techniques:

- **Image Resizing:** Scaling images to a uniform size ensures consistency across the dataset, making it easier for the model to learn from the data.
- **Normalization:** Adjusting pixel values to a standard range (e.g., 0 to 1) can help the model converge faster and improve accuracy.
- **Data Augmentation:** Applying various transformations to the data artificially increases the training dataset's size and variability. (Discussed in the Fine-Tuning section).

2.8 Fine-Tuning

Fine-tuning is a critical process in deep learning. In this process, a pre-trained model is adapted to a new task, typically with a smaller dataset, to enhance its performance on that specific task. This section delves into the essential training techniques, data augmentation methods, and optimizers commonly used in fine-tuning.

2.8.1 Training Techniques

Practical deep learning model training requires various strategies to optimize performance and prevent overfitting. This section discusses vital training techniques such as epochs, iterations, and early stopping.

Epochs and Iterations:

- **Epochs:** An epoch refers to one complete pass through the training dataset. During training, the model processes each sample in the dataset once per epoch. Training for multiple epochs allows the model to learn from the data iteratively and improve its performance with each pass.
- **Iterations:** An iteration refers to one update of the model's parameters. It is typically defined by the number of batches the training dataset is divided into. For example, if a dataset has 1000 samples and the batch size is 50, then one epoch consists of 20 iterations (1000/50).

Early Stopping:

Early stopping is a regularisation technique used to prevent overfitting during training. It involves monitoring the model's performance on a validation set and stopping the training process when the performance ceases to improve. This technique ensures the model fits the training data and generalises unseen data well. [19].

Benefits of Early Stopping:

- **Prevents Overfitting:** Early stopping helps maintain the model's ability to generalise to new data by stopping the training process before the model begins to overfit.
- **Saves Time and Resources:** Early stopping can reduce the training time by halting the process once no significant improvements are observed, thus saving computational resources.

2.8.2 Learning Rate

The learning rate (LR) is one of the most critical hyperparameters in training deep learning models. It controls the size of the optimizer's steps when updating the model's weights. A well-chosen learning rate can significantly affect the speed and quality of the training process.

Impact of Learning Rate:

- **Too High Learning Rate:** If the learning rate is too high, the model's weights may change too drastically, leading to instability and divergence from the optimal solution. This can result in poor convergence and high loss.
- **Too Low Learning Rate:** If the learning rate is too low, the training process may become excessively slow, as the model makes minimal weight updates. This can lead to getting stuck in local minima and failing to converge to the best solution.
- **Optimal Learning Rate:** An optimal learning rate strikes a balance, allowing the model to converge efficiently to a reasonable solution without overshooting or slow convergence.

Standard Learning Rate Techniques:

1. **Fixed Learning Rate:** The learning rate remains constant throughout training. This approach is simple but may not be ideal for all scenarios.

2. **Learning Rate Schedules:** The learning rate is adjusted at predefined intervals or epochs. Common schedules include:
    - **Step Decay:** The learning rate is reduced by a factor at specific epochs.
    - **Exponential Decay:** The learning rate decreases exponentially over epochs.
    - **Polynomial Decay:** The learning rate decreases following a polynomial function over epochs.
3. **Adaptive Learning Rates:** Algorithms like Adam and RMSprop automatically adjust the learning rate for each parameter, providing faster convergence and often better performance.

Benefits of Learning Rate Techniques:

- **Improved Convergence:** Dynamic learning rate adjustment can help the model converge more efficiently to an optimal solution.
- **Reduced Overfitting:** Gradually reducing the learning rate can help fine-tune the model, reducing the risk of overfitting.
- **Enhanced Performance:** Adaptive learning rates can lead to better performance on the validation set, resulting in a more robust model.

2.8.3 Data Augmentation.

Data augmentation is a technique used to artificially increase the size and variability of a training dataset by applying various transformations to the original data. This process helps to improve the robustness and generalisation ability of deep learning models by exposing them to a broader range of variations during training. In object detection, data augmentation is significant due to the high variability in object appearances, scales, and orientations [9].

Common Data Augmentation Techniques:

- **Horizontal and Vertical Flipping:** Flipping images horizontally or vertically to create mirror images.
- **Rotation:** Rotating images by a certain angle introduces variations in object orientation.
- **Scaling and Cropping:** Randomly scaling and cropping images to simulate different distances and viewpoints.
- **Translation:** Shifting images horizontally or vertically to alter the position of objects within the frame.
- **Color Jittering:** Adjust the brightness, contrast, saturation, and hue of images to mimic different lighting conditions.

Benefits of Data Augmentation:

- **Improved Generalization:** By training on a more diverse set of images, the model learns to generalise unseen data better.
- **Reduced Overfitting:** Data augmentation reduces the risk of overfitting by preventing the model from learning spurious correlations in the training data.
- **Enhanced Robustness:** Exposure to various transformations makes the model more robust to changes in object appearance and environmental conditions.

2.8.4 Optimizers

Optimizers are algorithms used to update the weights of a neural network during training by minimising the loss function. They are crucial in how effectively and quickly a model converges to the optimal solution.

Common Optimizers:

- **Stochastic Gradient Descent (SGD):** One of the simplest and most widely used optimization algorithms. It updates the model parameters by computing

the gradient of the loss function with respect to the parameters. Despite its simplicity, SGD can be slow and may get stuck in local minima [19].

- **SGD with Momentum:** An improvement over standard SGD that helps accelerate gradients in the right direction, leading to faster convergence. It accumulates past gradients to determine the direction to move in [19].
- **Adam (Adaptive Moment Estimation):** computes adaptive learning rates for each parameter, maintaining exponentially decaying averages of past gradients and squared gradients. It is known for its fast convergence and effectiveness in handling sparse gradients.
- **AdamW (Adam with Weight Decay):** A variant of Adam that decouples the weight decay (L2 regularisation) from the gradient update. This allows for better handling of regularisation and can lead to better generalisation performance. AdamW includes a weight decay term, helping prevent overfitting by effectively applying regularisation.

Table 2.5: Summary of Optimizers.

Optimizer	Advantages	Disadvantages
SGD	Simple, easy to implement	It can be slow and may get stuck in local minima
SGD with Momentum	Faster convergence reduces oscillations	Requires tuning of momentum term
Adam	Adaptive learning rates, fast convergence	Computationally expensive, requires tuning
AdamW	Better generalisation, effective regularisation	It is slightly more complex and requires tuning

2.9 Deploying Models with ONNX

ONNX (Open Neural Network Exchange) is an open-source format designed to represent and share deep learning models across different frameworks. This versatility allows models deployed on various platforms, such as servers and mobile devices, that support these frameworks. [19], [22].

2.10 GPS Integration for Efficient Orchard Management

For efficient orchard management, the detected mangoes' GPS coordinates are visualised on a map. This allows farmers to pinpoint the exact locations of detected mangoes, facilitating targeted harvesting and resource allocation. Integrating GPS data with object detection results enhances the system's practical utility, enabling precise management of agricultural resources and improving overall productivity.

Critical Benefits of GPS Integration:

- **Targeted Harvesting:** This technique enables farmers to identify and locate mangoes accurately, reducing the time and effort required for harvesting.
- **Resource Allocation:** Helps optimize resources such as labour, water, and fertilisers by focusing on specific areas within the orchard.
- **Improved Monitoring:** This technology facilitates continuous monitoring of mango orchards, providing valuable insights into the health and status of the crops.

2.11 Related Work in Mango Detection

The detection and localisation of mango fruits from aerial images have gathered significant attention in recent years due to the potential benefits of precision agriculture. Various studies have explored different methodologies and technologies to enhance the accuracy and efficiency of mango detection.

2.11.1 Deep Fruit Detection in Orchards

Bargoti and Underwood (2017) employed Faster R-CNN for fruit detection in orchards, including mangoes. Their approach involved using a Region Proposal Network (RPN) that shares convolutional layers with the detection network, leading to high detection accuracy. This study demonstrated that Faster R-CNN could effectively detect mangoes with F1-scores exceeding 0.9, highlighting the model's robustness in

handling high-resolution images and significant variations in fruit appearance and environmental conditions. [13].

2.11.2 Visual Detection of Green Mangoes by an Unmanned Aerial Vehicle in Orchards Based on a Deep Learning Method

Alzubaidi et al. (2021) explored the development of a method for visually detecting green mangoes in orchards using unmanned aerial vehicles (UAVs). They employed the YOLOv2 deep learning model for its rapid detection capabilities. The methodology involved capturing mango images through UAVs, followed by manual annotation to create precise training and test datasets, which were then used to fine-tune the YOLOv2 model's parameters. The results were promising, showing effective learning and improved performance in mango detection, as indicated by high precision (0.961) and recall (0.890). However, the study's dependence on the older YOLOv2 model highlighted a limitation, suggesting that integrating more recent YOLO models could enhance accuracy and robustness. [23].

2.11.3 Detection and Localization of Farm Mangoes Using YOLOv5 Deep Learning Technique

Ranjan and Machavaram (2022) presented a study on detecting and localising farm mangoes using the YOLOv5 deep learning technique. This research focused on addressing the complexities of mango fruit detection in natural farm environments, such as occlusion, shadow, and variable lighting, which are vital for yield monitoring and harvesting automation. They trained the YOLOv5 model on a dataset of 150 mango images using parameters like 20,000 training steps and batch size 64. The developed model detected and localised farm mangoes with a mean average precision (mAP) of 0.434 and an F1 score of 0.57, demonstrating 94.3% accuracy in test images and 80.76% accuracy in real-time video detection. Despite its advancements, the study identified room for improved detection accuracy under varying farm conditions. [24].

2.11.4 Computer Vision System for Mango Fruit Defect Detection Using Deep Convolutional Neural Network

Nithya et al. (2022) proposed a CNN-based automatic mango defect detection system. Their research utilised 100 images of Kent mangoes, augmented to 800 images through data augmentation techniques. The CNN model was trained and tested on this dataset, achieving a classification accuracy of 98.5%. Despite its effectiveness in identifying defects, a fundamental limitation noted was the computational intensity of the training process and the need for a large amount of data to train the model effectively. [25].

2.11.5 Real-Time Ripe Fruit Detection Using Faster R-CNN Deep Neural Network Models

Pham (2023) investigated ripe fruit detection using the Faster R-CNN model, integrating VGG16, ResNet50, and InceptionV2 architectures. The study used a Fruits-360 dataset, partitioned into training (60%), validation (20%), and test (20%) groups, focusing on 300 images with data augmentation. It achieved a mean average precision (mAP) of 0.869. The research highlights the trade-off between speed and accuracy in CNN architectures, noting Faster R-CNN's higher accuracy but greater computational demands. The main limitation lies in the computational complexity and slower processing speed of the two-stage Faster R-CNN approach, suggesting a need for improved efficiency in real-time agricultural scenarios. [11].

Table 2.6: Comparison of Key Aspects in Mango Detection.

Aspect	Faster R-CNN (Bargoti & Underwood, Pham)	YOLO (Alzubaidi, Ranjan & Machavaram)	CNN (Nithya et al.)
Speed	Slower, two-stage process	Faster, real-time detection	Moderate
Accuracy	High (mAP ~0.869)	Moderate to high (mAP ~0.434)	High (98.5%)
Computational Demand	High	Moderate	High
Application	Ground-based	Aerial (UAV)	Ground-based
Limitations	Computational complexity, limited angles	Earlier versions are less accurate; there is room for improvement	High data requirements, computational intensity

2.12 Research Gaps and Project Contribution

This section identifies the critical research gaps in mango fruit detection using aerial imagery and outlines the proposed solutions to address these gaps. Table 2.7 This paper summarises these research gaps, their specific issues, and the corresponding solutions. The table highlights three primary areas of concern: Data Quality and UAV Integration, Small-Object Detection and Generalization, and GPS Integration. By addressing these gaps, this research aims to improve the effectiveness and accuracy of mango fruit detection systems.

Table 2.7: Research Gaps and Proposed Solutions.

Research Gap	Issue	Solution
Data Quality and UAV Integration	Insufficient quantity and diversity of high-quality datasets from drones.	Enhance data collection using UAVs to create comprehensive and varied aerial imagery datasets, including annotations for mango varieties and ripeness stages.
Small-Object Detection and Generalization	Difficulty detecting small and occluded mangoes; ensuring models are robust across varying conditions.	Implement multi-scale feature extraction and advanced data augmentation techniques, leveraging transfer learning to enhance generalisation.
GPS Integration	Lack of GPS data integration with detection results for efficient resource allocation and viewing on a map.	Combine GPS data with detection results to facilitate targeted harvesting, resource allocation, and visualisation on a map.

CHAPTER 3 METHODOLOGY

This chapter outlines the comprehensive methodology employed to develop and evaluate a mango fruit detection system using aerial images and deep learning models. The methodology encompasses data collection, dataset preparation, model architectures, training procedures, and performance evaluation metrics. The research design follows a systematic approach to the project's objectives of developing accurate and efficient mango detection models.

3.1 System Flowchart

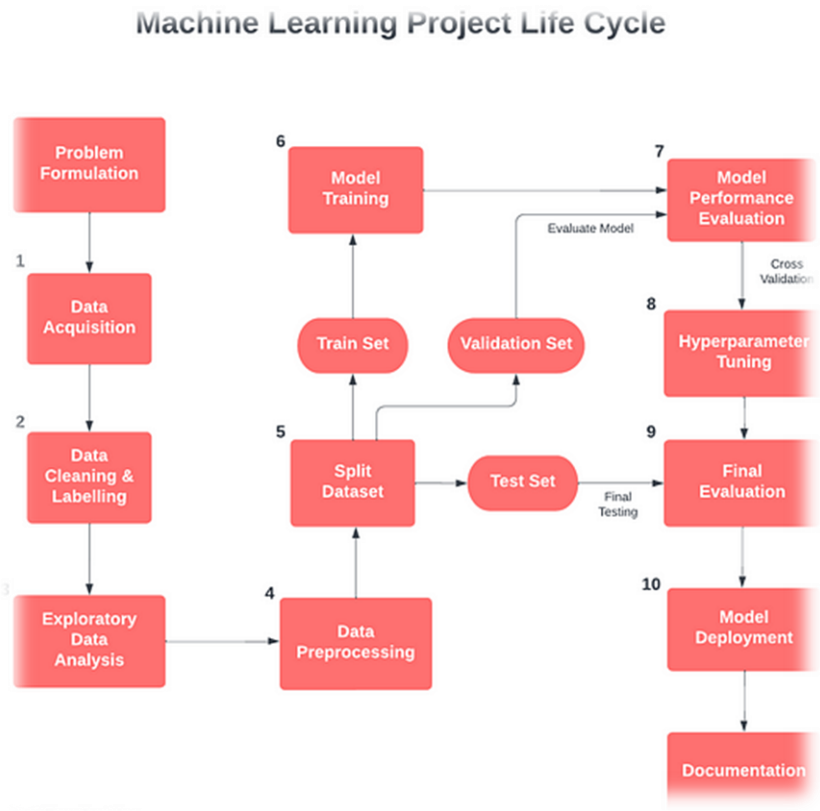


Figure 3.1: Machine Learning Project Life Cycle [26].

Figure 3.1 outlines the systematic process of the project's implementation. It begins with 'Problem Formulation' to define the project's goals, followed by 'Data Acquisition' to collect aerial images using drones. 'Data Cleaning & Labelling' involves annotating

images with bounding boxes around mango fruits. 'Exploratory Data Analysis' helps understand data patterns, while 'Data Preprocessing' prepares the data for model training by resizing and normalising images. The dataset is then split into training, validation, and test sets. 'Model Training' utilises the training data, followed by 'Model Performance Evaluation' to assess accuracy using validation data. 'Hyperparameter Tuning' optimizes model parameters for better performance, leading to a 'Final Evaluation' of the test set. The project concludes with 'Model Deployment' for real-time detection and 'Documentation' of the entire process for future reference. Finally, the findings and methodologies are shared through 'Publication' in relevant academic journals or conferences. This structured approach ensures thorough planning and execution at each stage, leveraging deep learning models for precise mango fruit detection from aerial images

3.2 Experimental Setup

The experimental setup for this research project was meticulously designed to ensure optimal performance and reliability in mango detection tasks.

3.2.1 Hardware Configuration

The computational backbone of this study is a high-performance workstation, specifically a Lenovo Legion 5 laptop. Table 3.1 Provides an overview of the hardware specifications.

Table 3.1: Hardware Specifications.

Component	**Specification**
Laptop	Lenovo Legion 5
Processor	AMD Ryzen 7 5800H with Radeon Graphics, 3.20 GHz
RAM	16.0 GB
Graphics Card	Nvidia RTX 3060

The system is equipped with an AMD Ryzen 7 processor, which provides robust multi-core processing capabilities essential for deep learning tasks. An NVIDIA RTX 3060

graphics card enables accelerated GPU computations, significantly enhancing the training speed of our neural network models.

3.2.2 Software Environment

The software environment was carefully well-chosen to support advanced deep-learning operations. Table 3.2 Presents a comprehensive list of the critical software components and their respective versions.

Table 3.2: Software Specifications

Software	Version	Reference
Nvidia CUDA Toolkit	11.8	[27]
Python	3.10.9	[28]
PyTorch	2.0.1+cu117	[29]
Ultralytics YOLO	8.1.34	[30]
Detectron2	0.6	[31]
VS Code	1.91.0	[32]

We employed two primary tools for object detection tasks: Ultralytics YOLOv8 version 8.1.34 and Detectron2 version 0.6. These frameworks were chosen for their state-of-the-art performance in object detection tasks and compatibility with our research objectives.

3.2.3 Development Environment

The project utilised Visual Studio Code (VSCode) as the primary integrated development environment due to its support for Jupyter Notebooks and extensive extensions for Python and machine learning projects. Google Colab was also employed for specific experiments, leveraging its cloud-based GPU resources.

CUDA stands for Compute Unified Device Architecture. The Nvidia CUDA toolkit provides a development environment for creating high-performance GPU-accelerated applications. It is essential for leveraging the GPU's computational power in training and running deep learning models. This setup phase lays the groundwork for efficient

data processing, model training, and evaluation in the subsequent stages of the project. [27].

This dual-environmental approach allowed for flexibility in our development process, enabling us to optimize our use of computational resources based on the specific requirements of each experimental phase. The setup phase laid the groundwork for efficient data processing, model training, and evaluation in the subsequent stages of the project.

3.3 Datasets

This study utilised three datasets for mango detection: a local dataset, a dataset from CQUniversity Rockhampton North, Australia, and a dataset from The University of Sydney, Australia. Table 3.3 Summarises the composition of each dataset.

Table 3.3: Summary of the datasets used in the project.

Dataset	Total Images	Training Images	Validation Images	Test Images
Local Dataset	118	83	24	11
CQUniversity	536	452	11	73
The University of Sydney	1964	1464	250	250

A structured organisation into folders for images, annotations, and dataset splits (train, test, validation). The local dataset was prepared using RoboFlow and collected for the project. [33]The CQUniversity dataset, found online, was initially used for segmentation tasks. However, a part of this dataset was repurposed for object detection for this project. Using RoboFlow, the dataset was re-split into new training, validation, and test sets to suit this study's needs better. This allowed for a more balanced distribution of images and ensured that the data was annotated correctly and ready for training the object detection models. [34]. The primary dataset used for training the Faster R-CNN model was from The University of Sydney. [13] .

Sample images from each dataset:

Figure 3.2: The University of Sydney Dataset Sample.

Figure 3.2 This is a sample image from the University of Sydney primary training dataset. The dataset contains many images, providing a robust model training and evaluation foundation.

Figure 3.3 This is a sample image from the CQUniversity dataset initially used for segmentation tasks. The dataset was repurposed and re-split using RoboFlow to fit the needs of this study, facilitating accurate object detection model training.

Figure 3.3: CQUniversity Dataset Sample

Figure 3.4: Local Dataset Sample

Figure 3.4 shows a sample image from the local dataset, specifically collected and prepared using RoboFlow for this project. Despite its smaller size, this dataset

contributed valuable data to enhance the training and evaluation of the mango detection models.

3.4 Data Collection and Labeling

The local dataset was collected using a DJI Mini 3 drone, which captured high-resolution aerial images of mango orchards. Each image was annotated to identify mango fruits, using bounding boxes to delineate the location of each fruit. The CQUniversity and University of Sydney datasets were obtained from publicly available sources, with annotations already provided.

The labelling process involved the following steps:

- **Image Acquisition:** High-resolution aerial images were captured using a drone.
- **Annotation:** Each fruit was marked with a bounding box and assigned a class label.
- **Dataset Split:** The annotated datasets were split into training, validation, and test sets, ensuring a balanced distribution of images across each set.

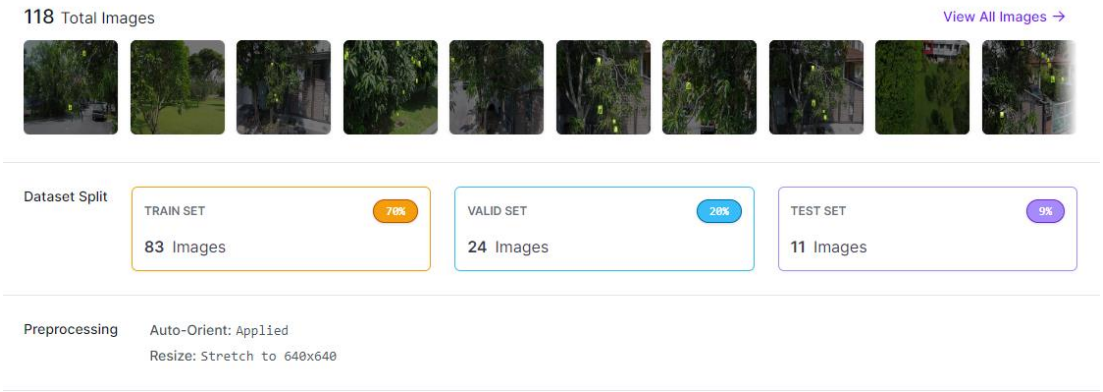


Figure 3.5: Illustrates the distribution of images in the training, validation, and test sets for the local dataset after annotation on RoboFlow.

Before training the Faster R-CNN models, we prepared a combined CSV file annotation format to streamline the training process and ensure consistency across

different experiments. This combined annotation file includes the following information:

- Image file names
- Bounding box coordinates (x, y, dx, dy)
- Class labels (in this case, 'mango')

#item	x	y	dx	dy	label
0	275.2631579	53.68421053	36.2398272	38.8550875	1
0	341.4285714	287.1428571	56.02620255	58.52263859	1
1	438.0952381	309.5238095	56.02620255	70.81239269	1
2	421.9047619	373.3333333	50.42358229	63.73115342	1
3	452.3809524	442.3809524	44.03335633	55.65444699	1
0	266.4705882	359.4117647	17.2327251	20.33762724	1
0	455.2941176	165.2941176	40.07298388	67.18531057	1

Figure 3.6: The snippet of the Sydney dataset combined with the CSV annotation file.

Figure 3.6 Shows the snippet of the Sydney dataset combined with the CSV annotation file. This combined CSV annotation format effectively streamlined the training process, allowing for efficient data handling and consistent performance evaluation across different model architectures. The figure illustrates how the combined annotation format organises essential information, facilitating the training and evaluation of Faster R-CNN models. This structured approach ensures that all data is consistently formatted, contributing to the reliable performance of the models across various experiments.

The annotations for Sydney and local datasets at detectron2 were stored in JSON format, containing essential information for each detected object.

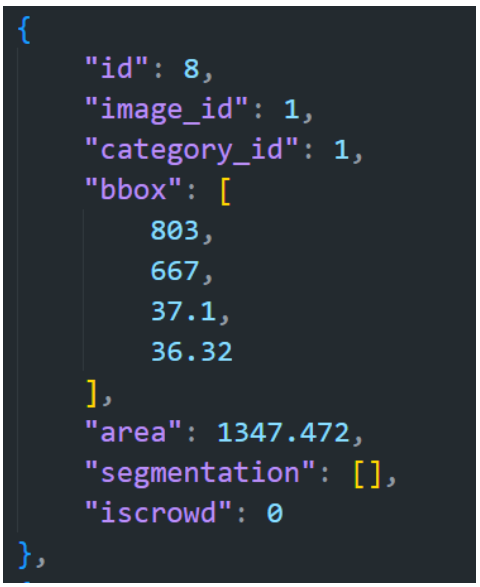


Figure 3.7: A sample entry from the local dataset json annotation file.

As shown in Figure 3.7 Figure 3.7, each annotation entry includes:

- **"id"**: A unique identifier for the annotation (8 in this example).
- **"image_id"**: The ID of the image containing this annotation (1 in this case).
- **"category_id"**: The category of the detected object (1, presumably representing mangoes).
- **"bbox"**: Bounding box coordinates [x, y, width, height] (803, 667, 37.1, 36.32).
- **"area"**: The area of the bounding box (1347.472 square pixels).
- **"segmentation":** An empty array in this case, indicating no pixel-wise segmentation data.
- **"iscrowd":** A boolean flag (0 in this case) typically used to indicate whether the annotation represents a single object or a group of objects.

This structured annotation format allows efficient data loading, processing, and evaluation during model training and testing. It provides a standardised way to represent object locations and properties within images, which is crucial for training and evaluating object detection models like Faster R-CNN.

```
0 0.3263975 0.13823555555555556 0.0376775 0.06840444444444445
0 0.34563499999999997 0.18003999999999998 0.038744999999999995 0.06840444444444445
0 0.3740925 0.39855999999999997 0.0438225 0.07410666666666667
0 0.355255 0.39380888888888893 0.02966 0.07885777777777778
```

Figure 3.8: An example of a Yolo text annotation image file.

This file represents the annotations for objects detected in an image formatted for YOLO training. Each line corresponds to a single object and contains the following information:

The first number represents the class of the object (in this case, 0 for mango). The following four numbers represent the bounding box coordinates normalised to the width and height of the image:

- ‘x_center’ is the x-coordinate of the centre of the bounding box.
- ‘y_center’ is the y-coordinate of the centre of the bounding box.
- ‘width’ is the width of the bounding box.
- ‘height’ is the height of the bounding box.

These annotations are crucial for training the YOLO model as they provide the necessary data to learn how to detect and localise objects within images.

3.5 Data Preparation

The datasets were converted into formats suitable for the respective models. Three formats were used in this project: CSV, TXT, and JSON. The conversion processes ensured compatibility and consistency across different models and frameworks.

RoboFlow was used to directly convert the annotations into the required formats for the local and CQUniversity datasets. However, manual verification and fixing of annotations were necessary to ensure accuracy. RoboFlow facilitated the conversion

of annotations to YOLO format and COCO JSON format, and the datasets were split into training, validation, and test sets using RoboFlow.

For the University of Sydney dataset, custom scripts were used to convert the annotations between different formats to ensure compatibility with various models:

- CSV to YOLO TXT: The annotations were initially provided in CSV format. After processing each CSV file, the bounding box coordinates were converted to YOLO TXT format.
- YOLO TXT to COCO JSON: To maintain flexibility between models, annotations in YOLO TXT format were also converted to COCO JSON format.

The process involved reading the CSV files, extracting the bounding box coordinates and class labels, and creating the respective YOLO TXT and COCO JSON objects with the necessary fields. The function converted dataset splits (train, validation, test) to COCO format, ensuring images and annotations were correctly converted and saved in separate JSON files for each split.

In all cases, the accuracy and consistency of annotations were ensured through thorough verification and fixing processes, which included cross-checking with original images, correcting errors in bounding boxes and labels, and updating annotation files where needed.

3.6 Model Implementation

The models were implemented using YOLO (You Only Look Once) and Faster R-CNN (Region-based Convolutional Neural Networks) frameworks. The implementation covered dataset preparation, model training, fine-tuning, and evaluation.

3.6.1 YOLO Implementation

The YOLO models (YOLOv8 and YOLOv10) were utilised for object detection tasks across three datasets: CQUniversity, The University of Sydney, and a local dataset. The implementation details for YOLOv8 and YOLOv10 are provided, highlighting the differences and specific configurations used during training.

Variants Selection: The following tables show different versions of YOLOv8 and YOLOv10, along with their performance metrics. This study chose the "n" versions (YOLOv8n and YOLOv10n) due to their balance between performance and computational efficiency.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 3.9: YOLOv8 Versions.

Model	Test Size	#Params	FLOPs	AP^{val}	Latency
YOLOv10-N	640	2.3M	6.7G	38.5%	1.84ms
YOLOv10-S	640	7.2M	21.6G	46.3%	2.49ms
YOLOv10-M	640	15.4M	59.1G	51.1%	4.74ms
YOLOv10-B	640	19.1M	92.0G	52.5%	5.74ms
YOLOv10-L	640	24.4M	120.3G	53.2%	7.28ms
YOLOv10-X	640	29.5M	160.4G	54.4%	10.70ms

Figure 3.10: YOLOv10 Versions.

Initial Training: The models were trained initially for 50 epochs to establish a baseline performance. The training process involved setting up paths for training, validation, and test datasets, creating a '`data.yaml`' file specifying the dataset paths and class names, and initialising the YOLO models with pre-trained weights.

Fine-tuning: Training epochs were increased, and early stopping was implemented to prevent overfitting. Data augmentation techniques such as hue adjustment, saturation changes, rotation, translation, scaling, and flipping were applied to enhance the model's robustness. The learning rate was fine-tuned to optimize model performance further.

This table summarises the data augmentation techniques used in the project to increase diversity and robustness in the training data.

Table 3.4: Data Augmentation Techniques Used and Their Descriptions.

Augmentation Technique	Description
hsv_h	Adjusts the image's hue by a fraction, adding variety to the colour spectrum.
hsv_s	Changes the saturation levels, making colours more or less vivid.
hsv_v	Alters the brightness (value) of the image, simulating different lighting conditions.
degrees	Rotates the image by a specified number of degrees, introducing variation in object orientation.
translate	Shifts the image horizontally and vertically, changing the object's position within the frame.
scale	Scales the image up or down, simulating different distances from the camera.
shear	Shears the image by a specified degree, distorting it to introduce geometric variations.
fliplr	Horizontally flips the image with a given probability, creating mirror images.
mosaic	Combines four images into one during training, providing more complex training samples.
mixup	Blends two images, creating a new training sample that enhances model generalisation.

Optimizer: The optimizer was switched to stochastic gradient descent (SGD) to evaluate its impact on model performance.

Evaluation: Evaluation was done automatically using the Ultralytics library, making the YOLO models the best experience and most straightforward to configure, build, and achieve great results.

3.6.2 Faster R-CNN Implementation

Faster R-CNN models with VGG-16 and ResNet-50 backbones were used for object detection.

1. **Subset of the Dataset**: A subset of 500 images was randomly selected from the training dataset to ensure efficient use of computational resources and quicker iterations during fine-tuning. This subset was loaded into a data loader for training and evaluation.
2. **Training:** The models were trained on the prepared dataset using an SGD optimizer with a learning rate 0.001. The training lasted 20 epochs, and the models were evaluated and saved periodically. The model was set to save every two epochs to allow continuation from the last checkpoint in case of an emergency or disconnection.
3. **Evaluation:** The models were periodically evaluated on the validation set to monitor performance and prevent overfitting. Evaluation metrics included mean Average Precision (mAP) and mean Average recall (mAR).
4. **Adjust IoU Thresholds:** The cocoeval.py file was modified to adjust IoU thresholds to 0.2, aligning with the methodology described in the paper "Deep Fruit Detection in Orchards." [13].

Detectron2 was used for comprehensive object detection tasks, providing a complete pipeline for data preprocessing, model training, evaluation, and deployment.

Table 3.5: Configuration for Sydney Dataset.

Configuration Component	Details
Dataset Registration	Registered ‘sydney_train’, ‘sydney_val’, and ‘sydney_test’ datasets.
	They provided paths for the Sydney dataset in the ‘register_coco_instances’ calls.
Configuration	Output directory name: Sydney Dataset DETECTRON2 Results
	Number of iterations (‘cfg.SOLVER.MAX_ITER’): 3000
	Learning rate decay milestones (‘cfg.SOLVER.STEPS’): [15000, 18000]
	Images per batch: 4
	Base learning rate (‘cfg.SOLVER.BASE_LR’): 0.00025
	Warmup iterations: 1000
	Warmup method: linear
	Batch size per image: 256
	Evaluation period (‘cfg.TEST.EVAL_PERIOD’): 500
Optimizers	SGD, AdamW

For the Sydney dataset, the number of iterations was set to 3000 to provide sufficient time for the model to learn from the extensive dataset. The learning rate decay milestones were placed at [15000, 18000] to gradually reduce the learning rate over a more extended training period, ensuring a steady learning pace. A batch size of 4 images per iteration helps stabilise gradient updates, enhancing learning efficiency. The base learning rate of 0.00025 offers a balanced starting point, ensuring both speed and stability in training. Warmup iterations were set to 1000 to prevent significant, destabilising gradient updates at the beginning of training. The batch size per image of 256 optimizes processing a more significant number of Regions of Interest (ROIs), improving the model's robustness. The evaluation period of 500 iterations allows for a good balance between frequent evaluations and sufficient training progress.

Table 3.6: Configuration for Local Dataset

Configuration Component	Details
Dataset Registration	Registered ‘mango_train’, ‘mango_val’, and ‘mango_test’ datasets.
	They provided paths for the Sydney dataset in the ‘register_coco_instances’ calls.
Configuration	Output directory name: Local Dataset DETECTRON2 Results
	Number of iterations (‘cfg.SOLVER.MAX_ITER’): 400
	Learning rate decay milestones (‘cfg.SOLVER.STEPS’): [100, 300]
	Images per batch: 2
	Base learning rate (‘cfg.SOLVER.BASE_LR’): 0.00025
	Warmup iterations: 1000
	Warmup method: linear
	Batch size per image: 128
	Evaluation period (‘cfg.TEST.EVAL_PERIOD’): 50
Optimizers	SGD, AdamW
Learning rate schedulers	WarmupCosineLR, WarmupMultiStepLR

The local dataset's number of iterations was set to 400 to match its smaller size, ensuring efficient training without unnecessary prolongation. The learning rate decay milestones at [100, 300] provide an appropriate reduction in the learning rate during the shorter training period, preventing overfitting. A batch size of 2 images per iteration was chosen to balance memory usage and training efficiency. The base learning rate of 0.00025 remains consistent with the Sydney dataset, providing a stable starting point. Warmup iterations of 1000 helped introduce the learning rate gradually, maintaining stability. The batch size per image of 128 fits the model's capacity to the smaller dataset, preventing overfitting. The evaluation period of 50 iterations allows for closer monitoring of the model's performance, ensuring it remains on track throughout the training process.

In all implementations, the models were fine-tuned and optimized using appropriate data augmentation techniques, early stopping, and hyperparameter adjustments,

including fine-tuning the learning rate, to achieve the best possible performance on the mango fruit detection tasks.

3.7 MangoVision GUI

3.7.1 Overview

MangoVision is a graphical user interface (GUI) application designed to detect mango fruit from aerial images and videos. This application utilises the YOLO (You Only Look Once) model for object detection and provides various features for selecting media, detecting mangoes, viewing results, and saving annotated outputs. MangoVision was further enhanced to support the best overall model and the best YOLO model on the local dataset, ensuring users have access to the highest performing models for their detection tasks. This integration guarantees the most accurate and reliable mango detection results.

3.7.2 GPS Coordinates Extraction

The initial feature of MangoVision was the ability to extract GPS coordinates from image metadata and view them on an integrated map. The application reads the metadata from images using the Exifread library to extract GPS information. It processes the EXIF tags to retrieve latitude and longitude data, converting these coordinates from degrees, minutes, and seconds (DMS) format to decimals for easy mapping. Additionally, the application retrieves the original date and time when the image was captured. This feature allows users to perform spatial analysis of the detected mangoes, adding a valuable layer of functionality for agricultural and research purposes.

3.7.3 Interactive Map Display

MangoVision includes a feature to display GPS coordinates on an interactive map. Using the folium library, the application creates a map centred around the extracted coordinates and adds a marker to indicate the image location. The map is then saved as HTML content and displayed within the GUI using a WebView component. This interactive map feature enables users to visualise the geographic locations of detected mangoes effectively. The application checks for an internet connection before loading the map to ensure seamless functionality.

3.7.4 Video Processing

MangoVision supports video processing using YOLO models, allowing users to select and process a video file to detect mangoes frame by frame. The detection results are annotated on each frame, and the annotated frames are displayed in the GUI. Users can play, pause, and stop the video within the application, and the annotated video can be saved for future reference. This feature enhances the application's capability to handle various media types, making it a versatile tool for mango detection.

3.7.5 Image Processing

Users can select an image file for processing, and the application will detect mangoes within the image using YOLO models. The detected mangoes are annotated, and the annotated image is displayed in the GUI. GPS and datetime information are also extracted from the image metadata and displayed. Users can save the annotated image for future reference. This feature allows for efficient and accurate mango detection from single images, providing a valuable tool for analysis and research.

3.7.6 User-Friendly Design

To ensure user-friendliness, MangoVision is initialised using Tkinter, with a splash screen displayed during startup to enhance the user experience. The GUI window is organised with a main container and scrollable frame to dynamically handle different UI elements, making it adaptable and easy to navigate. A custom theme and icons ensure a consistent and visually appealing interface. The application supports multiple languages, dynamically allowing users to switch between English and Malay, updating all text elements in the UI accordingly, and enhancing accessibility. The language change functionality is implemented using a translation file that updates the text elements based on user selection from the toolbar.

3.7.7 IOU Threshold Adjustment

MangoVision allows users to adjust the Intersection over the Union (IoU) threshold for image and video detection. The IoU is set above 0.5 by default, but users can modify this value between 0 and 0.9 through a menu. This flexibility lets users fine-tune the detection sensitivity according to their needs.

3.7.8 Pre-trained YOLO Models

Two pre-trained YOLO models are loaded at startup: one specifically trained on the local dataset and the other trained on an online dataset. This setup allows users to compare the model's performance on the local dataset with those trained on larger, publicly available datasets. Media selection is streamlined through a file dialogue, allowing users to choose images or videos for processing easily. Depending on the file type, the application prepares the media for mango detection, providing flexibility and ease of use.

3.7.9 Detection Process

The detection process is initiated through a button click, with the YOLO model processing the selected media to detect mangoes. Detected mangoes are annotated with bounding boxes and confidence scores, and results are displayed within the application for easy review. GPS coordinates are extracted from image metadata if available, and users can view these coordinates on an integrated map.

3.7.10 Saving Annotated Results

Annotated results can be saved to disk in the desired format, ensuring users can keep their work. The application includes controls for video files to play, pause, and stop video playback, allowing users to navigate through the video frames and see the detected mangoes in action. The status bar provides real-time feedback about the application's state, keeping users informed throughout the detection process. Various utility functions are implemented to handle specific tasks, ensuring the application is robust, efficient, and user-friendly.

3.7.11 Conclusion

The development and enhancement of MangoVision showcase the potential of integrating deep learning models with user-friendly interfaces to create practical applications for real-world problems.

CHAPTER 4 EXPERIMENTS AND RESULTS

This chapter presents a comprehensive analysis of the experiments and results obtained from training and evaluating various object detection models for mango detection using aerial imagery. The study encompasses two primary models: Faster R-CNN, YOLOv8, and YOLOv10, each tested on multiple datasets, including the University of Sydney, CQUniversity, and a local dataset. The experiments were designed to evaluate the models' performance under different configurations, optimizers, and data augmentation techniques.

4.1 Faster R-CNN Model Performance

This section details the implementation and performance of Faster R-CNN models for mango detection using aerial imagery. We explore two backbone architectures, VGG-16 and ResNet-50, and analyse their performance on the University of Sydney dataset.

Table 4.1: Performance Metrics of VGG-16 and ResNet-50 Models for Mango Detection.

Model	mAP	mAR
VGG-16	0.572	0.646
ResNet-50	0.654	0.705

The ResNet-50 model outperformed VGG-16 due to its more profound architecture and residual connections, enhancing feature extraction and generalisation capabilities. This superior performance indicates the effectiveness of deeper networks in capturing intricate patterns in aerial mango imagery.

4.1.1 VGG-16 Performance Analysis

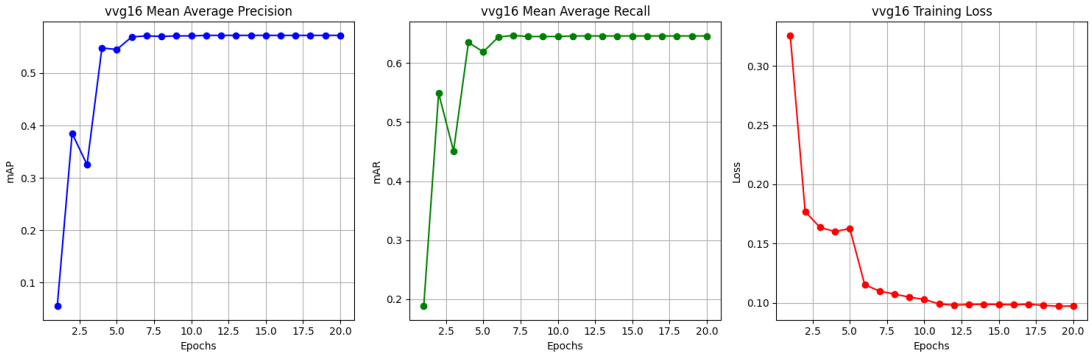


Figure 4.1: Performance curves for the VGG-16 model during training, showing mAP (blue), mAR (green), and Training Loss (red).

Key observations:

- **Rapid improvement:** Rapid improvement in Mean Average Precision (mAP), peaking early before stabilising at approximately 0.57.
- **Stable mAR:** Mean Average Recall (mAR) increases notably, stabilising at 0.646.
- **Steady decline in training loss:** Training loss decreases steadily, reflecting effective error reduction and model convergence.
- **Potential overfitting:** The plateau in loss suggests potential overfitting or exhaustion of learning from the current data, indicating areas for improvement through further training or data augmentation adjustments.

Figure 4.2: Mango detections by VGG-16 model (Red: predictions, green: ground truth).

The purpose of Figure 4.2 Is to illustrate the model's detection capability, demonstrating that the VGG-16 model can detect mangoes with reasonable accuracy. By comparing the red and green boxes, one can observe the alignment between the model's predictions and the actual locations of mangoes, providing insight into the model's performance. A close alignment between these boxes signifies accurate detections, while discrepancies highlight areas for potential improvement.

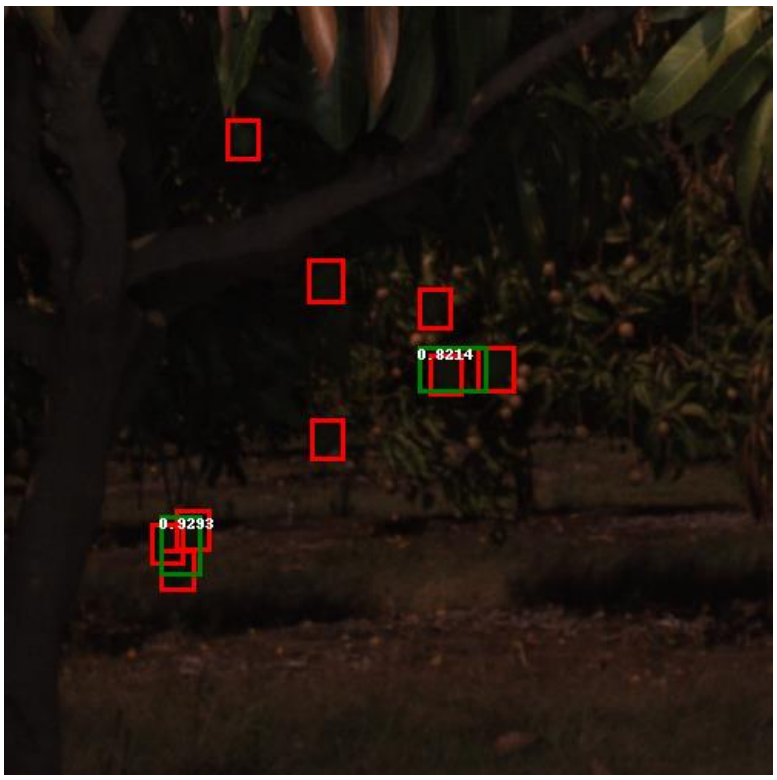


Figure 4.3: Example of mango detections using the VGG-16 model, with correct detections (green) and missed detections (red).

The figure aims to highlight the challenges the model faces, such as detecting mangoes obscured by foliage or situated at the boundaries of the image. The green boxes show where the model successfully identified mangoes, while the red boxes indicate missed detections. Additionally, the image's darkness poses an extra challenge for the model, potentially affecting its ability to detect mangoes accurately. This visual representation underscores specific difficulties the model encounters, providing valuable insights into the limitations of the VGG-16 model in different scenarios and suggesting areas for further enhancement.

4.1.2 ResNet-50 Model Performance Analysis

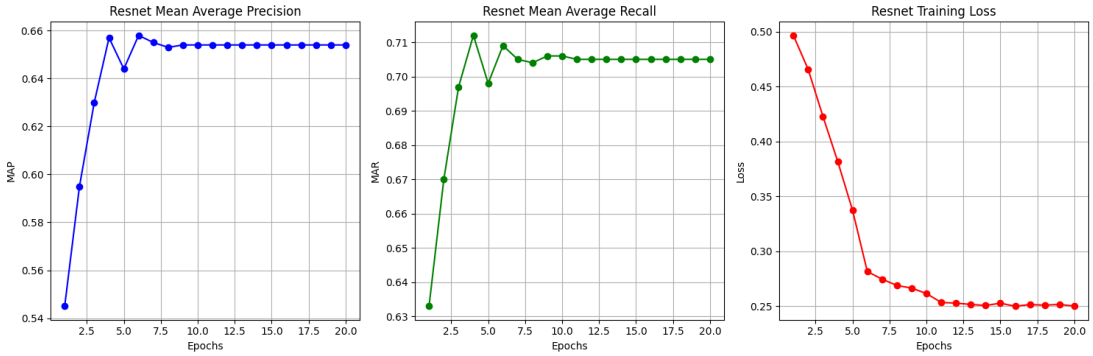


Figure 4.4: illustrates the Performance curves for the ResNet-50 model during training, showing mAP (blue), mAR (green), and Training Loss (red).

Key observations:

- **Consistent mAP increase:** Steady and consistent increase in mAP, reaching a higher stabilisation point at 0.654.
- **High mAR stabilisation:** mAR shows a robust increase and stabilises at a higher value of 0.705.
- **Significant decline in training loss:** Significant decline in training loss, maintaining a low level throughout training.
- **Superior feature extraction:** The ResNet-50 model demonstrates superior feature extraction and generalisation capability, likely due to its residual connections that alleviate the vanishing gradient problem.

Figure 4.5 demonstrates the ResNet50 model's performance in detecting mangoes, showing its ability to identify them accurately. By comparing the red and green boxes, one can evaluate how closely the model's predictions match the actual locations of mangoes. A close alignment signifies the model's effectiveness, while discrepancies

highlight areas where the model's accuracy could be improved. Compared to the VGG-16 model, ResNet50 generally shows enhanced accuracy and robustness, particularly in detecting mangoes in more challenging scenarios, such as those with complex backgrounds or varying lighting conditions.

Figure 4.5: Mango detections by ResNet50 model (Red: predictions, green: ground truth).

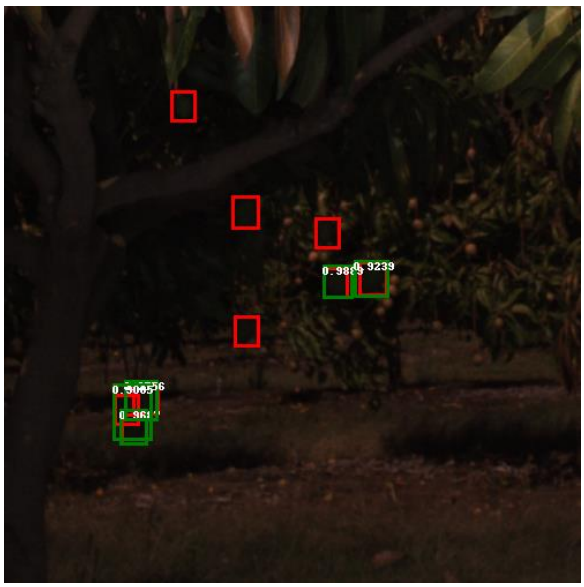


Figure 4.6: Example of mango detections using the ResNet-50 model, with correct detections (green) and false positives (red).

Figure 4.6 Emphasises the model's robustness in detecting mangoes even in challenging scenarios, such as those with occlusions or complex backgrounds. The green boxes show where the model correctly identified mangoes, while the red boxes indicate false positives, where the model incorrectly identified non-mango objects as mangoes. Compared to the VGG-16 model, the ResNet50 model demonstrates superior performance, with fewer false positives and a higher rate of correct detections. This visual representation highlights the strengths and limitations of the ResNet50 model, showcasing its effectiveness in various conditions while also pointing out incorrect detections.

4.2 Detectron2 Results Analysis

This section presents a comprehensive analysis of the results obtained using the Detectron2 framework to implement and evaluate Faster R-CNN models on both the Sydney and local datasets. The analysis focuses on the performance metrics, learning curves, and comparative studies of different optimizers for the ResNet-50 backbone architecture.

4.2.1 Results on Sydney Dataset

Two Faster R-CNN models with ResNet50 backbone were developed and evaluated on the Sydney dataset using different optimizers: Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (AdamW). The final performance metrics for both models are presented in Table 4.2.

Table 4.2: Final performance metrics for SGD and AdamW optimizers on the Sydney dataset.

Optimizer	Final Total Loss	Final AP50
SGD	0.488	93.05
AdamW	0.483	93.02

While the final performance metrics are remarkably similar, AdamW is considered the superior choice for several reasons:

- **Faster Initial Convergence:** AdamW demonstrated faster initial convergence, reaching high AP50 values earlier in training.
- **Stable Performance:** It also showed more stable performance throughout the training process, with smoother loss reduction.

These characteristics make AdamW more efficient and practical, especially when quick results or limited training time are crucial. Furthermore, AdamW's adaptive learning rate strategy allows it to handle varying feature importances more effectively, which can be particularly beneficial for complex datasets like mango detection in orchard images. Given AdamW's superior characteristics, we will analyse its learning curves for the Sydney dataset.

Figure 4.7: Total loss curve for ResNet-50 AdamW optimizer on the Sydney dataset.

The total loss curve for the AdamW optimizer on the Sydney dataset reveals interesting training dynamics. Initially, the loss starts at approximately 0.63 at iteration 500 and shows a general downward trend, indicating that the model is learning effectively. However, the curve exhibits notable fluctuations throughout the training process, which is somewhat unexpected for the AdamW optimizer. These fluctuations suggest that the model encountered challenging examples or experienced some instability during training.

Particularly noteworthy are the sharp drops in loss around iterations 1500 and 2500. These sudden improvements could be attributed to the model learning from especially informative batches of data. The loss reaches a minimum of about 0.35 at iteration 2500, demonstrating the model's best performance in minimising the objective function. By the final iteration 3000, the loss increases slightly to approximately 0.48, which might indicate a small amount of overfitting or the model finding a more generalisable solution.

AP50 Curve Analysis

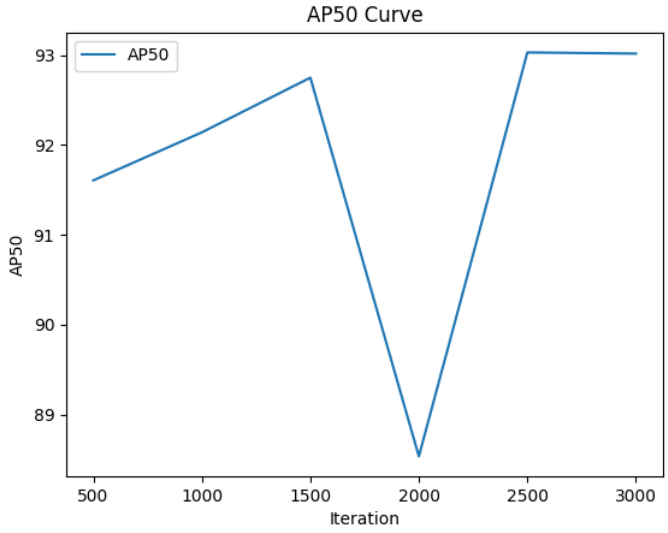


Figure 4.8: AP50 curve for AdamW optimizer on the Sydney dataset at 50% IoU.

The AP50 (Average Precision at 50% IoU) curve for the AdamW optimiser provides valuable insights into the model's detection performance over time. The curve starts at a remarkably high value of approximately 91.5 at iteration 500, indicating that the model was well-initialized and capable of good performance from the early stages of training.

From iterations 500 to 1500, there's a steady upward trend, showing consistent improvement in the model's detection capabilities. However, a significant drop occurs around iteration 2000, with the AP50 value decreasing to about 88.5. This drop correlates with a spike in the loss curve, suggesting a period of instability in the training process.

Impressively, the model demonstrates remarkable resilience by quickly recovering from this drop. It reaches its peak performance with an AP50 of approximately 93 by iteration 2500. This rapid recovery showcases the model's ability to overcome temporary performance setbacks. The AP50 value remains stable at this peak until the end of training at iteration 3000, indicating that the model has reached a robust and consistent performance state.

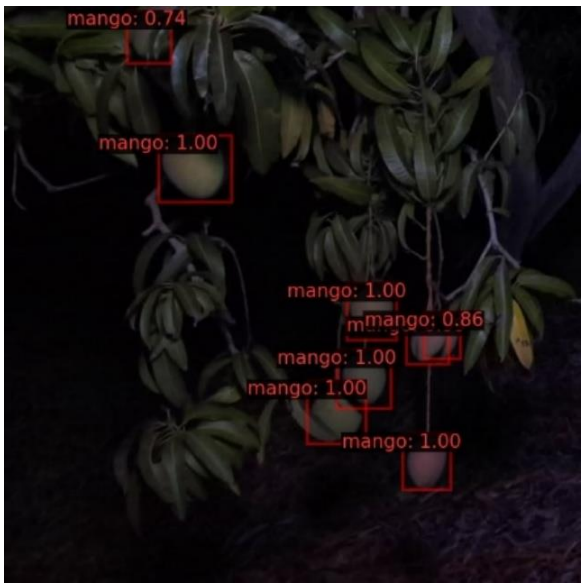


Figure 4.9: Mango Detection in the Sydney Dataset Using the Detectron2 framework demonstrates the detection capabilities.

4.2.2 Results on Local Dataset

For the local dataset, we again found that the ResNet50 model with AdamW optimizer performed the best. Let's analyse its performance metrics.

Table 4.3 : Final performance metrics for different optimizers on the local dataset.

Optimizer	Final Total Loss	Final AP50
SGD	0.913	56.24
AdamW	0.590	80.52
Cosine	0.814	68.66
MultiStep	1.098	57.74

Analysis of local dataset results:

AdamW significantly outperformed all other optimizers in both final loss and AP50. Cosine Annealing achieved the second-best performance, showing potential for longer training schedules. AdamW's superiority on the local dataset is clear and convincing:

- Highest AP50: AdamW achieved 80.52, far surpassing SGD's 56.24.
- Lowest Loss: Final loss of 0.590 for AdamW compared to 0.913 for SGD.

- Consistent Outperformance: AdamW maintained its lead throughout training.
- Weight Decay Regularization: The 'W' in AdamW stands for decoupled weight decay, which can lead to better generalisation.

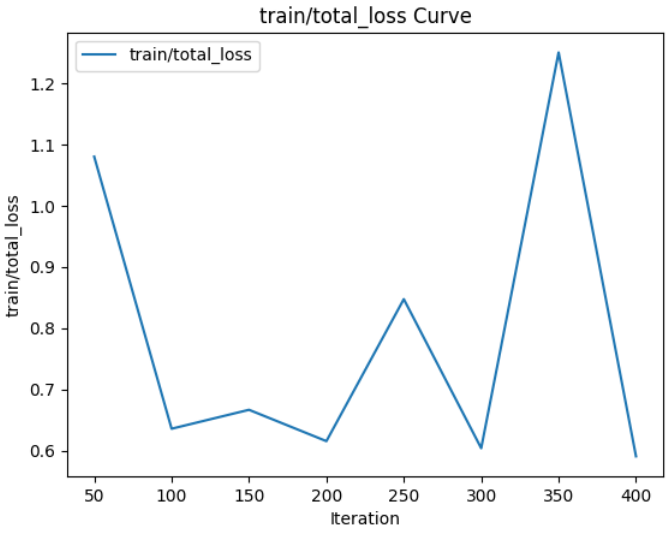


Figure 4.10: Total loss curve for the ResNet-50 model on the local dataset using AdamW.

Key observations:

- Initial Loss: The training starts with a higher loss of around 1.08 at iteration 50.
- Rapid Decrease: There's a sharp decline in loss during the early iterations.
- Final Loss: By iteration 400, the loss has significantly reduced to approximately 0.59.
- Consistent Trend: The loss curve shows a smooth downward trend.

Analysis:

The higher initial loss on the local dataset compared to the Sydney dataset suggests that this dataset may have been more challenging for the model at the outset. However, the rapid and consistent decrease in loss indicates that the model quickly adapted to the dataset's characteristics. The final loss value of 0.59, matching that of the Sydney

dataset, suggests that the model achieved comparable optimization levels on both datasets despite their differences.

AP50 Metric Analysis

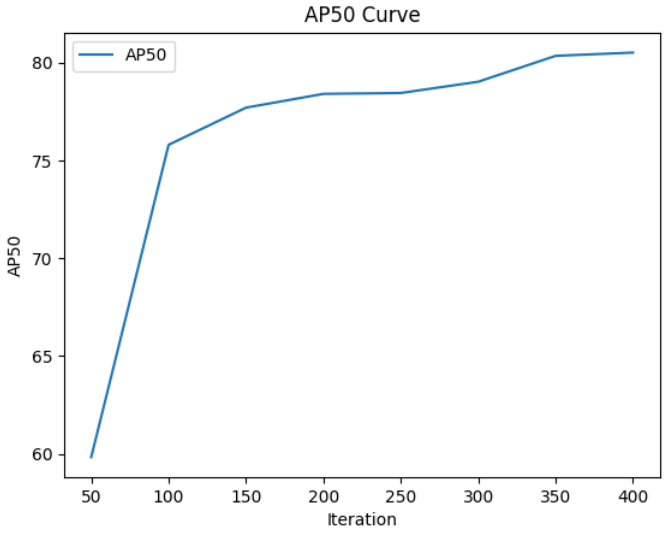


Figure 4.11: The AP50 metric for the ResNet-50 model on the local dataset.

Key observations:

- Starting Performance: The AP50 begins at a lower value of approximately 59.83 at iteration 50.
- Significant Improvement: There's substantially increased in AP50 throughout the training process.
- Final Performance: The AP50 reaches around 80.52 by iteration 400.
- Continuous Growth: Unlike the Sydney dataset, the AP50 curve shows no apparent plateau.

Analysis:

The lower initial AP50 value on the local dataset indicates that this dataset presented a more significant challenge to the model. However, the substantial improvement from

59.83 to 80.52 demonstrates the model's strong learning capability and adaptability. The continuous growth without an apparent plateau suggests that further training or data augmentation might yield even better results on this dataset.

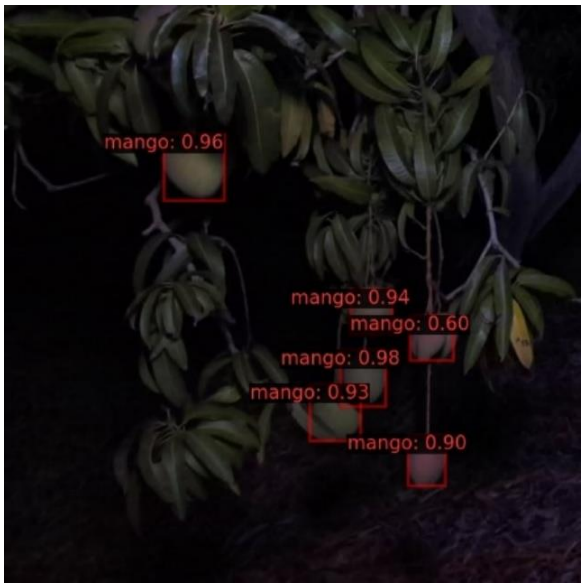


Figure 4.12: Local Dataset Model Example of Mango Detection, illustrating the model's detection capability.

Despite the significant improvement on the local dataset, there remains a considerable performance gap (93.03 vs 80.52) between the two datasets. This indicates that the local dataset may benefit from further data augmentation or a more extensive dataset because it is minimal in comparison. Also, despite their apparent differences, the model's ability to perform well on both datasets demonstrates its robustness and generalisation capability.

4.2.3 Comparison of Detectron2 Datasets Results

Table 4.4: Comparative Performance Metrics for Detectron2 on Sydney and Local Datasets.

Dataset	Final Total Loss	Final AP50
Sydney (AdamW)	0.483	93.02
Local (AdamW)	0.590	80.52

Analysis:

- The Sydney dataset results in a lower final total loss and higher AP50, indicating better overall model performance.
- The local dataset, while presenting more challenges, still achieves commendable results with significant improvements during training.

The Detectron2 implementation of Faster R-CNN with ResNet-50 backbone and AdamW optimizer has demonstrated strong performance on both the Sydney and local datasets. While the model achieved higher absolute performance on the Sydney dataset, it showed remarkable learning and adaptation on the more challenging local dataset.

4.3 Comparison of Faster R-CNN Results with and without Detectron2

Model/Optimizer	Dataset	Final AP50
VGG-16	Sydney	0.572
ResNet-50	Sydney	0.654
ResNet-50 with Detectron2	Sydney	93.02
ResNet-50 with Detectron2	Local	80.52

Detectron2 with AdamW optimizer shows superior performance compared to traditional Faster R-CNN implementations. ResNet-50 with AdamW in Detectron2 provides the best balance of low total loss and high AP50, demonstrating its robustness and efficiency.

4.4 YOLO Model Performance

This section presents the performance of YOLOv8 and YOLOv10 models across various datasets, detailing their training processes, evaluation metrics, and key findings.

4.4.1 YOLOv8 Implementation

The CQUniversity Dataset was used to evaluate the performance of YOLOv8 models under various configurations.

Table 4.5: YOLOv8 CQUniversity Dataset models metrics results.

Configuration	mAP@0.5	F1 Score (confidence)
Early Stopping Only	0.809	0.98 (0.570)
Early Stopping with Augmentation	0.987	0.98 (0.757)
Early Stopping with SGD	0.992	0.99 (0.479)
Early Stopping with SGD and Augmentation.	0.988	0.98 (0.610)

The analysis of the YOLOv8 models reveals that the best performance for the CQUniversity dataset was achieved using early stopping with SGD (mAP@0.5 of 0.992). This configuration balances high precision and recall with practical learning.

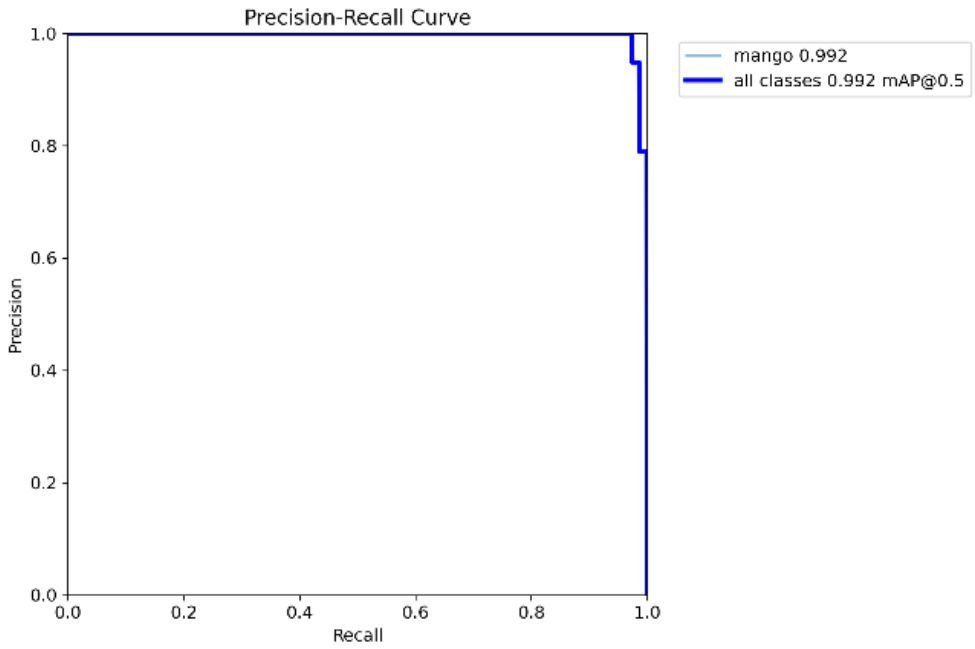


Figure 4.13: Precision-Recall Curve for Early Stopping with SGD on CQUniversity dataset.

This curve illustrates the trade-off between precision and recall for the YOLOv8 model using early stopping with SGD. A higher area under the curve indicates better performance.

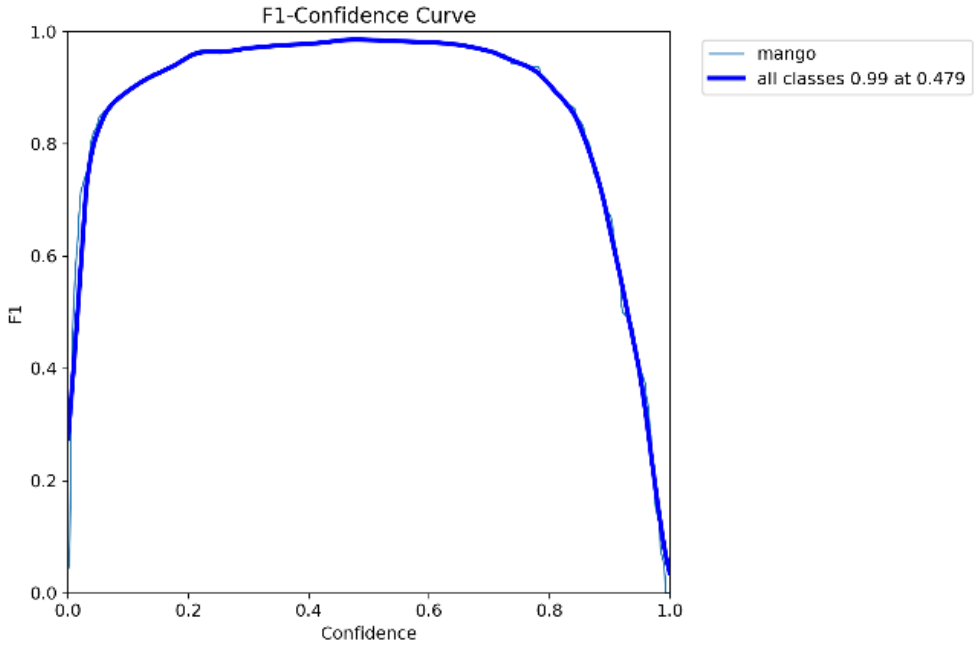


Figure 4.14: F1-Confidence Curve for Early Stopping with SGD on CQUniversity dataset.

This curve shows the F1 score at different confidence levels for the YOLOv8 model using early stopping with SGD. A higher peak indicates a better balance between precision and recall.

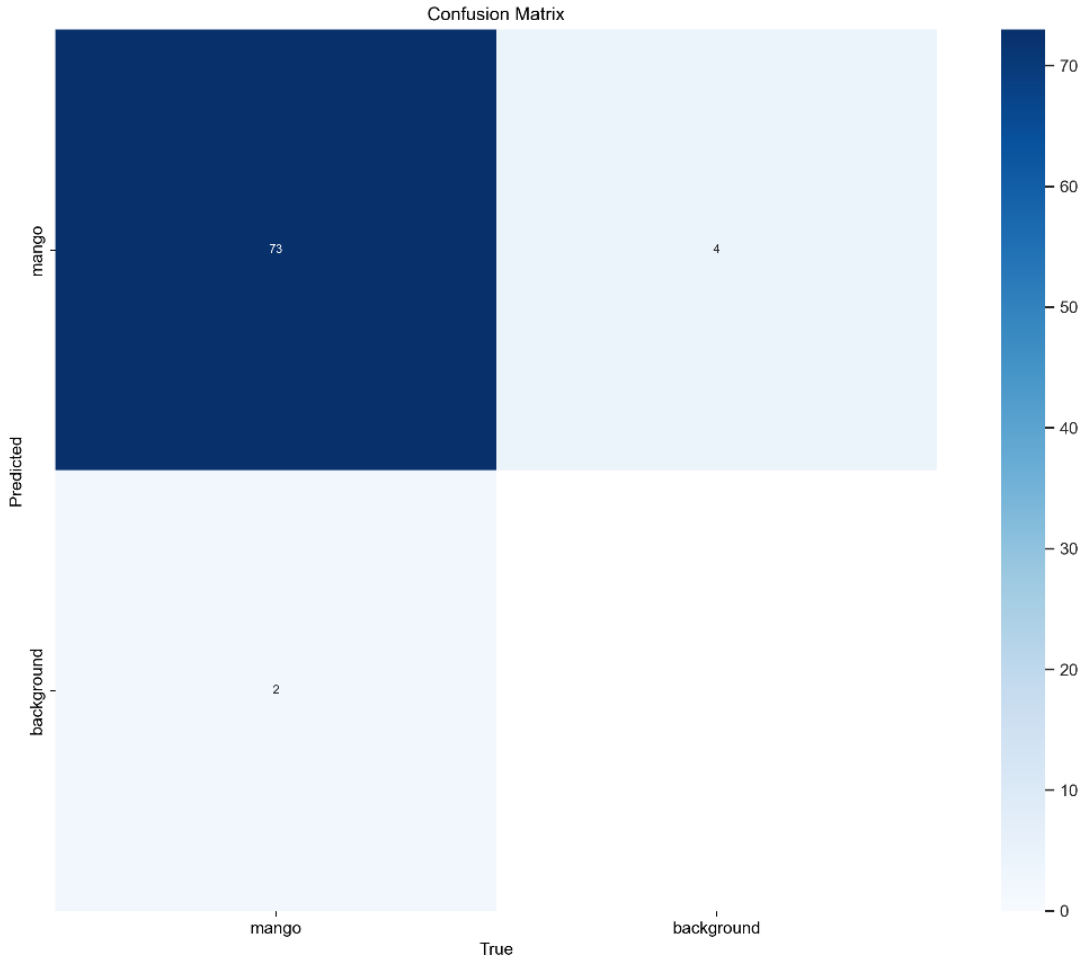


Figure 4.15: The confusion matrix visualises the performance of the YOLOv8 model in terms of true positives, false negatives, and false positives.

In this case, there are 73 true positives, two false negatives, and four false positives, indicating high accuracy.

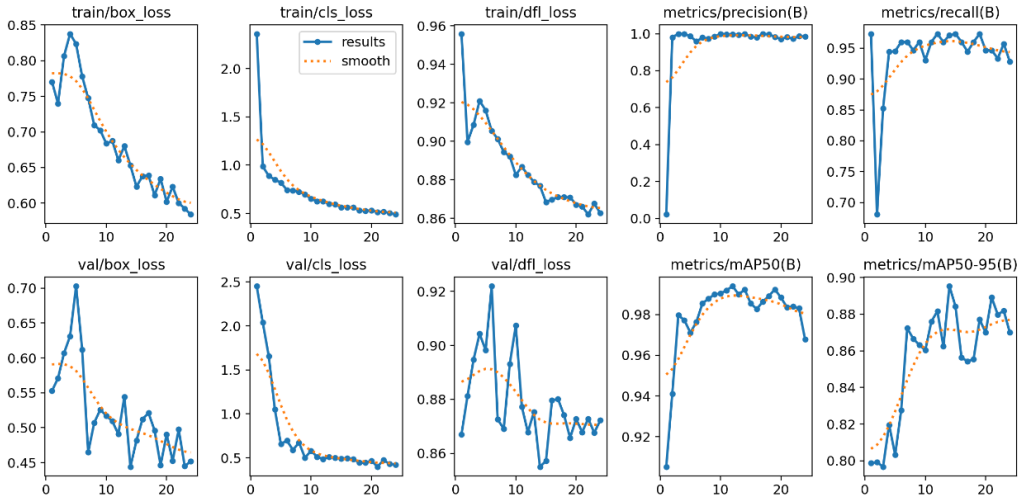


Figure 4.16: Training and Evaluation Metrics for Early Stopping with SGD on CQUniversity dataset.

This figure displays various training and evaluation metrics for the YOLOv8 model using early stopping with SGD, including loss curves and mAP metrics. The consistent decrease in loss and increase in mAP demonstrate effective learning and optimization.

Data augmentation improved the model's robustness and generalization ability, as evidenced by the higher mAP scores when applying augmentation techniques. However, the combination of early stopping with SGD without augmentation yielded the best results, suggesting that the optimizer played a more significant role in this case. Results indicate that the YOLOv8 models have strong generalisation capabilities, particularly with the SGD optimizer. The models performed well on the CQUniversity dataset, demonstrating their potential for real-world applications in mango detection.

Local Dataset:

Table 4.6: YOLOv8 Local Dataset models metrics results.

Configuration	mAP@0.5	F1 Score (confidence)
Early Stopping Only	0.792	0.80 (0.245)
Early Stopping with Augmentation	0.809	0.80 (0.320)
Early Stopping with SGD	0.784	0.78 (0. 279)
Early Stopping with SGD and Augmentation.	0.804	0.78 (0.380)

The best performance for the local dataset was obtained with early stopping and augmentation (mAP@0.5 of 0.809). This configuration shows effective learning and optimization for this specific dataset.

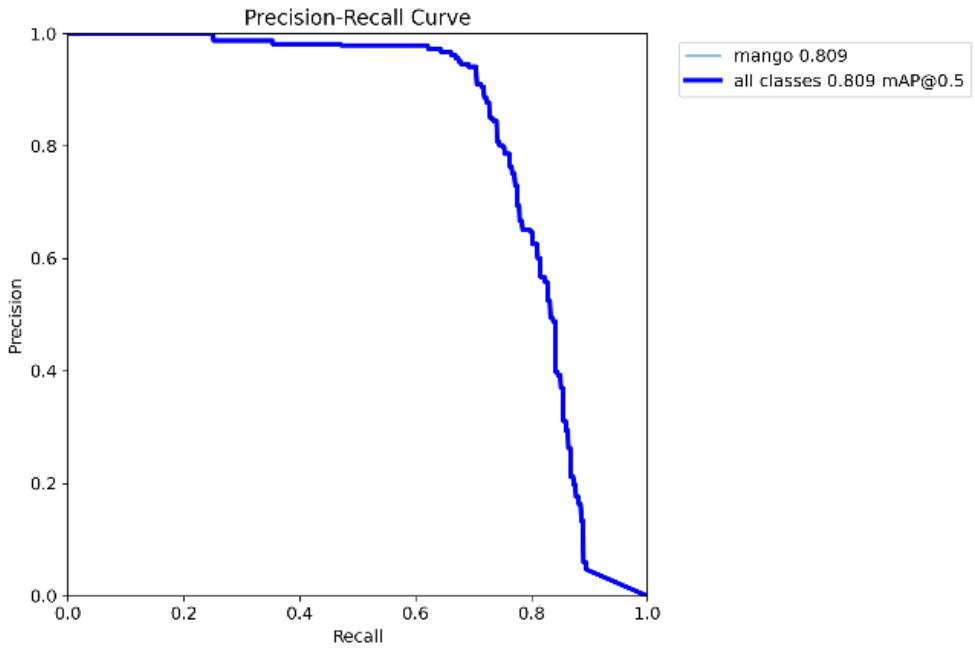


Figure 4.17: Precision-Recall Curve for Early Stopping with Augmentation on the local dataset.

The precision-recall curve indicates that the model maintains high precision across a range of recall values, suggesting that it can accurately identify mangoes with few false positives.

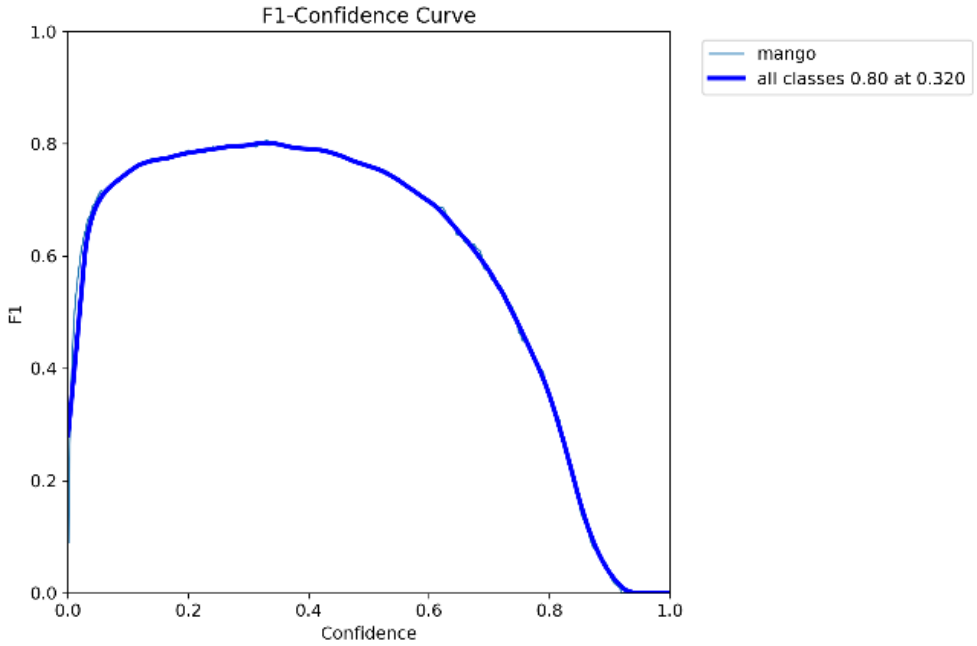


Figure 4.18: F1-Confidence Curve for Early Stopping with Augmentation on the local dataset.

The F1-confidence curve demonstrates that the optimal balance between precision and recall is achieved at a confidence level of around 0.320, where the F1 score is the highest.

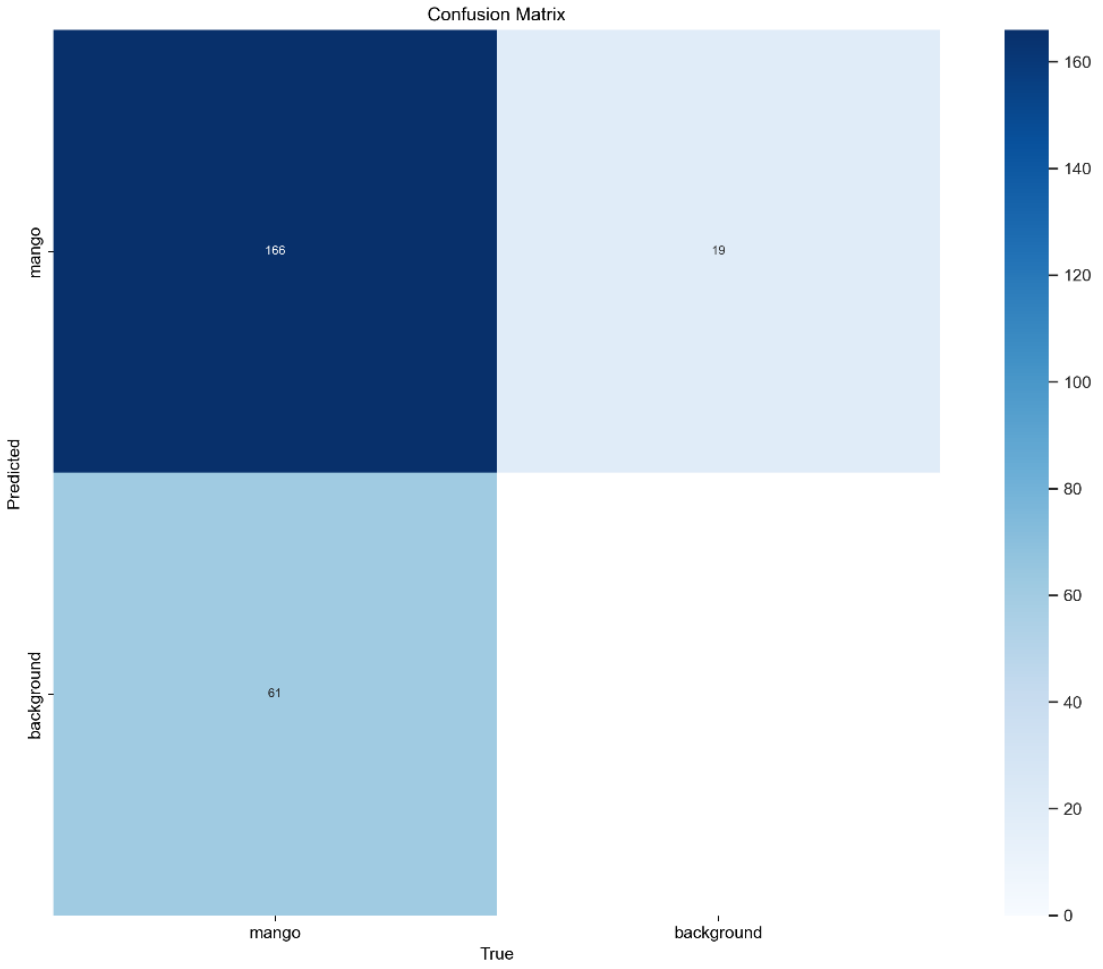


Figure 4.19: The confusion matrix provides a detailed view of the model's performance, showing the number of true positives, false positives, and false negatives.

The model successfully identified 166 mangoes but missed 61 and incorrectly identified 19 non-mango objects as mangoes.

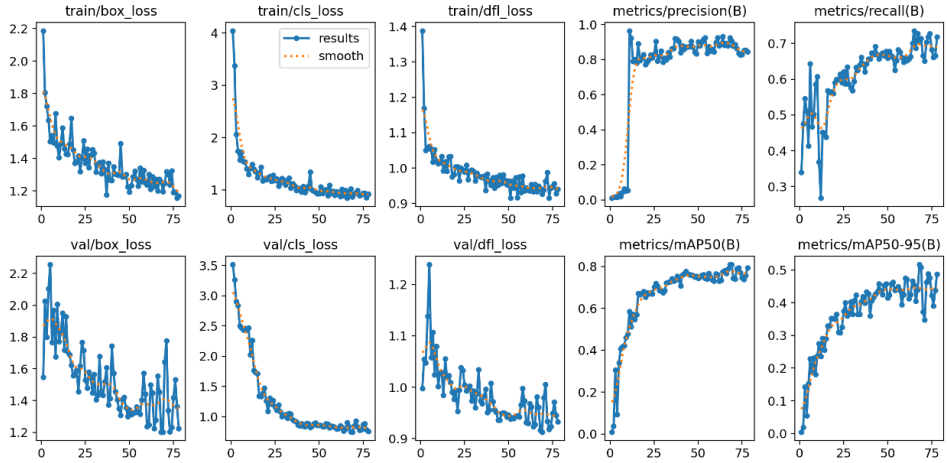


Figure 4.20: Training and Evaluation Metrics for Early Stopping with Augmentation on the local dataset.

These curves illustrate the training and validation loss and precision, recall, and mAP metrics over epochs. The model shows a steady decrease in loss and improvement in performance metrics, indicating practical training.

Figure 4.21: Saved detection from Gui using the local dataset best model. This image shows an example of the YOLOv8 model detecting mangoes in the field.

The bounding boxes highlight the detected mangoes with high confidence scores, demonstrating the model's practical application.

Sydney Dataset:

Table 4.7: YOLOv8 Sydney Dataset models metrics results.

Configuration	mAP@0.5	F1 Score (confidence)
Early Stopping Only	0.946	0.89 (0.430)
Early Stopping with Augmentation	0.959	0.91 (0.395)
Early Stopping with SGD	0.933	0.88 (0.146)
Early Stopping with SGD and Augmentation.	0.949	0.90 (0.412)

The best performance for the Sydney dataset was achieved using early stopping with augmentation (mAP@0.5 of 0.959). This configuration effectively balances precision and recall while demonstrating strong learning capabilities.

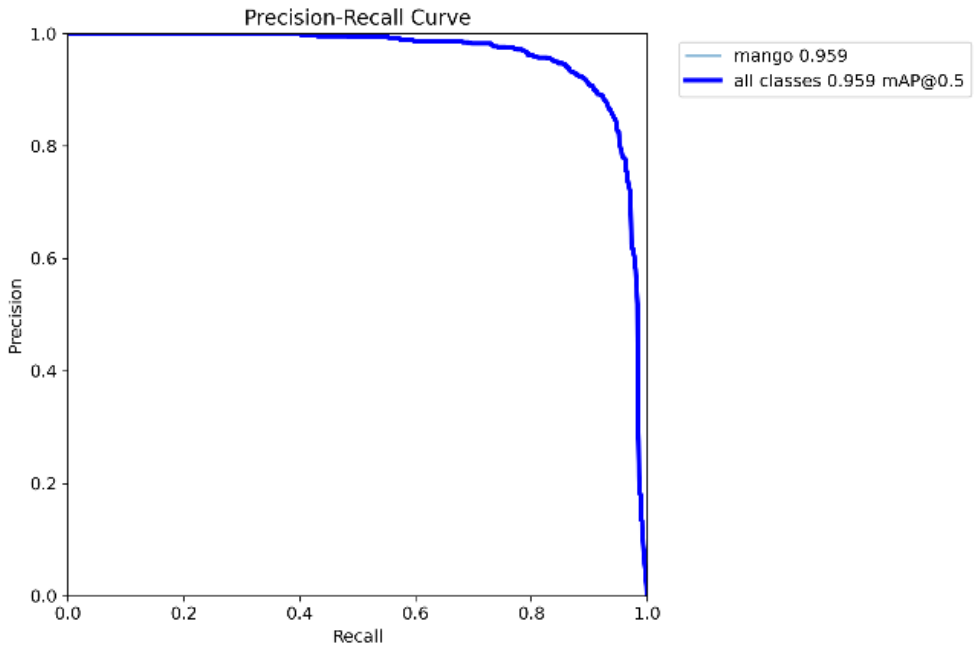


Figure 4.22: Precision-Recall Curve for Early Stopping with Augmentation on Sydney dataset.

The precision-recall curve indicates that the model maintains high precision across a range of recall values, suggesting that it can accurately identify mangoes with few false positives.

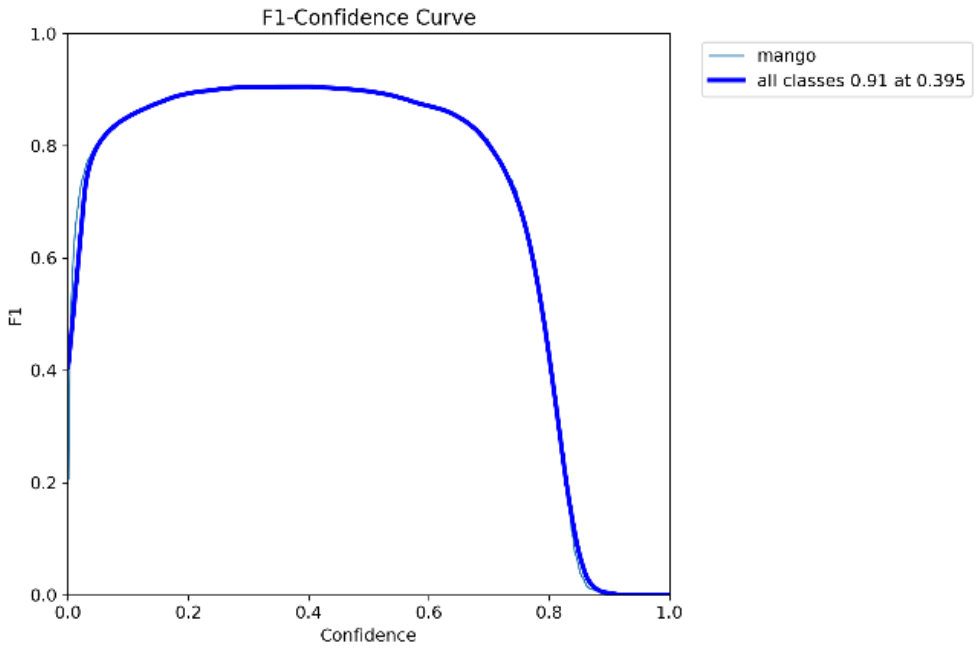


Figure 4.23: F1-Confidence Curve for Early Stopping with Augmentation on Sydney dataset.

The F1-confidence curve demonstrates that the optimal balance between precision and recall is achieved at a confidence level of around 0.395, where the F1 score is the highest.

The Confusion Matrix shows 863 true positives, 59 false negatives, and 113 false positives.

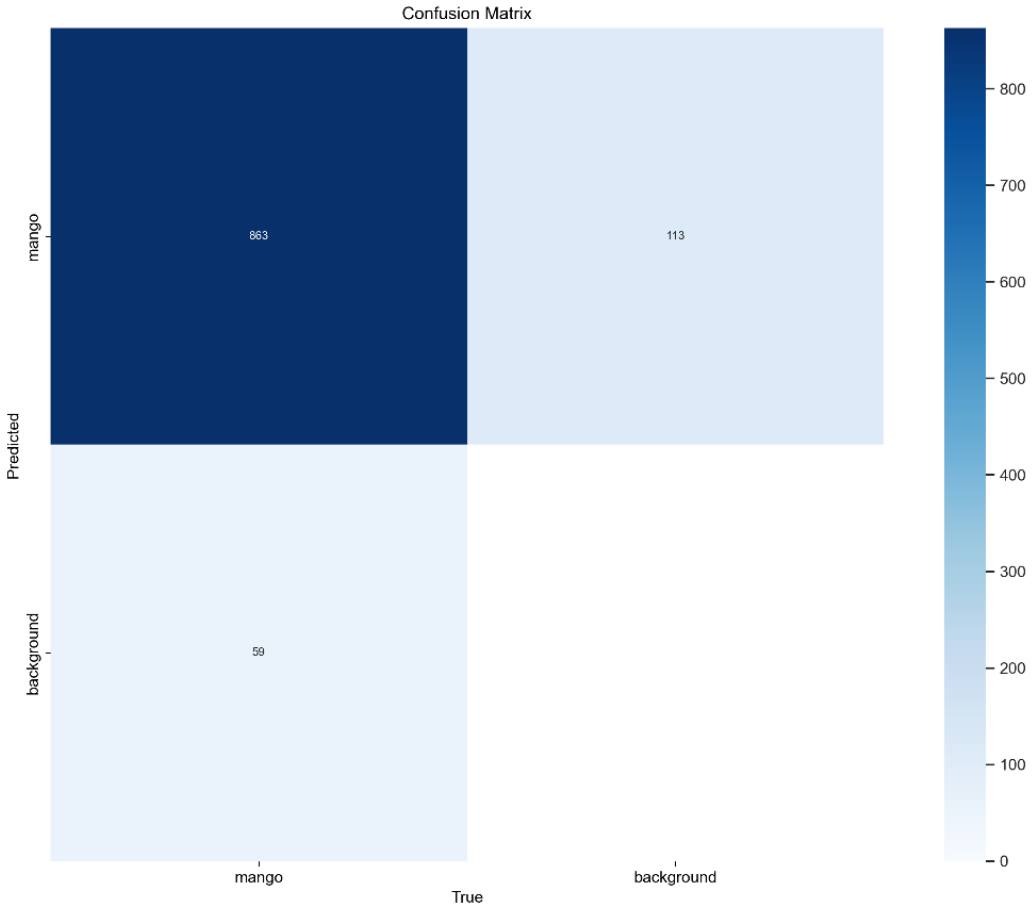


Figure 4.24: Confusion Matrix for Early Stopping with Augmentation on Sydney dataset.

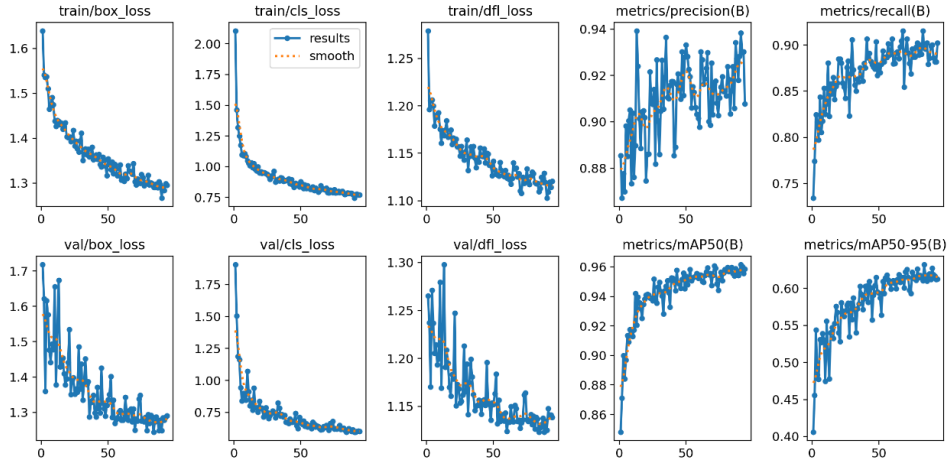


Figure 4.25: Training and Evaluation Metrics for Early Stopping with Augmentation on the Sydney dataset.

These curves illustrate the training and validation loss, precision, recall, and mAP metrics over epochs. The model shows a steady decrease in loss and improvement in performance metrics, indicating practical training.

Figure 4.26: Saved detection from GUI using the Sydney dataset best model.

This figure shows a saved detection from the GUI using the best-performing model trained on the Sydney dataset. It illustrates the model's practical application, visualising and saving detected mangoes within the user interface, and highlights the model's effectiveness and the GUI's added value.

4.4.2 YOLOv10 Implementation

CQUniversity Dataset:

Table 4.8: YOLOv10 CQUniversity Dataset models metrics results.

Configuration	mAP@0.5	F1 Score (confidence)
Early Stopping Only	0.953	0.92 (0.488)
Early Stopping with Augmentation	0.970	0.93 (0.340)
Early Stopping with SGD	0.947	0.91 (0.383)
Early Stopping with SGD and Augmentation.	0.960	0.91 (0.172)

Early stopping and augmentation achieved the best performance for the CQUniversity dataset (mAP@0.5 of 0.970). This setup balances high precision and recall with practical learning and optimization.

Including data augmentation and optimization techniques, they have significantly improved the YOLOv10 model's performance. Data augmentation introduced variability, enhancing the model's ability to generalise to different scenarios, while optimization methods like SGD and early stopping contributed to more effective learning and better convergence. These strategies led to higher accuracy, stability, and improved model performance.

The Figure 4.27 shows the trade-off between precision and recall for the YOLOv10 model trained with early stopping and augmentation.

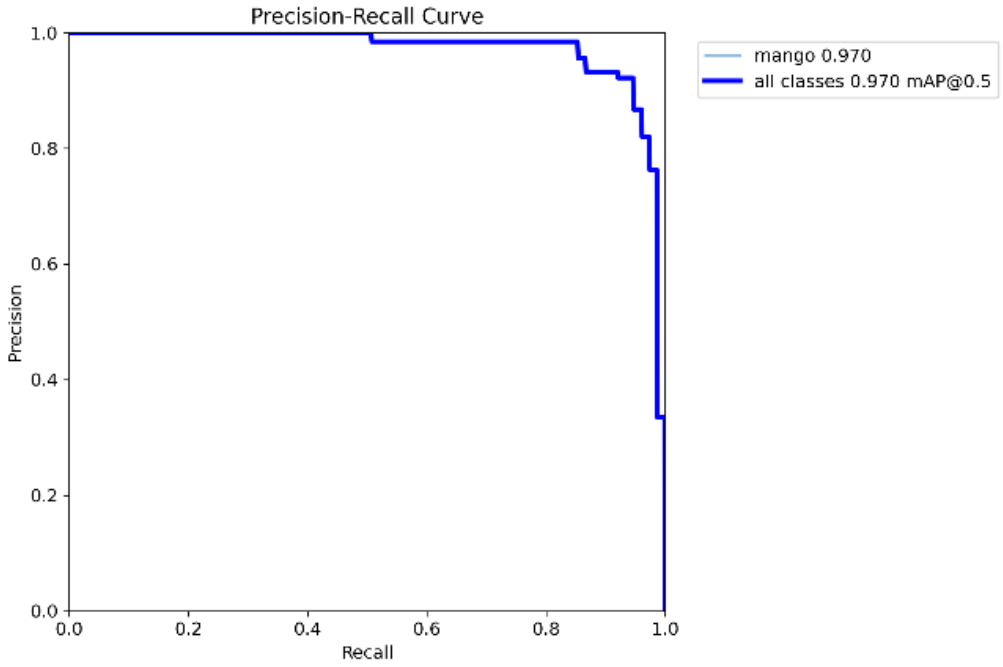


Figure 4.27: Precision-Recall Curve for Early Stopping with Augmentation on the CQUniversity dataset.

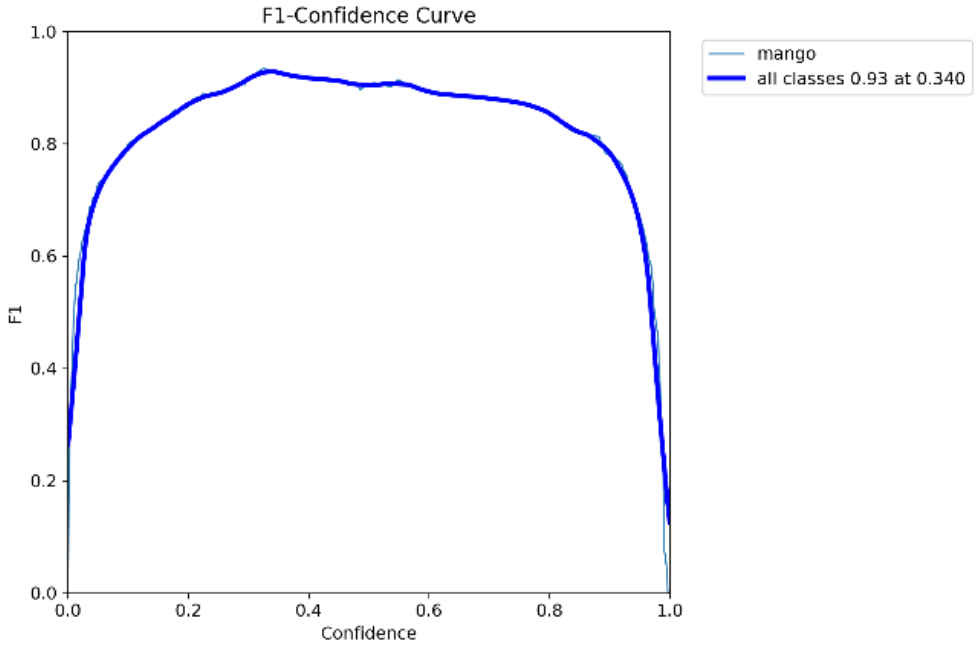


Figure 4.28: The curve illustrates the F1 score across different confidence thresholds.

The curve indicates optimal performance at a confidence of 0.340.

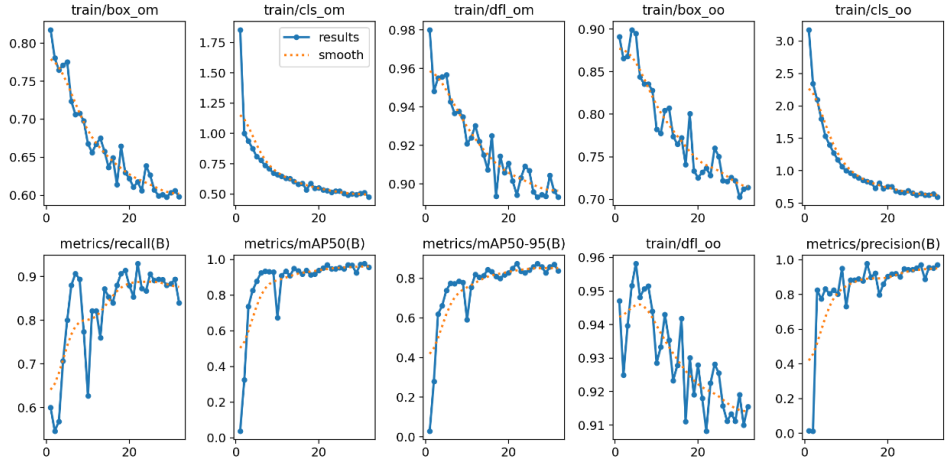


Figure 4.29: Training and Evaluation Metrics for Early Stopping with Augmentation on the CQUniversity dataset.

The plots provide insights into the training and validation losses and performance metrics over epochs, showing stable and practical learning.

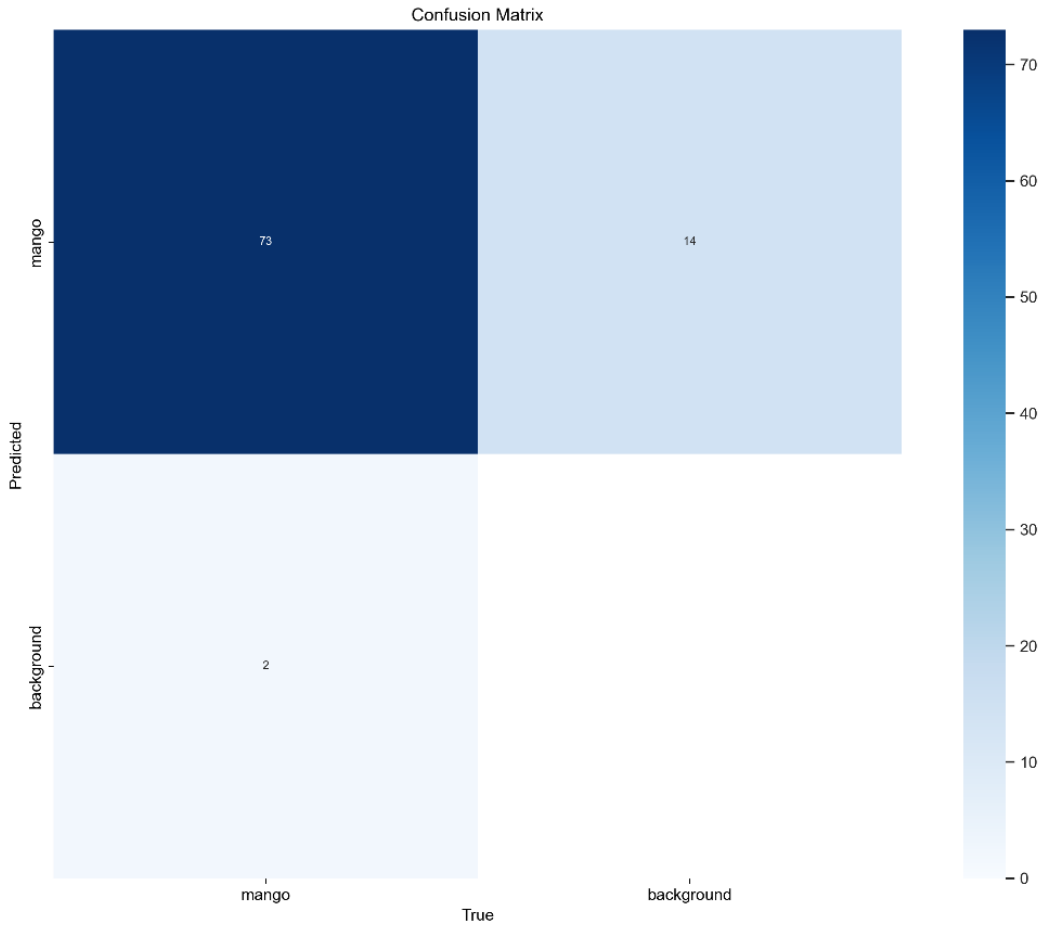


Figure 4.30: Confusion Matrix for Early Stopping with Augmentation on the CQUniversity dataset.

The confusion matrix shows 73 true positives, two false negatives, and 14 false positives.

Local Dataset:

Table 4.9: YOLOv10 local Dataset models metrics results.

Configuration	mAP@0.5	F1 Score (confidence)
Early Stopping Only	0.688	0.69 (0.256)
Early Stopping with Augmentation	0.692	0.69 (0.271)
Early Stopping with SGD	0.723	0.72 (0.168)
Early Stopping with SGD and Augmentation.	0.682	0.69 (0.404)

The analysis of the YOLOv10 models on the local dataset indicates that the best performance was achieved using early stopping with SGD, which effectively balanced

high precision and recall. While data augmentation contributed to improved generalisation, its combination with SGD did not surpass the performance of SGD alone. This suggests that while augmentation can enhance model robustness, the optimization technique is crucial in achieving superior performance. The precision-recall and F1-confidence curves and the confusion matrix provide insights into the model's strengths and areas for further improvement, particularly in reducing false negatives. Overall, the results highlight the importance of optimization strategies and data augmentation in training effective object detection models for agricultural applications.

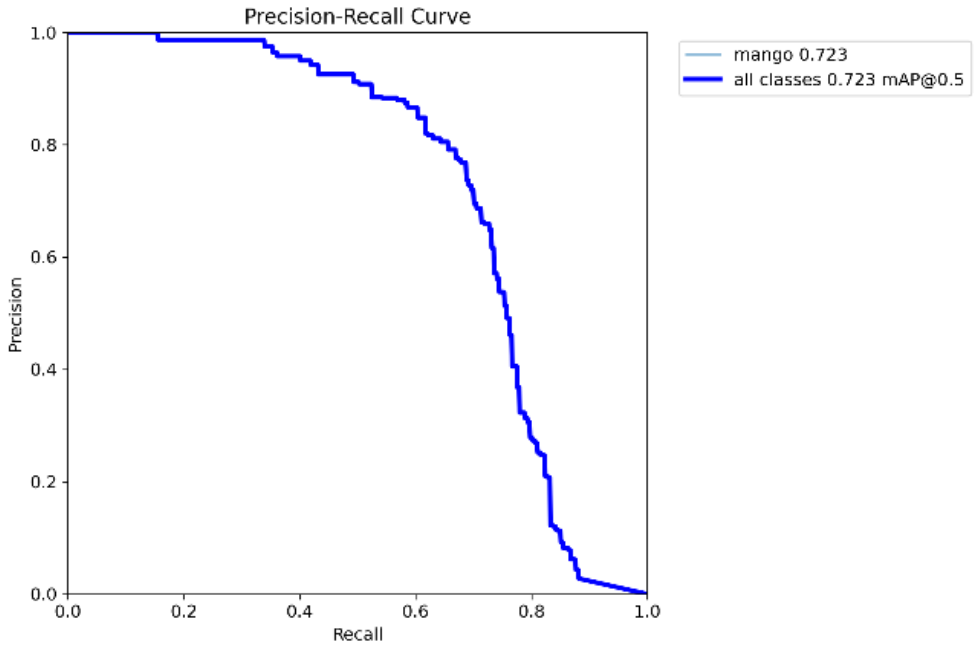


Figure 4.31: Precision-Recall Curve for Early Stopping with SGD on the local dataset.

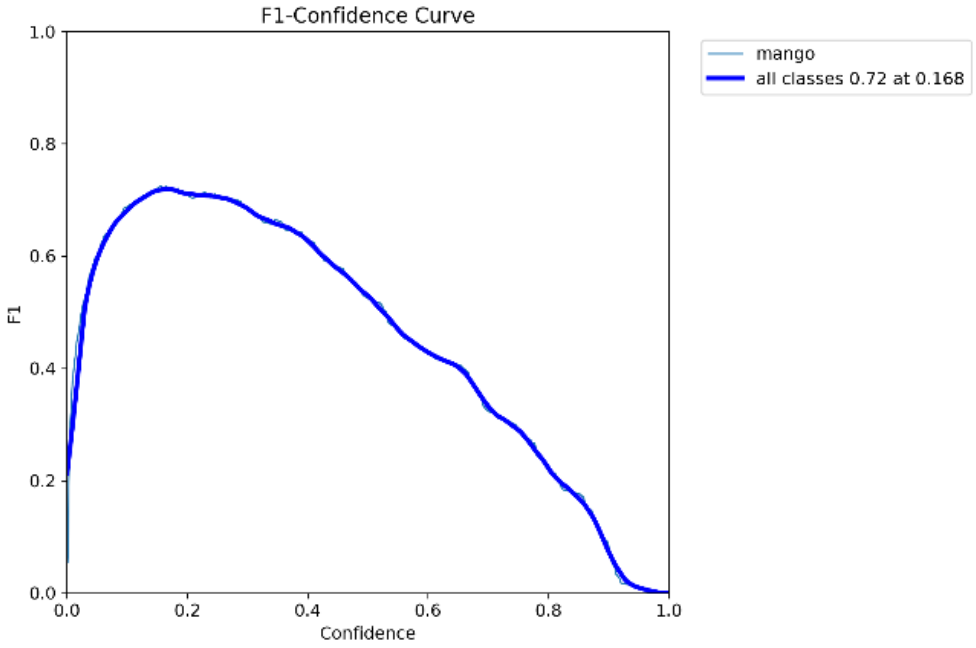


Figure 4.32: F1-Confidence Curve for Early Stopping with SGD on the local dataset.

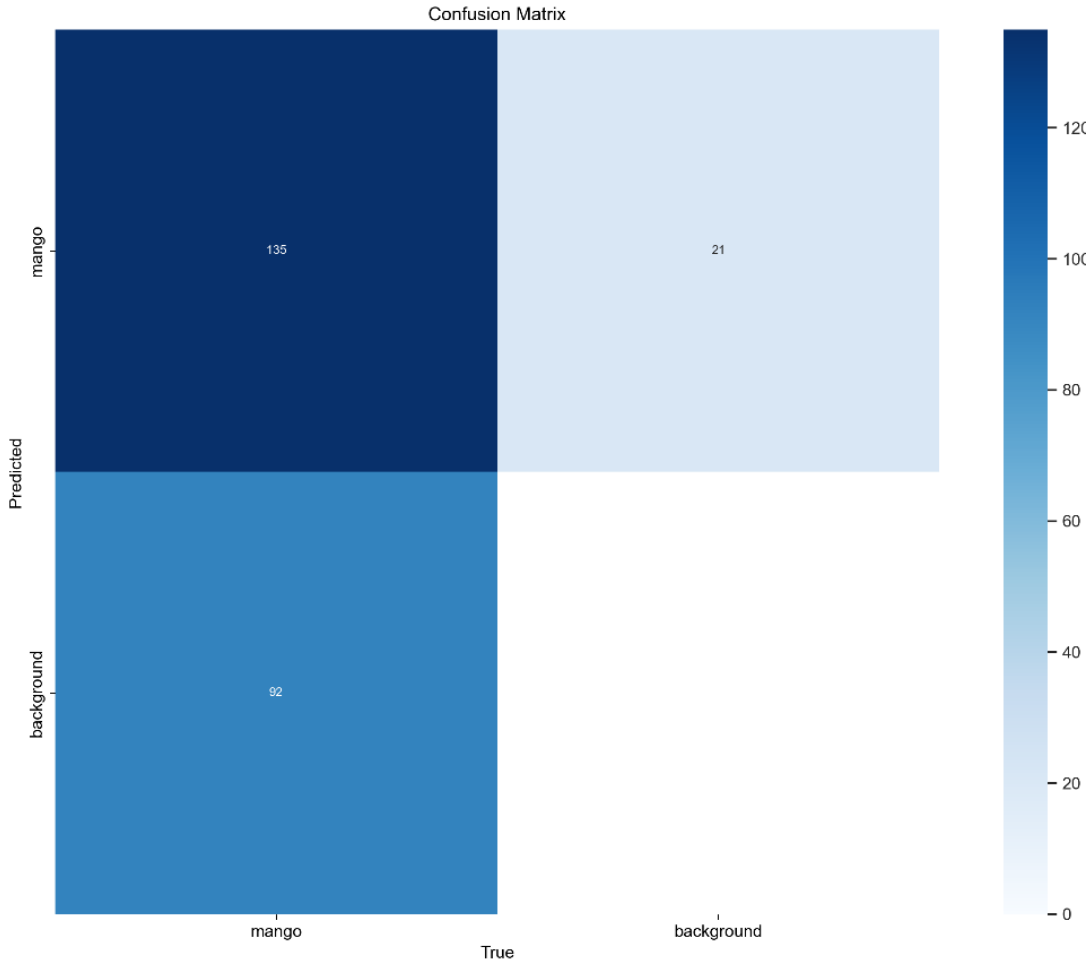


Figure 4.33: Confusion Matrix for Early Stopping with SGD on the local dataset. Confusion Matrix: 135 true positives, 92 false negatives, 21 false positives.

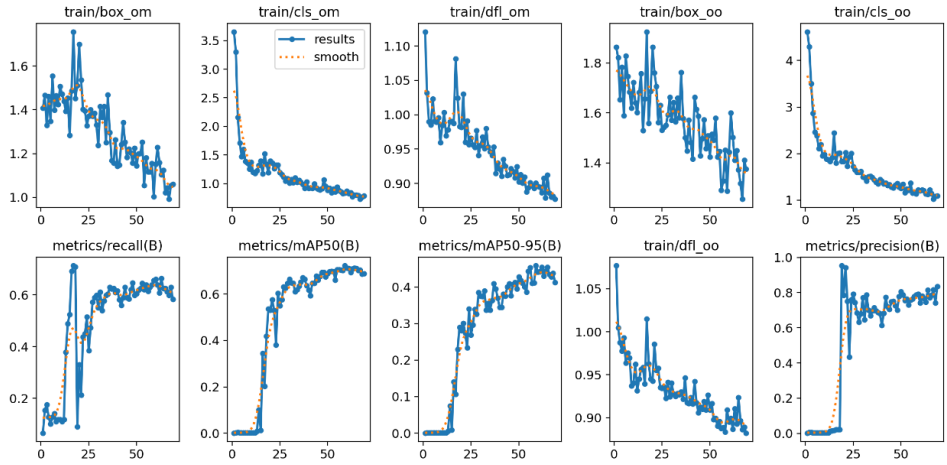


Figure 4.34: Training and Evaluation Metrics for Early Stopping with SGD on the local dataset.

Sydney Dataset:

Table 4.10: YOLOv10 Sydney Dataset models metrics results.

Configuration	mAP@0.5	F1 Score (confidence)
Early Stopping with Augmentation	0.926	0.87 (0.383)
Early Stopping with SGD	0.922	0.87 (0.386)
Early Stopping with SGD and Augmentation.	0.939	0.88 (0.435)

The best performance for the Sydney dataset was achieved using early stopping with SGD and augmentation (mAP@0.5 of 0.939). This configuration demonstrates strong learning capabilities and effective optimization, achieving a good balance between precision and recall. Data augmentation significantly improved the model's generalisation ability, while the SGD optimizer enhanced the learning process. The combined SGD and data augmentation approach resulted in the highest mAP@0.5 and F1 score, demonstrating strong learning capabilities and effective optimization.

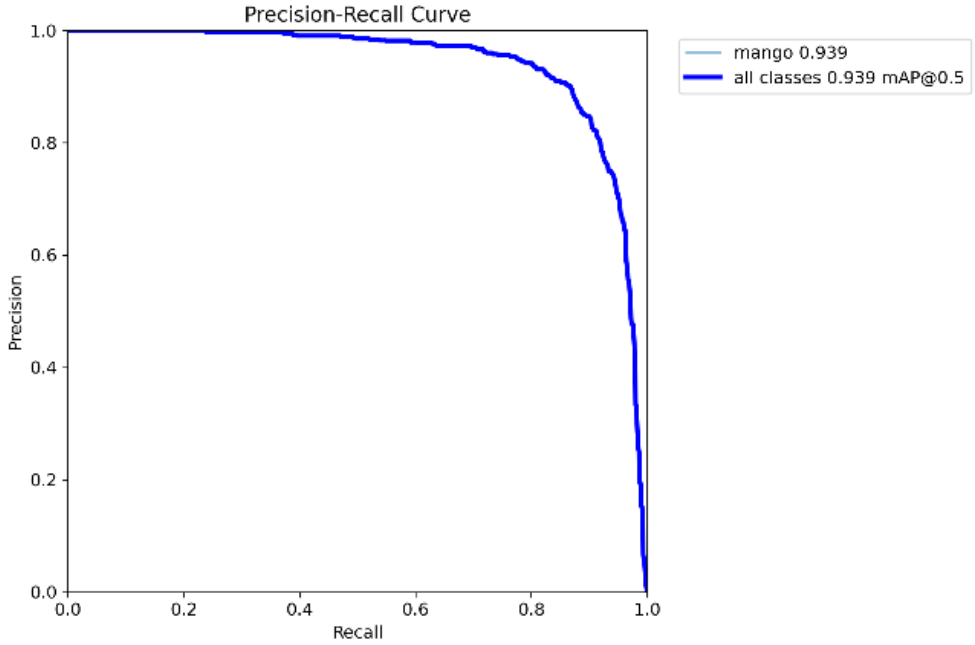


Figure 4.35: Precision-Recall Curve for Early Stopping with SGD and Augmentation on the Sydney dataset.

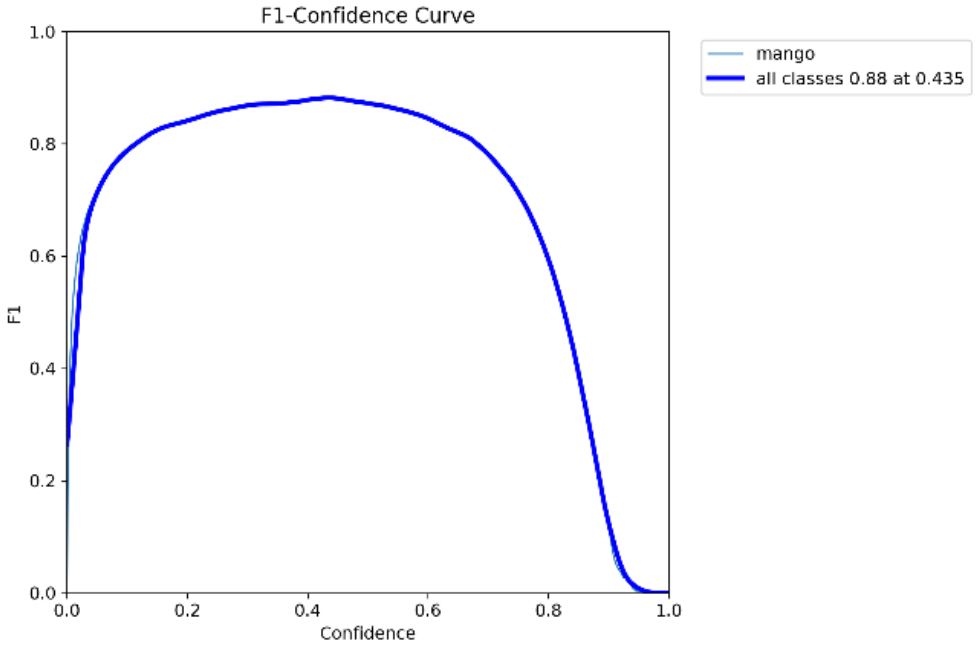


Figure 4.36: F1-Confidence Curve for Early Stopping with SGD and Augmentation on the Sydney dataset.

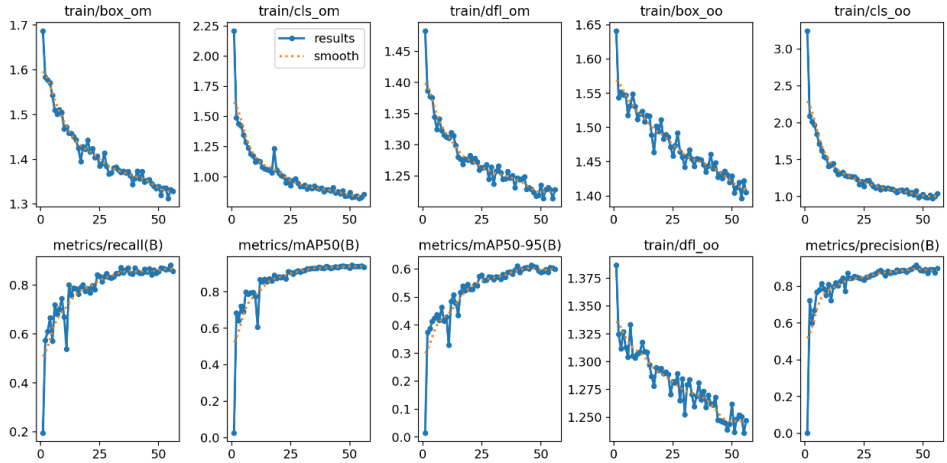


Figure 4.37: Training and Evaluation Metrics for Early Stopping with SGD and Augmentation on the Sydney dataset.

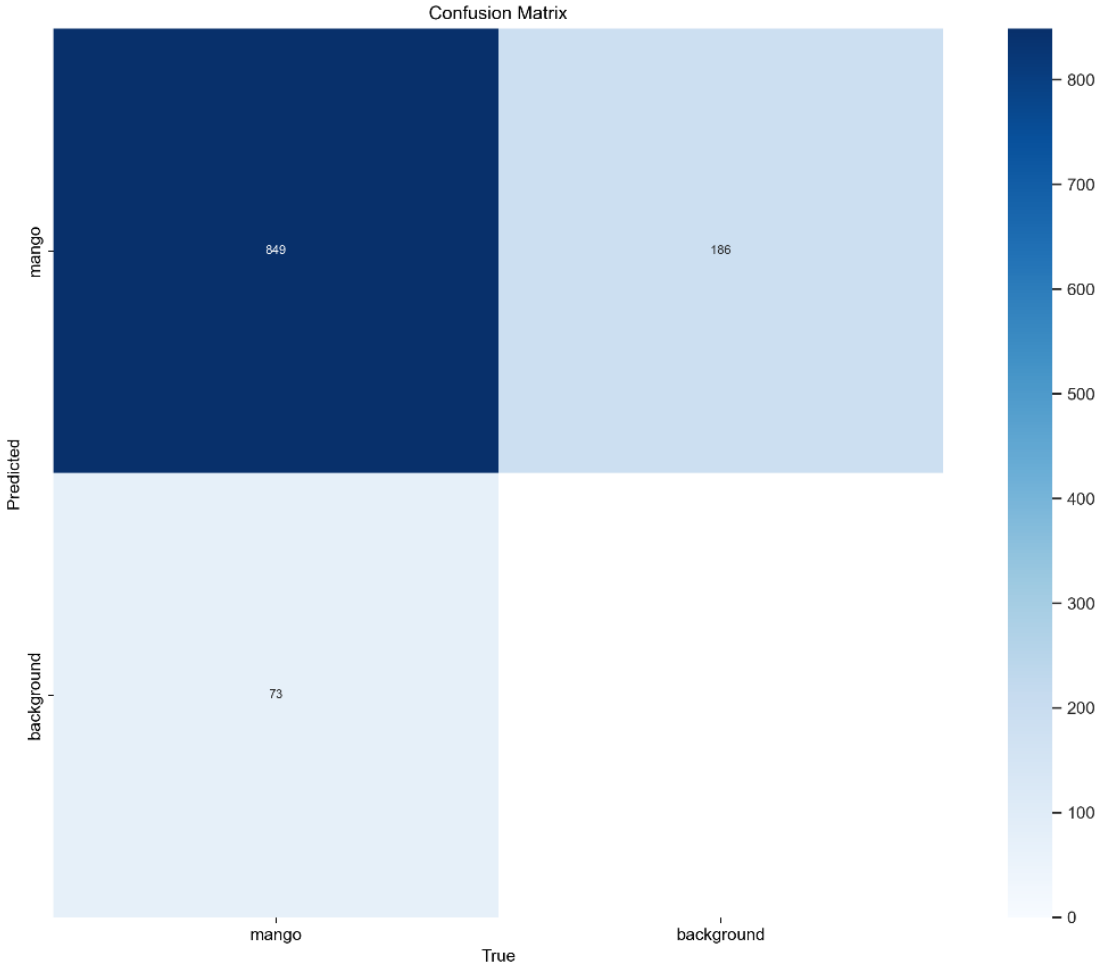


Figure 4.38: Confusion Matrix for Early Stopping with SGD and Augmentation on the Sydney dataset.

The matrix shows 849 true positives, 73 false negatives, and 186 false positives.

4.4.1 Comparison of Best YOLO Models on Each Dataset

Model	Dataset	Configuration	mAP@0.5	F1 Score (Confidence)
YOLOv8	CQUniversity	Early Stopping with SGD	0.992	0.99 (0.479)
YOLOv8	Local	Early Stopping with Augmentation	0.809	0.80 (0.320)
YOLOv8	Sydney	Early Stopping with SGD	0.985	0.99 (0.488)
YOLOv10	CQUniversity	Early Stopping with Augmentation	0.970	0.93 (0.340)
YOLOv10	Local	Early Stopping with SGD	0.723	0.72 (0.168)
YOLOv10	Sydney	Early Stopping with SGD and Augmentation	0.939	0.88 (0.435)

Best Dataset Model in YOLOv8:

CQUniversity dataset with YOLOv8 and early stopping with SGD emerged as the best configuration, indicating its superior capability in detecting mangoes with high precision and recall.

Best Dataset Model in YOLOv10:

For YOLOv10, the Sydney dataset with early stopping and SGD augmentation showed the highest performance, with a mAP@0.5 of 0.939 and an F1 score of 0.88 (Confidence: 0.435).

Figure 4.39: val_batch0_pred for Sydney dataset best model.

This figure showcases the val_batch0_pred results generated by the YOLO library for the Sydney dataset using the best-performing YOLOv10 model. It illustrates the model's predictions on a batch of validation images. Red bounding boxes indicate detected mangoes and their corresponding confidence scores, demonstrating the model's high accuracy and reliability in diverse conditions. The latest model.

Overall Best Model:

Across all datasets, YOLOv8 consistently shows better performance compared to YOLOv10**.** This includes higher mAP@0.5 and F1 scores, indicating superior

precision and recall. Among all models, the YOLOv8 model on the CQUniversity dataset with early stopping and SGD configuration is the best-performing model, demonstrating exceptional accuracy and robustness in mango detection tasks.

4.5 GUI Demonstration: MangoVision

MangoVision is a user-friendly graphical user interface (GUI) application that detects mango fruit in aerial images and videos. Built to focus on ease of use and efficiency, It integrates advanced object detection models with intuitive features to streamline the detection process.

Key Features of MangoVision:

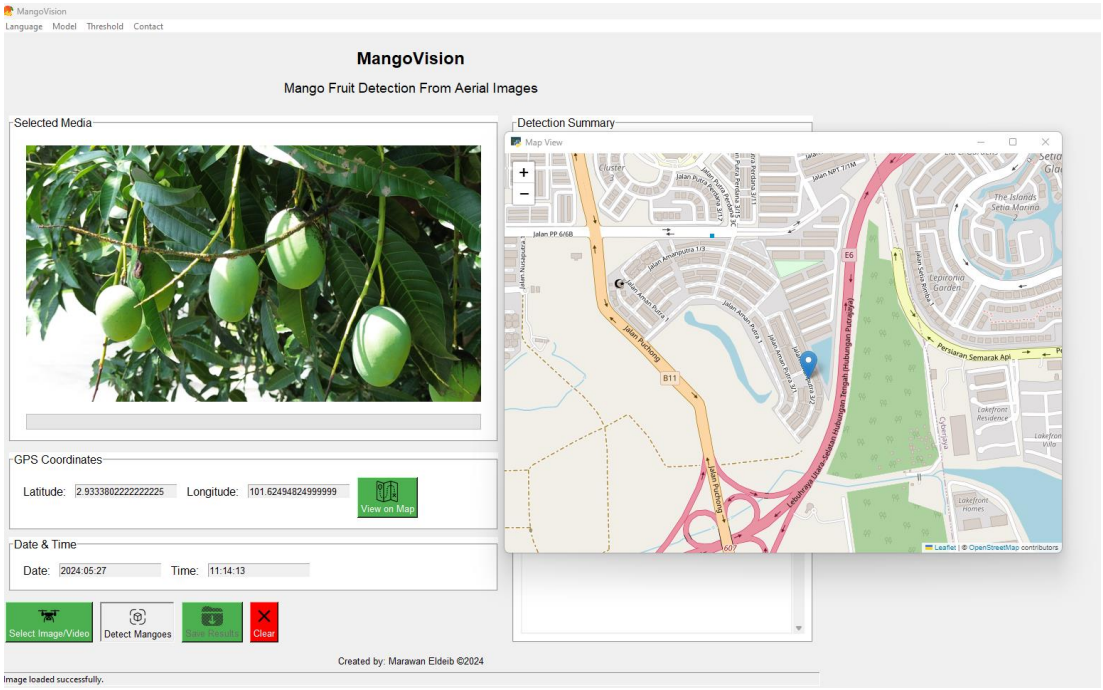


Figure 4.40: GPS Coordinates Extraction and Map View. Demonstrates the extraction of GPS coordinates and visualisation on a map.

- Image and Video Processing: Users can easily select image or video files for processing, initiate mango detection, and view real-time results.

- GPS Coordinates Extraction: The application supports extracting and displaying GPS coordinates from image metadata.
- Annotation and Saving: Detected mangoes can be annotated, and these annotated results can be saved for future reference.
- Multilingual Support: The application offers support for multiple languages, enhancing accessibility for users worldwide.
- Integrated Map View: Visualize GPS coordinates on an integrated map, providing a clear view of the detected mango locations.
- Video Playback Controls: Comprehensive controls for video playback allow users to navigate the footage efficiently.

These features make MangoVision a practical and accessible tool for agricultural applications, providing high accuracy and a straightforward interface.

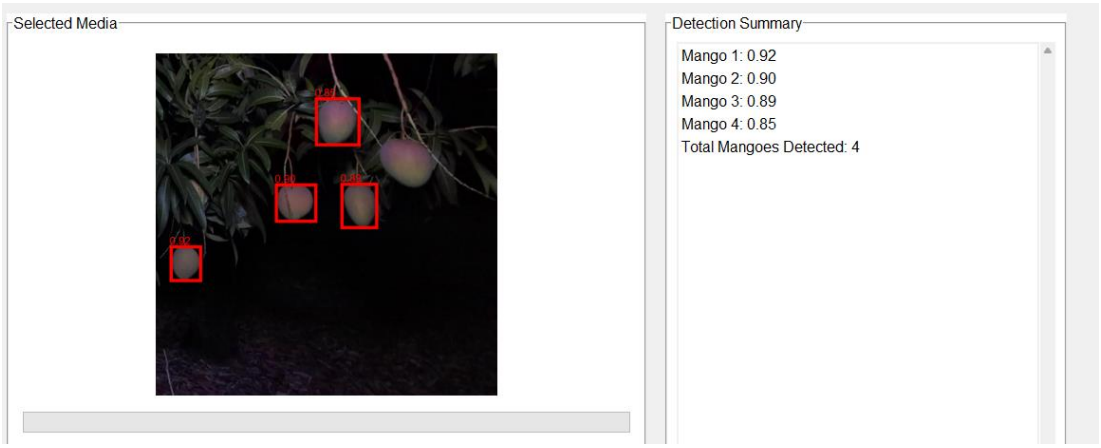


Figure 4.41: Mango Detection in Image. Shows mango detections annotated with bounding boxes in an image.

MangoVision demonstrates the integration of advanced deep learning models with practical, user-centric design, providing a powerful tool for mango fruit detection.

CHAPTER 5 CONCLUSIONS

5.1 Summary and Conclusions

This research has explored applying deep learning models, specifically Faster R-CNN and YOLO variants, to mango detection using aerial imagery. The primary objective was to identify the most effective model for accurate and efficient mango detection to aid agricultural monitoring.

The study involved extensive experiments and comparisons across two primary models: Faster R-CNN with VGG-16 and ResNet-50 backbones, YOLOv8, and YOLOv10, evaluated using datasets from the University of Sydney, CQUniversity, and a locally prepared dataset. The results demonstrated significant differences in performance between the models.

The Faster R-CNN models showed that ResNet-50 outperformed VGG-16, achieving a higher AP50 due to its more profound architecture and residual connections. Utilising the Detectron2 framework with the AdamW optimizer further enhanced the performance of Faster R-CNN models, with ResNet-50 achieving an AP50 of 93.02 on the Sydney dataset and 80.52 on the local dataset. This combination emerged as the best-performing configuration among the Faster R-CNN models.

YOLO models, particularly YOLOv8, exhibited strong performance across various configurations and datasets. The best configuration for YOLOv8 on the CQUniversity dataset was early stopping with SGD, achieving a mAP@0.5 of 0.992. On the Sydney dataset, early stopping with augmentation yielded a mAP@0.5 of 0.959. YOLOv10 also performed well, with the best results on the CQUniversity dataset achieved using early stopping with augmentation, reaching a mAP@0.5 of 0.970. Early stopping with SGD and augmentation on the Sydney dataset produced a mAP@0.5 of 0.939.

In a comparative analysis, YOLOv8 models consistently outperformed YOLOv10 models across all datasets. The CQUniversity dataset with YOLOv8 and early stopping.

with SGD emerged as the best overall configuration, indicating superior capability in detecting mangoes with high precision and recall. Therefore, the YOLOv8 model on the CQUniversity dataset with early stopping and SGD configuration is the best-performing model among all evaluated models, demonstrating exceptional accuracy and robustness in mango detection tasks.

Overall, the findings indicate that YOLO models outperform Detectron2, which outperforms traditional Faster R-CNN implementations. This performance hierarchy underscores the advancements in object detection techniques, with YOLO models leading in accuracy and efficiency.

The development of MangoVision, a user-friendly GUI application, integrated these advanced object detection models with practical features such as image and video processing, GPS coordinates extraction, annotation, and results saving. MangoVision provides a valuable solution for real-time mango detection in agricultural applications, enhancing accessibility and usability for end-users.

This research contributes to agricultural technology and paves the way for more efficient and accurate monitoring systems in precision agriculture.

5.2 Areas of Future Research

This research opens several avenues for future exploration:

- **Model Enhancements:** Future studies could incorporate more advanced architectures like EfficientDet to improve detection accuracy and efficiency.
- **Dataset Expansion:** Expanding the datasets to include more diverse and challenging conditions, such as varying weather, lighting, and occlusion scenarios, would enhance the models' generalizability. Creating and annotating a dataset for mango ripeness detection could provide valuable insights into fruit maturity and optimise harvesting decisions.

- **Field Deployment:** Deploying these models in real-world scenarios, such as integrating them with drones for automated mango detection and harvesting, would offer practical insights and applications, further bridging the gap between research and field implementation.

In conclusion, this thesis has demonstrated the potential of deep learning models for mango detection, providing a robust foundation for future advancements. The successful integration of these models into practical tools like MangoVision underscores the significant impact of artificial intelligence and machine learning in agriculture. By enhancing precision and efficiency in mango detection, this research contributes to more innovative and sustainable agricultural practices, paving the way for future technological developments.

REFERENCES

[1] I. Guiamba, “Nutritional Value and Quality of Processed Mango Fruits”.

[2] D. T. Zia, “Precision Farming: How AI and Drones Are Reshaping Agriculture,” Techopedia. Accessed: Jun. 20, 2024. [Online]. Available: https://www.techopedia.com/precision-farming-how-ai-and-drones-are-reshaping-agriculture

[3] A. Banafa, “AI and Drones 🤖,” OpenMind. Accessed: Jun. 20, 2024. [Online]. Available: https://www.bbvaopenmind.com/en/technology/artificial-intelligence/ai-and-drones/

[4] T. Bell, “Make way for robots in the sky: How drones are transforming farming in South Africa Make way for robots in the sky: How drones are transforming farming in South Africa,” Daily Maverick. Accessed: Jun. 20, 2024. [Online]. Available: https://www.dailymaverick.co.za/article/2021-11-09-make-way-for-robots-in-the-sky-how-drones-are-transforming-farming-in-south-africa/

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection.” arXiv, May 09, 2016. Accessed: Jul. 03, 2024. [Online]. Available: http://arxiv.org/abs/1506.02640

[6] J. Huang *et al.*, “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 3296–3297. doi: 10.1109/CVPR.2017.351.

[7] “Brief summary of YOLOv8 model structure · Issue #189 · ultralytics/ultralytics,” GitHub. Accessed: Jul. 07, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/189

[8] D. Reis, J. Kupec, J. Hong, and A. Daoudi, “Real-Time Flying Object Detection with YOLOv8.” arXiv, 2023. doi: 10.48550/ARXIV.2305.09972.

[9] A. Wang *et al.*, “YOLOv10: Real-Time End-to-End Object Detection.” arXiv, 2024. doi: 10.48550/ARXIV.2405.14458.

[10] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” arXiv, Jan. 06, 2016. doi: 10.48550/arXiv.1506.01497.

[11] S. C, K. JaganMohan, and M. Arulaalan, “Real Time Riped Fruit Detection using Faster R-CNN Deep Neural Network Models,” in *2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*, Villupuram, India: IEEE, Mar. 2022, pp. 1–4. doi: 10.1109/ICSTSN53084.2022.9761356.

[12] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 936–944. doi: 10.1109/CVPR.2017.106.

[13] S. Bargoti and J. Underwood, “Deep fruit detection in orchards,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, Singapore: IEEE, May 2017, pp. 3626–3633. doi: 10.1109/ICRA.2017.7989417.

[14] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks.” arXiv, Nov. 28, 2013. Accessed: Jul. 05, 2024. [Online]. Available: http://arxiv.org/abs/1311.2901

[15] “VGG-16 | CNN model,” GeeksforGeeks. Accessed: Jul. 05, 2024. [Online]. Available: https://www.geeksforgeeks.org/vgg-16-cnn-model/

[16] S. Mukherjee, “The Annotated ResNet-50,” Medium. Accessed: Jul. 05, 2024. [Online]. Available: https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[18] “Detectron2.” Accessed: Jul. 07, 2024. [Online]. Available: https://ai.meta.com/tools/detectron2

[19] V. V. Pham, *Hands-on computer vision with Detectron2: develop object detection and segmentation models with a code and visualization approach*, 1st edition. Birmingham, UK: Packt Publishing Ltd., 2023.

[20] “Interpreting Loss Curves | Machine Learning,” Google for Developers. Accessed: Jul. 07, 2024. [Online]. Available: https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic

[21] “What is A Confusion Matrix in Machine Learning? The Model Evaluation Tool Explained.” Accessed: Jul. 07, 2024. [Online]. Available: https://www.datacamp.com/tutorial/what-is-a-confusion-matrix-in-machine-learning

[22] “ONNX | Home.” Accessed: Jul. 05, 2024. [Online]. Available: https://onnx.ai/

[23] J. Xiong *et al.*, “Visual detection of green mangoes by an unmanned aerial vehicle in orchards based on a deep learning method,” *Biosystems Engineering*, vol. 194, pp. 261–272, Jun. 2020, doi: 10.1016/j.biosystemseng.2020.04.006.

[24] A. Ranjan and R. Machavaram, “Detection and Localisation of Farm Mangoes using YOLOv5 Deep Learning Technique,” in *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, Mumbai, India: IEEE, Apr. 2022, pp. 1–5. doi: 10.1109/I2CT54291.2022.9825078.

[25] R. Nithya, B. Santhi, R. Manikandan, M. Rahimi, and A. H. Gandomi, “Computer Vision System for Mango Fruit Defect Detection Using Deep

Convolutional Neural Network," *Foods*, vol. 11, no. 21, p. 3483, Nov. 2022, doi: 10.3390/foods11213483.

[26] I. H. Asif, "Complete Machine Learning Project Flowchart Explained!," Medium. Accessed: Jul. 07, 2024. [Online]. Available: https://ihsanulpro.medium.com/complete-machine-learning-project-flowchart-explained-0f55e52b9381

[27] "CUDA Toolkit 11.8 Downloads," NVIDIA Developer. Accessed: Jul. 07, 2024. [Online]. Available: https://developer.nvidia.com/cuda-11-8-0-download-archive

[28] "Python Release Python 3.10.9," Python.org. Accessed: Jul. 07, 2024. [Online]. Available: https://www.python.org/downloads/release/python-3109/

[29] "Start Locally," PyTorch. Accessed: Jul. 07, 2024. [Online]. Available: https://pytorch.org/get-started/locally/

[30] Ultralytics, "Home." Accessed: Jul. 07, 2024. [Online]. Available: https://docs.ultralytics.com/

[31] "Installation — detectron2 0.6 documentation." Accessed: Jul. 07, 2024. [Online]. Available: https://detectron2.readthedocs.io/en/latest/tutorials/install.html

[32] "Visual Studio Code June 2024." Accessed: Jul. 07, 2024. [Online]. Available: https://code.visualstudio.com/updates/v1_91

[33] "Mango Fruit Detection fyp - v4 2024-06-12 11:01pm," Roboflow. Accessed: Jul. 07, 2024. [Online]. Available: https://universe.roboflow.com/mango-fruit-detection/mango-fruit-detection-fyp

[34] "On-tree mango instance segmentation dataset." CQUniversity, Dec. 02, 2022. doi: 10.25946/21655628.v1.

APPENDIX A

Everything is available here at OneDrive.

https://mmuedumy-my.sharepoint.com/:f:/g/personal/1181102334_student_mmu_edu_my/EmdOMF8KSAFJvBCALKgcujMBfsWbWlUmXwoUbQ8sWFaiPw?e=Q3GJyq

Data_prep.ipynb for preparing the dataset for faster rcnn:

```
from google.colab import drive
drive.mount('/content/drive')
import shutil
import os
import pandas as pd
import glob
def update_annotation_files(train_annotations_dir):
    csv_list = os.listdir(train_annotations_dir)
    for j in csv_list:
        # Read the CSV file
        df = pd.read_csv(os.path.join(train_annotations_dir, j))

        # Update DataFrame with new columns
        df['filename'] = j.split('.csv')[0]
        df['dx_new'] = df['x'] + df['dx']
        df['dy_new'] = df['y'] + df['dy']

        # Save the updated DataFrame back to the CSV
        df.to_csv(os.path.join(train_annotations_dir, j), index=False)
training_list_dir = r'/content/drive/MyDrive/mangoes/sets/train.txt'
f = open(training_list_dir,mode='r')
images_dir = r'/content/drive/MyDrive/mangoes/images'
annotations_dir = r'/content/drive/MyDrive/mangoes/annotations'
train_images_dir = r'/content/drive/MyDrive/mangoes/train_images'
train_annotations_dir = r'/content/drive/MyDrive/mangoes/train_annotations'
test_images_dir = r'/content/drive/MyDrive/mangoes/test_images'
test_annotations_dir = r'/content/drive/MyDrive/mangoes/test_annotations'

# Create directories if they don't exist
os.makedirs(train_images_dir, exist_ok=True)
os.makedirs(train_annotations_dir, exist_ok=True)
os.makedirs(test_images_dir, exist_ok=True)
os.makedirs(test_annotations_dir, exist_ok=True)
with open(training_list_dir) as f:
    files_list = f.read().splitlines()
for i in files_list:
    shutil.copy(os.path.join(images_dir,(i+'.png')),train_images_dir)
for i in files_list:
    shutil.copy(os.path.join(annotations_dir,(i+'.csv')),train_annotations_dir)

# Update the training annotations CSV files with the new fields
update_annotation_files(train_annotations_dir)
#Combine individual csv filesused for training
```

```
os.chdir(r'/content/drive/MyDrive/mangoes/train_annotations')
extension = 'csv'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]
combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames ])
combined_csv.to_csv( "/content/drive/MyDrive/mangoes/combined_csv.csv", index=False,
encoding='utf-8-sig')
test_list_dir = r'/content/drive/MyDrive/mangoes/sets/test.txt'
f = open(test_list_dir,mode='r')
with open(test_list_dir) as f:
    files_list = f.read().splitlines()
for i in files_list:
    shutil.copy(os.path.join(images_dir,(i+'.png')),test_images_dir)
for i in files_list:
    shutil.copy(os.path.join(annotations_dir,(i+'.csv')),test_annotations_dir)
csv_list = os.listdir(test_annotations_dir)
for j in csv_list:
    df=pd.read_csv(os.path.join(test_annotations_dir,j))
    df['filename'] = j.split('.csv')[0]
    df['dx_new']= df['x']+df['dx']
    df['dy_new']= df['y']+df['dy']
    df.to_csv(os.path.join(test_annotations_dir,j))
#Combine individual csv filesused for testing
os.chdir(r'/content/drive/MyDrive/mangoes/test_annotations')
extension = 'csv'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]
combined_csv_test = pd.concat([pd.read_csv(f) for f in all_filenames ])
!mkdir "/content/drive/MyDrive/mangoes/combined_test"
combined_csv_test.to_csv('/content/drive/MyDrive/mangoes/combined_test/combined_test_csv.csv')
```

Fyp_part_1_mango_detection_faster_rcnn.ipynb:

```
from google.colab import drive
drive.mount('/content/drive')
# In cocoeval.py to get the thresholds of IoU @0.2 according to the paper(Deep Fruit Detection in
Orchards)
#Line 461
#stats[1] = _summarize(1, iouThr=.2, maxDets=self.params.maxDets[2])
#Line 466
#stats[6] = _summarize(0, iouThr=.2, maxDets=self.params.maxDets[2])
#Line 506
#self.iouThrs = np.linspace(.2, 1, int(np.round((1 - .2) / .05)) + 1, endpoint=True)
import torch
print(torch.__version__)
!pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
!git clone https://github.com/pytorch/vision.git
%cd vision
!cp references/detection/utils.py ../
!cp references/detection/transforms.py ../
!cp references/detection/coco_eval.py ../
!cp references/detection/engine.py ../
!cp references/detection/coco_utils.py ../
!pip install pycocotools
from pycocotools.cocoeval import COCOeval
%cd ..
# /usr/local/lib/python3.10/dist-packages/pycocotools/cocoeval.py
import pycocotools
```

```
import os
new_dir = '/content/drive/MyDrive/Fruit Detection DL/pytorch object detection/vision/references/detection'
os.makedirs(new_dir, exist_ok=True)
dir = r'/content/drive/MyDrive/Fruit Detection DL/pytorch object detection/vision/references/detection'
os.chdir(dir)
import numpy as np
import torch
import torch.utils.data
import torch.nn as nn
import PIL
from PIL import Image, ImageDraw
import pandas as pd
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator
from engine import train_one_epoch, evaluate
import utils
import os,sys
from torchvision import transforms as T
#import transforms as T
#import original_transforms as T2
import custom_transforms as T2
def parse_one_annot(path_to_data_file, filename):
    data = pd.read_csv(path_to_data_file)
    boxes_array = data[data["filename"] == filename][["x", "y", "dx_new", "dy_new"]].values
    return boxes_array
# Define the datset class. This loads the path to the images and path to the csv file with dimensions of bounding boxes
class FruitDataset(torch.utils.data.Dataset):
    def __init__(self, root, data_file, transforms=None):

        self.root = root
        self.transforms = transforms
        self.imgs = sorted(os.listdir(self.root))
        self.path_to_data_file = data_file


    def __getitem__(self, idx):

        # load images and bounding boxes

        img_path = os.path.join(self.root, self.imgs[idx])
        img = Image.open(img_path).convert("RGB")
        draw = ImageDraw.Draw(img)
        box_list = parse_one_annot(self.path_to_data_file,self.imgs[idx][:-4])
        boxes = torch.as_tensor(box_list, dtype=torch.float32)
        num_objs = len(box_list)
        labels = torch.ones((num_objs,), dtype=torch.int64)
        image_id = torch.tensor([idx])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:,0])
        # suppose all instances are not crowd
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)
        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["image_id"] = image_id
```

```
        target["area"] = area
        target["iscrowd"] = iscrowd
        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target

    def __len__(self):
        return len(self.imgs)
dataset = FruitDataset(root= r"/content/drive/MyDrive/mangoes/train_images",
data_file= r"/content/drive/MyDrive/mangoes/combined_csv.csv")
dataset.__getitem__(1)
from torchvision.models.detection.backbone_utils import resnet_fpn_backbone
def get_model(num_classes):
    # Load a pre-trained ResNet-50 backbone with FPN
    backbone = resnet_fpn_backbone('resnet50', pretrained=True)
    backbone.out_channels = 256

    # Anchor sizes and aspect ratios per feature map level
    anchor_sizes = ((32,), (64,), (128,), (256,), (512,))
    aspect_ratios = ((0.5, 1.0, 2.0),) * len(anchor_sizes)

    anchor_generator = AnchorGenerator(sizes=anchor_sizes, aspect_ratios=aspect_ratios)

    roi_pooler = torchvision.ops.MultiScaleRoIAlign(featmap_names=['0', '1', '2', '3'], output_size=7,
sampling_ratio=2)

    model = FasterRCNN(backbone, num_classes=num_classes,
rpn_anchor_generator=anchor_generator, box_roi_pool=roi_pooler)

    return model

#def get_model(num_classes):

    #Selecting the pretrained VGG16 faster-RCNN as the backbone model
    #from torchvision.models import vgg16, VGG16_Weights

    #backbone = torchvision.models.vgg16(pretrained=True).features
    #backbone.out_channels = 512

    #anchor_generator = AnchorGenerator(sizes=(( 64, 128,256),),aspect_ratios=((0.5, 1.0, 2.0),))

    #roi_pooler =
torchvision.ops.MultiScaleRoIAlign(featmap_names=["0"],output_size=7,sampling_ratio=2)

    #model =
FasterRCNN(backbone,num_classes=2,rpn_anchor_generator=anchor_generator,box_roi_pool=roi_p
ooler)

    #return model
class CustomCompose(object):
    def __init__(self, transforms):
        self.transforms = transforms

    def __call__(self, img, target=None):
        for t in self.transforms:
            if isinstance(t, (T2.RandomHorizontalFlip, T2.Resize)):  # Add T2.RandomGrayscale if using
it
```

```
            img, target = t(img, target)
        else:
            img = t(img)
    return img, target

def get_transform(train):

    transform_list = []
    # converts the image, a PIL image, into a PyTorch Tensor
    transform_list.append(T.ToTensor())
    if train:
      #Flip transform- images are flipped with 0.5 probability
      transform_list.append(T2.RandomHorizontalFlip(0.5))
      #Scale transform to 300x300
      transform_list.append(T2.Resize(size = 300))

      #Convert images to grayscale with a probability of 0.2
      #transform_list.append(T.RandomGrayscale(0.2))

    return CustomCompose(transform_list)
import multiprocessing

cores = multiprocessing.cpu_count() # Count the number of cores in a computer
cores
dataset = FruitDataset(root= r"/content/drive/MyDrive/mangoes/train_images",
data_file= r"/content/drive/MyDrive/mangoes/combined_csv.csv",transforms =
get_transform(train=True))

dataset_test = FruitDataset(root= r"/content/drive/MyDrive/mangoes/test_images",
        data_file= r"/content/drive/MyDrive/mangoes/combined_test/combined_test_csv.csv",
            transforms = get_transform(train=False))
#Random seed value
torch.manual_seed(1)
indices = torch.randperm(len(dataset)).tolist()

#Choose a subset from the training dataset
dataset_subset = torch.utils.data.Subset(dataset,indices[:500])

data_loader = torch.utils.data.DataLoader(dataset_subset, batch_size=1, shuffle=True,
num_workers=2,collate_fn=utils.collate_fn)

data_loader_test = torch.utils.data.DataLoader(dataset_test, batch_size=1, shuffle=False,
num_workers=1,collate_fn=utils.collate_fn)

print("Train dataset: ", len(data_loader), " \nTest dataset length",len(data_loader_test))
dataset.__getitem__(1)
#Check if GPU is available
torch.cuda.is_available()
device = torch.device('cuda:0')
# Our dataset has two classes only - mango and not mango
num_classes = 2
# Get the model using our helper function
model = get_model(num_classes)
# Move model to the right device
model.to(device)
# Construct an optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,momentum=0.9, weight_decay=0.0005)
```

```
# A learning rate scheduler which decreases the learning rate by # 10x every 5 epochs. Change this value based on the dataset size
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,step_size=5,gamma=0.1)
import os

# Number of epochs to train for
num_epochs = 20

for epoch in range(num_epochs):
    # Train for one epoch
    torch.cuda.empty_cache()
    train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=500)

    # Update the learning rate
    lr_scheduler.step()

    # Save the model every 2 epochs
    if (epoch % 2) == 0:
        save_path = r"/content/drive/MyDrive/Fruit Detection DL/pytorch object detection/fruit/Resnet50_size500_fifthtry_greyscale"
        # Create the directory if it does not exist
        os.makedirs(os.path.dirname(save_path), exist_ok=True)
        torch.save(model.state_dict(), save_path)

    # Evaluate the model on test set every 2 epochs
    if (epoch % 2) == 0:
        evaluate(model, data_loader_test, device=device)

# Save the model after training
os.makedirs(os.path.dirname(save_path), exist_ok=True)
torch.save(model.state_dict(), save_path)

# Evaluate on the test dataset after training
evaluate(model, data_loader_test, device=device)
#Load a saved model
loaded_model = get_model(num_classes = 2)
loaded_model.load_state_dict(torch.load(r"/content/drive/MyDrive/Fruit Detection DL/pytorch object detection/fruit/vgg16_flip_scale_size500"))
idx = 249
img, _ = dataset_test[idx]
label_boxes = np.array(dataset_test[idx][1]["boxes"])
#put the model in evaluation mode
loaded_model.eval()
with torch.no_grad():
  prediction = loaded_model([img])
image = Image.fromarray(img.mul(255).permute(1, 2,0).byte().numpy())
draw = ImageDraw.Draw(image)
# Draw groundtruth box in Green
for elem in range(len(label_boxes)):
  draw.rectangle([(label_boxes[elem][0], label_boxes[elem][1]),
  (label_boxes[elem][2], label_boxes[elem][3])],
  outline ="green", width =3)

#Draw prediction box in Red
for element in range(len(prediction[0]["boxes"])):
  boxes = prediction[0]["boxes"][element].cpu().numpy()
  score = np.round(prediction[0]["scores"][element].cpu().numpy(),
              decimals= 4)
```

```
  #Drawing prediction boxes above a certain confidence threshold
  if score > 0.8:
    draw.rectangle([(boxes[0], boxes[1]), (boxes[2], boxes[3])],
    outline ="red", width =3)
    draw.text((boxes[0], boxes[1]), text = str(score))
image
import matplotlib.pyplot as plt

# Data for Model vvg16
epochs = list(range(1, 21))
map_values = [ 0.055, 0.385, 0.325, 0.548, 0.545, 0.569, 0.571, 0.570, 0.571, 0.571, 0.572 ]
mar_values = [ 0.188, 0.549, 0.451, 0.635, 0.619, 0.644, 0.647, 0.645, 0.645, 0.645, 0.646 ]
training_loss = [ 0.3258, 0.1770, 0.1635, 0.1601, 0.1626, 0.1151, 0.1098, 0.1072, 0.1047, 0.1027,
            0.0989, 0.0979, 0.0985, 0.0986, 0.0985, 0.0982, 0.0987, 0.0977, 0.0970, 0.0972 ]

# Adjusting the length of map_values and mar_values to match the length of epochs
map_values += [map_values[-1]] * (len(epochs) - len(map_values))
mar_values += [mar_values[-1]] * (len(epochs) - len(mar_values))

# Plotting Mean Average Precision
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.plot(epochs, map_values, marker='o', color='b')
plt.title('vvg16 Mean Average Precision')
plt.xlabel('Epochs')
plt.ylabel('mAP')
plt.grid(True)

# Plotting Mean Average Recall
plt.subplot(1, 3, 2)
plt.plot(epochs, mar_values, marker='o', color='g')
plt.title('vvg16 Mean Average Recall')
plt.xlabel('Epochs')
plt.ylabel('mAR')
plt.grid(True)

# Plotting Training Loss
plt.subplot(1, 3, 3)
plt.plot(epochs, training_loss, marker='o', color='r')
plt.title('vvg16 Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)

plt.tight_layout()
plt.show()
import matplotlib.pyplot as plt

# Data for Resnet
epochs = list(range(1, 21))
map_values = [0.545, 0.595, 0.630, 0.657, 0.644, 0.658, 0.655, 0.653, 0.654, 0.654, 0.654]
mar_values = [0.633, 0.670, 0.697, 0.712, 0.698, 0.709, 0.705, 0.704, 0.706, 0.706, 0.705]
training_loss = [0.4967, 0.4657, 0.4229, 0.3818, 0.3375, 0.2817, 0.2747, 0.2690, 0.2666, 0.2616,
            0.2535, 0.2529, 0.2517, 0.2508, 0.2528, 0.2501, 0.2516, 0.2511, 0.2517, 0.2502]

# Extend MAP and MAR values to match the length of epochs
```

```
map_values.extend([map_values[-1]] * (len(epochs) - len(map_values)))
mar_values.extend([mar_values[-1]] * (len(epochs) - len(mar_values)))

# Plotting Mean Average Precision
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.plot(epochs, map_values, marker='o', color='b')
plt.title('Resnet Mean Average Precision')
plt.xlabel('Epochs')
plt.ylabel('MAP')
plt.grid(True)

# Plotting Mean Average Recall
plt.subplot(1, 3, 2)
plt.plot(epochs, mar_values, marker='o', color='g')
plt.title('Resnet Mean Average Recall')
plt.xlabel('Epochs')
plt.ylabel('MAR')
plt.grid(True)

# Plotting Training Loss
plt.subplot(1, 3, 3)
plt.plot(epochs, training_loss, marker='o', color='r')
plt.title('Resnet Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)

plt.tight_layout()
plt.show()
```

Detectron2, Sydney_train.py:

```
import detectron2
from detectron2.engine import DefaultTrainer, hooks
from detectron2.config import get_cfg
from detectron2.data.datasets import register_coco_instances
from detectron2.data import DatasetCatalog, MetadataCatalog
from detectron2 import model_zoo
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader, build_detection_train_loader, DatasetMapper
from detectron2.data import transforms as T
import os
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, f1_score
import numpy as np
import torch
import json
from detectron2.solver import build_optimizer, WarmupCosineLR, WarmupMultiStepLR

class TrainerWithEval(DefaultTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        return COCOEvaluator(dataset_name, cfg, True, output_folder)

    @classmethod
    def build_train_loader(cls, cfg):
        mapper = DatasetMapper(cfg, is_train=True, augmentations=[
            T.RandomFlip(horizontal=True, vertical=False),
            T.RandomRotation(angle=[-10, 10]),
            T.ResizeShortestEdge(short_edge_length=(800, 800), max_size=1333),
        ])
        return build_detection_train_loader(cfg, mapper=mapper)

    @classmethod
    def build_optimizer(cls, cfg, model):
        if hasattr(cfg.SOLVER, 'OPTIMIZER') and cfg.SOLVER.OPTIMIZER == "AdamW":
            return torch.optim.AdamW(model.parameters(), lr=cfg.SOLVER.BASE_LR,
weight_decay=cfg.SOLVER.WEIGHT_DECAY)
        else:
            return super().build_optimizer(cfg, model)

    @classmethod
    def build_lr_scheduler(cls, cfg, optimizer):
        if cfg.SOLVER.LR_SCHEDULER_NAME == "WarmupCosineLR":
            return WarmupCosineLR(
                optimizer,
                cfg.SOLVER.MAX_ITER,
                warmup_factor=cfg.SOLVER.WARMUP_FACTOR,
                warmup_iters=cfg.SOLVER.WARMUP_ITERS,
                warmup_method=cfg.SOLVER.WARMUP_METHOD,
            )
        elif cfg.SOLVER.LR_SCHEDULER_NAME == "WarmupMultiStepLR":
            return WarmupMultiStepLR(
                optimizer,
                milestones=cfg.SOLVER.STEPS,
                gamma=cfg.SOLVER.GAMMA,
                warmup_factor=cfg.SOLVER.WARMUP_FACTOR,
                warmup_iters=cfg.SOLVER.WARMUP_ITERS,
```

```
            warmup_method=cfg.SOLVER.WARMUP_METHOD,
        )
    return super().build_lr_scheduler(cfg, optimizer)

class CustomHookWithEarlyStopping(hooks.HookBase):
    def __init__(self, eval_dataset_name, patience=2, min_delta=0.001):
        self.eval_dataset_name = eval_dataset_name
        self.results = []
        self.patience = patience
        self.min_delta = min_delta
        self.best_metric = None
        self.counter = 0

    def after_step(self):
        if (self.trainer.iter + 1) % self.trainer.cfg.TEST.EVAL_PERIOD == 0:
            evaluator = COCOEvaluator(self.eval_dataset_name,
output_dir=self.trainer.cfg.OUTPUT_DIR)
            val_loader = build_detection_test_loader(self.trainer.cfg, self.eval_dataset_name)
            metrics = inference_on_dataset(self.trainer.model, val_loader, evaluator)
            print("Metrics:", metrics)
            result = {
                'iteration': self.trainer.iter + 1,
                'train/total_loss': self.trainer.storage.history("total_loss").values()[-1][0] if "total_loss" in
self.trainer.storage.histories() else None,
                'train/loss_cls': self.trainer.storage.history("loss_cls").values()[-1][0] if "loss_cls" in
self.trainer.storage.histories() else None,
                'train/loss_box_reg': self.trainer.storage.history("loss_box_reg").values()[-1][0] if
"loss_box_reg" in self.trainer.storage.histories() else None,
                'train/loss_rpn_cls': self.trainer.storage.history("loss_rpn_cls").values()[-1][0] if
"loss_rpn_cls" in self.trainer.storage.histories() else None,
                'train/loss_rpn_loc': self.trainer.storage.history("loss_rpn_loc").values()[-1][0] if
"loss_rpn_loc" in self.trainer.storage.histories() else None,
                'lr': self.trainer.optimizer.param_groups[0]['lr'],
                'AP': metrics['bbox']['AP'] if 'bbox' in metrics else None,
                'AP50': metrics['bbox']['AP50'] if 'bbox' in metrics else None,
                'AP75': metrics['bbox']['AP75'] if 'bbox' in metrics else None,
                'APs': metrics['bbox']['APs'] if 'bbox' in metrics else None,
                'APm': metrics['bbox']['APm'] if 'bbox' in metrics else None,
                'APl': metrics['bbox']['APl'] if 'bbox' in metrics else None,
            }
            print("Result:", result)
            self.results.append(result)

            current_metric = metrics['bbox']['AP'] if 'bbox' in metrics else None
            if self.best_metric is None:
                self.best_metric = current_metric
            elif current_metric < self.best_metric - self.min_delta:
                self.counter += 1
                if self.counter >= self.patience:
                    print("Early stopping triggered")
                    self.trainer.iter = self.trainer.max_iter
            else:
                self.best_metric = current_metric
                self.counter = 0

    def after_train(self):
        df = pd.DataFrame(self.results)
        df.to_excel(os.path.join(self.trainer.cfg.OUTPUT_DIR, "training_results.xlsx"), index=False)
```

```
        self.plot_curves(df)

    def plot_curves(self, df):
        metrics = ['train/total_loss', 'train/loss_cls', 'train/loss_box_reg', 'train/loss_rpn_cls',
'train/loss_rpn_loc', 'lr']
        for metric in metrics:
            if metric in df.columns:
                plt.figure()
                plt.plot(df['iteration'], df[metric], label=metric)
                plt.xlabel('Iteration')
                plt.ylabel(metric)
                plt.legend()
                plt.title(f'{metric} Curve')
                metric_dir = os.path.join(self.trainer.cfg.OUTPUT_DIR, os.path.dirname(metric))
                os.makedirs(metric_dir, exist_ok=True)
                plt.savefig(os.path.join(metric_dir, f"{os.path.basename(metric)}_curve.png"))
                plt.close()

        ap_metrics = ['AP', 'AP50', 'AP75', 'APs', 'APm', 'APl']
        for ap_metric in ap_metrics:
            if ap_metric in df.columns:
                ap_values = df[ap_metric]
                plt.figure()
                plt.plot(df['iteration'], ap_values, label=ap_metric)
                plt.xlabel('Iteration')
                plt.ylabel(ap_metric)
                plt.legend()
                plt.title(f'{ap_metric} Curve')
                plt.savefig(os.path.join(self.trainer.cfg.OUTPUT_DIR, f"{ap_metric}_curve.png"))
                plt.close()

        y_true = [1] * len(df)
        y_scores = df['AP50']
        precision, recall, _ = precision_recall_curve(y_true, y_scores)
        plt.figure()
        plt.plot(recall, precision, label='Precision-Recall Curve')
        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.legend()
        plt.title('Precision-Recall Curve')
        plt.savefig(os.path.join(self.trainer.cfg.OUTPUT_DIR, "precision_recall_curve.png"))
        plt.close()

        f1_scores = [f1_score([1], [1 if score >= 0.5 else 0], average='binary') for score in y_scores]
        plt.figure()
        plt.plot(df['iteration'], f1_scores, label='F1 Score')
        plt.xlabel('Iteration')
        plt.ylabel('F1 Score')
        plt.legend()
        plt.title('F1 Score Curve')
        plt.savefig(os.path.join(self.trainer.cfg.OUTPUT_DIR, "f1_score_curve.png"))
        plt.close()

def setup_config(model_name, output_subdir, optimizer="SGD", lr_scheduler=None):
    cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file(model_name))
    cfg.DATASETS.TRAIN = ("sydney_train",)
    cfg.DATASETS.TEST = ("sydney_val",)
```

```
    cfg.DATALOADER.NUM_WORKERS = 4
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(model_name)
    cfg.SOLVER.IMS_PER_BATCH = 4
    cfg.SOLVER.BASE_LR = 0.00025
    cfg.SOLVER.MAX_ITER = 3000
    cfg.SOLVER.STEPS = [15000, 18000]
    cfg.SOLVER.GAMMA = 0.1
    cfg.SOLVER.WARMUP_FACTOR = 1.0 / 1000
    cfg.SOLVER.WARMUP_ITERS = 1000
    cfg.SOLVER.WARMUP_METHOD = "linear"
    cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 256
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
    cfg.TEST.EVAL_PERIOD = 500
    cfg.OUTPUT_DIR = os.path.join("Sydney Dataset DETECTRON2 Results", output_subdir)
    if optimizer == "AdamW":
        cfg.SOLVER.OPTIMIZER = "AdamW"
        cfg.SOLVER.WEIGHT_DECAY = 0.01
    if lr_scheduler == "WarmupCosineLR":
        cfg.SOLVER.LR_SCHEDULER_NAME = "WarmupCosineLR"
    if lr_scheduler == "WarmupMultiStepLR":
        cfg.SOLVER.LR_SCHEDULER_NAME = "WarmupMultiStepLR"
    return cfg

def train_model(model_name, output_subdir, optimizer="SGD", lr_scheduler=None):
    if "sydney_train" not in DatasetCatalog.list():
        register_coco_instances("sydney_train", {}, "F:/Final_year_project/collected
dataset/Detectron2/SydneyUniversity_dataset/detectron2_split_dataset-20240706T042233Z-
001/detectron2_split_dataset/train/annotations_train.json", "F:/Final_year_project/collected
dataset/Detectron2/SydneyUniversity_dataset/detectron2_split_dataset-20240706T042233Z-
001/detectron2_split_dataset/train/images")
    if "sydney_val" not in DatasetCatalog.list():
        register_coco_instances("sydney_val", {}, "F:/Final_year_project/collected
dataset/Detectron2/SydneyUniversity_dataset/detectron2_split_dataset-20240706T042233Z-
001/detectron2_split_dataset/val/annotations_val.json", "F:/Final_year_project/collected
dataset/Detectron2/SydneyUniversity_dataset/detectron2_split_dataset-20240706T042233Z-
001/detectron2_split_dataset/val/images")
    if "sydney_test" not in DatasetCatalog.list():
        register_coco_instances("sydney_test", {}, "F:/Final_year_project/collected
dataset/Detectron2/SydneyUniversity_dataset/detectron2_split_dataset-20240706T042233Z-
001/detectron2_split_dataset/test/annotations_test.json", "F:/Final_year_project/collected
dataset/Detectron2/SydneyUniversity_dataset/detectron2_split_dataset/test/images")

    cfg = setup_config(model_name, output_subdir, optimizer, lr_scheduler)
    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

    trainer = TrainerWithEval(cfg)
    trainer.resume_or_load(resume=False)
    trainer.register_hooks([CustomHookWithEarlyStopping("sydney_val")])
    trainer.train()

    evaluator = COCOEvaluator("sydney_val", output_dir=cfg.OUTPUT_DIR)
    val_loader = build_detection_test_loader(cfg, "sydney_val")
    evaluation_results = inference_on_dataset(trainer.model, val_loader, evaluator)
    with open(os.path.join(cfg.OUTPUT_DIR, "evaluation_results.json"), "w") as f:
        json.dump(evaluation_results, f)

if __name__ == '__main__':
    import multiprocessing as mp
```

```
    mp.set_start_method('spawn', force=True)
    train_model("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml", "resnet50_sgd")
    train_model("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml", "resnet50_adamw",
optimizer="AdamW")
```

local_train.py:

```
import detectron2
from detectron2.engine import DefaultTrainer, hooks
from detectron2.config import get_cfg
from detectron2.data.datasets import register_coco_instances
from detectron2.data import DatasetCatalog, MetadataCatalog
from detectron2 import model_zoo
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader
import os
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, f1_score
import numpy as np
from detectron2.data import detection_utils as utils
from detectron2.data import transforms as T
from detectron2.data import build_detection_train_loader, DatasetMapper
import json
import torch
from detectron2.solver import build_optimizer, WarmupCosineLR, WarmupMultiStepLR

# Custom trainer with evaluation and augmented data loader
class TrainerWithEval(DefaultTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        return COCOEvaluator(dataset_name, cfg, True, output_folder)

    @classmethod
    def build_train_loader(cls, cfg):
        mapper = DatasetMapper(cfg, is_train=True, augmentations=[
            T.RandomFlip(horizontal=True, vertical=False),  # Only horizontal flipping
            T.RandomRotation(angle=[-10, 10]),  # Light rotation
            T.ResizeShortestEdge(short_edge_length=(800, 800), max_size=1333),  # Resize with fixed
shortest edge
        ])
        return build_detection_train_loader(cfg, mapper=mapper)

    @classmethod
    def build_optimizer(cls, cfg, model):
        if hasattr(cfg.SOLVER, 'OPTIMIZER') and cfg.SOLVER.OPTIMIZER == "AdamW":
            return torch.optim.AdamW(model.parameters(), lr=cfg.SOLVER.BASE_LR,
weight_decay=cfg.SOLVER.WEIGHT_DECAY)
        else:
            return super().build_optimizer(cfg, model)

    @classmethod
    def build_lr_scheduler(cls, cfg, optimizer):
        if cfg.SOLVER.LR_SCHEDULER_NAME == "WarmupCosineLR":
            return WarmupCosineLR(
                optimizer,
```

```
            cfg.SOLVER.MAX_ITER,
            warmup_factor=cfg.SOLVER.WARMUP_FACTOR,
            warmup_iters=cfg.SOLVER.WARMUP_ITERS,
            warmup_method=cfg.SOLVER.WARMUP_METHOD,
        )
    elif cfg.SOLVER.LR_SCHEDULER_NAME == "WarmupMultiStepLR":
        return WarmupMultiStepLR(
            optimizer,
            milestones=cfg.SOLVER.STEPS,
            gamma=cfg.SOLVER.GAMMA,
            warmup_factor=cfg.SOLVER.WARMUP_FACTOR,
            warmup_iters=cfg.SOLVER.WARMUP_ITERS,
            warmup_method=cfg.SOLVER.WARMUP_METHOD,
        )
    return super().build_lr_scheduler(cfg, optimizer)

# Custom hook for early stopping based on evaluation metric
class CustomHookWithEarlyStopping(hooks.HookBase):
    def __init__(self, eval_dataset_name, patience=2, min_delta=0.001):
        self.eval_dataset_name = eval_dataset_name
        self.results = []
        self.patience = patience
        self.min_delta = min_delta
        self.best_metric = None
        self.counter = 0

    def after_step(self):
        if (self.trainer.iter + 1) % self.trainer.cfg.TEST.EVAL_PERIOD == 0:
            evaluator = COCOEvaluator(self.eval_dataset_name, output_dir=self.trainer.cfg.OUTPUT_DIR)
            val_loader = build_detection_test_loader(self.trainer.cfg, self.eval_dataset_name)
            metrics = inference_on_dataset(self.trainer.model, val_loader, evaluator)

            print("Metrics:", metrics)

            result = {
                'iteration': self.trainer.iter + 1,
                'train/total_loss': self.trainer.storage.history("total_loss").values()[-1][0] if "total_loss" in self.trainer.storage.histories() else None,
                'train/loss_cls': self.trainer.storage.history("loss_cls").values()[-1][0] if "loss_cls" in self.trainer.storage.histories() else None,
                'train/loss_box_reg': self.trainer.storage.history("loss_box_reg").values()[-1][0] if "loss_box_reg" in self.trainer.storage.histories() else None,
                'train/loss_rpn_cls': self.trainer.storage.history("loss_rpn_cls").values()[-1][0] if "loss_rpn_cls" in self.trainer.storage.histories() else None,
                'train/loss_rpn_loc': self.trainer.storage.history("loss_rpn_loc").values()[-1][0] if "loss_rpn_loc" in self.trainer.storage.histories() else None,
                'lr': self.trainer.optimizer.param_groups[0]['lr'],  # Correctly extract learning rate
                'AP': metrics['bbox']['AP'] if 'bbox' in metrics else None,
                'AP50': metrics['bbox']['AP50'] if 'bbox' in metrics else None,
                'AP75': metrics['bbox']['AP75'] if 'bbox' in metrics else None,
                'APs': metrics['bbox']['APs'] if 'bbox' in metrics else None,
                'APm': metrics['bbox']['APm'] if 'bbox' in metrics else None,
                'APl': metrics['bbox']['APl'] if 'bbox' in metrics else None,
            }
            print("Result:", result)
            self.results.append(result)
```

```
            current_metric = metrics['bbox']['AP'] if 'bbox' in metrics else None
            if self.best_metric is None:
                self.best_metric = current_metric
            elif current_metric < self.best_metric - self.min_delta:
                self.counter += 1
                if self.counter >= self.patience:
                    print("Early stopping triggered")
                    self.trainer.iter = self.trainer.max_iter  # Set current iteration to max_iter to stop training
            else:
                self.best_metric = current_metric
                self.counter = 0

    def after_train(self):
        df = pd.DataFrame(self.results)
        df.to_excel(os.path.join(self.trainer.cfg.OUTPUT_DIR, "training_results.xlsx"), index=False)
        self.plot_curves(df)

    # Plotting function to visualize training metrics and evaluation metrics
    def plot_curves(self, df):
        metrics = ['train/total_loss', 'train/loss_cls', 'train/loss_box_reg', 'train/loss_rpn_cls',
'train/loss_rpn_loc', 'lr']
        for metric in metrics:
            if metric in df.columns:
                plt.figure()
                plt.plot(df['iteration'], df[metric], label=metric)
                plt.xlabel('Iteration')
                plt.ylabel(metric)
                plt.legend()
                plt.title(f'{metric} Curve')
                metric_dir = os.path.join(self.trainer.cfg.OUTPUT_DIR, os.path.dirname(metric))
                os.makedirs(metric_dir, exist_ok=True)
                plt.savefig(os.path.join(metric_dir, f"{os.path.basename(metric)}_curve.png"))
                plt.close()

        # Generate AP and AR curves
        ap_metrics = ['AP', 'AP50', 'AP75', 'APs', 'APm', 'APl']

        for ap_metric in ap_metrics:
            if ap_metric in df.columns:
                ap_values = df[ap_metric]
                plt.figure()
                plt.plot(df['iteration'], ap_values, label=ap_metric)
                plt.xlabel('Iteration')
                plt.ylabel(ap_metric)
                plt.legend()
                plt.title(f'{ap_metric} Curve')
                plt.savefig(os.path.join(self.trainer.cfg.OUTPUT_DIR, f"{ap_metric}_curve.png"))
                plt.close()

        # Placeholder labels for precision-recall curve
        y_true = [1] * len(df)  # Ensure y_true has the same length as df
        y_scores = df['AP50']

        # Correctly calculate precision and recall
        precision, recall, _ = precision_recall_curve(y_true, y_scores)
        plt.figure()
        plt.plot(recall, precision, label='Precision-Recall Curve')
        plt.xlabel('Recall')
```

```
        plt.ylabel('Precision')
        plt.legend()
        plt.title('Precision-Recall Curve')
        plt.savefig(os.path.join(self.trainer.cfg.OUTPUT_DIR, "precision_recall_curve.png"))
        plt.close()

        # Correctly calculate F1 scores
        f1_scores = [f1_score([1], [1 if score >= 0.5 else 0], average='binary') for score in y_scores]

        plt.figure()
        plt.plot(df['iteration'], f1_scores, label='F1 Score')
        plt.xlabel('Iteration')
        plt.ylabel('F1 Score')
        plt.legend()
        plt.title('F1 Score Curve')
        plt.savefig(os.path.join(self.trainer.cfg.OUTPUT_DIR, "f1_score_curve.png"))
        plt.close()

# Function to setup configuration
def setup_config(model_name, output_subdir, optimizer="SGD", lr_scheduler=None):
    cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file(model_name))
    cfg.DATASETS.TRAIN = ("mango_train",)  # Name of the training dataset
    cfg.DATASETS.TEST = ("mango_val",)  # Name of the validation dataset
    cfg.DATALOADER.NUM_WORKERS = 2
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(model_name)
    cfg.SOLVER.IMS_PER_BATCH = 2
    cfg.SOLVER.BASE_LR = 0.00025
    cfg.SOLVER.MAX_ITER = 400
    cfg.SOLVER.STEPS = [100, 300]  # Learning rate decay milestones
    cfg.SOLVER.GAMMA = 0.1  # Learning rate decay factor
    cfg.SOLVER.WARMUP_FACTOR = 1.0 / 1000
    cfg.SOLVER.WARMUP_ITERS = 1000
    cfg.SOLVER.WARMUP_METHOD = "linear"
    cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
    cfg.TEST.EVAL_PERIOD = 50
    cfg.OUTPUT_DIR = os.path.join("Local Dataset DETECTRON2 Results", output_subdir)  # Output
directory for results
    if optimizer == "AdamW":
        cfg.SOLVER.OPTIMIZER = "AdamW"
        cfg.SOLVER.WEIGHT_DECAY = 0.01
    if lr_scheduler == "WarmupCosineLR":
        cfg.SOLVER.LR_SCHEDULER_NAME = "WarmupCosineLR"
    if lr_scheduler == "WarmupMultiStepLR":
        cfg.SOLVER.LR_SCHEDULER_NAME = "WarmupMultiStepLR"
    return cfg

# Function to train the model
def train_model(model_name, output_subdir, optimizer="SGD", lr_scheduler=None):
    # Register COCO format datasets
    if "mango_train" not in DatasetCatalog.list():
        register_coco_instances("mango_train", {}, "F:/Final_year_project/collected
dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/train/_annotations.coco.json",
"F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/train")
    if "mango_val" not in DatasetCatalog.list():
```

```
    register_coco_instances("mango_val", {}, "F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/valid/_annotations.coco.json", "F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/valid")
  if "mango_test" not in DatasetCatalog.list():
    register_coco_instances("mango_test", {}, "F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/test/_annotations.coco.json", "F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/test")

  cfg = setup_config(model_name, output_subdir, optimizer, lr_scheduler)

  os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)  # Create output directory if it doesn't exist

  trainer = TrainerWithEval(cfg)
  trainer.resume_or_load(resume=False)
  trainer.register_hooks([CustomHookWithEarlyStopping("mango_val")])
  trainer.train()

  # Save evaluation results
  evaluator = COCOEvaluator("mango_val", output_dir=cfg.OUTPUT_DIR)
  val_loader = build_detection_test_loader(cfg, "mango_val")
  evaluation_results = inference_on_dataset(trainer.model, val_loader, evaluator)
  with open(os.path.join(cfg.OUTPUT_DIR, "evaluation_results.json"), "w") as f:
    json.dump(evaluation_results, f)

if __name__ == '__main__':
  import multiprocessing as mp
  mp.set_start_method('spawn', force=True)  # Required for Windows
  train_model("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml", "resnet50_sgd")  # Train model with SGD optimizer
  train_model("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml", "resnet50_adamw", optimizer="AdamW")  # Train model with AdamW optimizer
  train_model("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml", "resnet50_cosine", optimizer="SGD", lr_scheduler="WarmupCosineLR")  # Train model with Warmup Cosine LR scheduler
  train_model("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml", "resnet50_multistep", optimizer="SGD", lr_scheduler="WarmupMultiStepLR")  # Train model with Warmup MultiStep LR scheduler
```

local dataset handling, fix_annotations.py:

```
import json

def fix_annotations(annotation_file):
  with open(annotation_file, 'r') as f:
    data = json.load(f)

  # Fix the categories to have unique IDs and names
  unique_categories = {}
  for category in data['categories']:
    unique_categories[category['name']] = category['id']

  # Ensure that all categories have unique IDs
  new_categories = []
  id_mapping = {}
  new_id = 1
  for name, old_id in unique_categories.items():
    new_categories.append({'id': new_id, 'name': name})
```

```
            id_mapping[old_id] = new_id
            new_id += 1

        data['categories'] = new_categories

        # Remap the category_id in annotations to the new unique IDs
        for annotation in data['annotations']:
            annotation['category_id'] = id_mapping[annotation['category_id']]

        with open(annotation_file, 'w') as f:
            json.dump(data, f, indent=4)

        print(f"Fixed annotations in {annotation_file}")

# Paths to your COCO JSON files
annotation_files = [
    'F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/train/_annotations.coco.json',
    'F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/test/_annotations.coco.json',
    'F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/valid/_annotations.coco.json'
]

# Fix the annotations for each file
for annotation_file in annotation_files:
    fix_annotations(annotation_file)
```

Local dataset fixing proof:

```
F:\Final_year_project\collected dataset\Detectron2>python fix_annotations.py
Fixed annotations in F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/train/_annotations.coco.json
Fixed annotations in F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/test/_annotations.coco.json
Fixed annotations in F:/Final_year_project/collected dataset/Detectron2/Mango Fruit Detection fyp.v5i.coco/valid/_annotations.coco.json
```

For sydney dataset conversion, this code: detectron2_dataset_preparation.ipynb:

```
from google.colab import drive
drive.mount('/content/drive')
import os
import json
import shutil
from PIL import Image
from google.colab import drive

# Function to convert YOLO annotations to COCO format
def yolo_to_coco(images_dir, yolo_dir, output_json_path, output_image_dir):
    coco_format = {
        "images": [],
        "annotations": [],
        "categories": [{"id": 1, "name": "mango"}]
    }

    annotation_id = 1
    image_id = 1
```

```
    for yolo_file in os.listdir(yolo_dir):
        if yolo_file.endswith('.txt'):
            img_file_name = yolo_file.replace('.txt', '.png')
            img_path = os.path.join(images_dir, img_file_name)
            output_image_path = os.path.join(output_image_dir, img_file_name)

            try:
                img = Image.open(img_path)
                img_width, img_height = img.size

                coco_format["images"].append({
                    "id": image_id,
                    "file_name": img_file_name,
                    "width": img_width,
                    "height": img_height
                })

                # Copy the image to the output directory
                shutil.copyfile(img_path, output_image_path)

                yolo_file_path = os.path.join(yolo_dir, yolo_file)
                with open(yolo_file_path, 'r') as f:
                    for line in f:
                        class_id, center_x, center_y, width, height = map(float, line.split())
                        x_min = (center_x - width / 2) * img_width
                        y_min = (center_y - height / 2) * img_height
                        width = width * img_width
                        height = height * img_height

                        coco_format["annotations"].append({
                            "id": annotation_id,
                            "image_id": image_id,
                            "category_id": 1,
                            "bbox": [x_min, y_min, width, height],
                            "area": width * height,
                            "iscrowd": 0
                        })
                        annotation_id += 1

                image_id += 1
                print(f"Processed {img_file_name}")

            except FileNotFoundError:
                print(f"Image {img_path} not found. Skipping.")

    with open(output_json_path, 'w') as f:
        json.dump(coco_format, f, indent=4)
    print(f"COCO JSON saved to {output_json_path}")

# Function to convert each split to COCO format
def convert_split_to_coco(base_dir, split_name):
    print(f"Converting {split_name} split...")
    images_dir = os.path.join(base_dir, split_name, 'images')
    yolo_dir = os.path.join(base_dir, split_name, 'labels')
    output_dir = os.path.join('detectron2_split_dataset', split_name)
    os.makedirs(output_dir, exist_ok=True)
    output_image_dir = os.path.join(output_dir, 'images')
```

```
    os.makedirs(output_image_dir, exist_ok=True)
    output_json_path = os.path.join(output_dir, f'annotations_{split_name}.json')
    yolo_to_coco(images_dir, yolo_dir, output_json_path, output_image_dir)
    print(f"{split_name} split conversion complete.")

# Function to move the converted dataset to Google Drive
def move_dataset_to_drive(drive_base_dir):
    local_base_dir = 'detectron2_split_dataset'
    drive_path = os.path.join(drive_base_dir, 'detectron2_split_dataset')
    if not os.path.exists(drive_path):
        os.makedirs(drive_path)

    for split in ['train', 'val', 'test']:
        src_dir = os.path.join(local_base_dir, split)
        dest_dir = os.path.join(drive_path, split)
        if not os.path.exists(dest_dir):
            os.makedirs(dest_dir)
        for item in os.listdir(src_dir):
            s = os.path.join(src_dir, item)
            d = os.path.join(dest_dir, item)
            if os.path.isdir(s):
                shutil.copytree(s, d)
            else:
                shutil.copy2(s, d)
    print(f"Dataset moved to {drive_path}")

# Main function to convert all splits and move to Google Drive
def main():
    drive.mount('/content/drive')
    base_dir = '/content/drive/My Drive/acfr-multifruit-2016/mangoes/split_dataset'
    for split in ['train', 'val', 'test']:
        convert_split_to_coco(base_dir, split)

    # Move the dataset to Google Drive
    drive_base_dir = '/content/drive/My Drive'
    move_dataset_to_drive(drive_base_dir)

if __name__ == '__main__':
    main()
import os
import shutil
from google.colab import drive

# Function to copy images to Google Drive
def copy_images_to_drive(local_base_dir, drive_base_dir):
    for split in ['train', 'val', 'test']:
        local_images_dir = os.path.join(local_base_dir, split, 'images')
        drive_images_dir = os.path.join(drive_base_dir, 'detectron2_split_dataset', split, 'images')

        if not os.path.exists(drive_images_dir):
            os.makedirs(drive_images_dir)

        for image_file in os.listdir(local_images_dir):
            local_image_path = os.path.join(local_images_dir, image_file)
            drive_image_path = os.path.join(drive_images_dir, image_file)
            shutil.copy2(local_image_path, drive_image_path)
            print(f"Copied {image_file} to {drive_images_dir}")
```

```
# Mount Google Drive
drive.mount('/content/drive')

# Define local and drive directories
local_base_dir = 'detectron2_split_dataset'
drive_base_dir = '/content/drive/My Drive'

# Copy images to Google Drive
copy_images_to_drive(local_base_dir, drive_base_dir)
```

this code was used to extract frames from video , to help ger more pictures fast:
 # Filename: extract_frames.py

```
import cv2
import os

def extract_frames(video_path, output_folder, interval=2):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Open the video file
    video_capture = cv2.VideoCapture(video_path)
    fps = video_capture.get(cv2.CAP_PROP_FPS)  # Frames per second
    frame_interval = int(fps * interval)  # Interval in frames

    success, frame = video_capture.read()
    count = 0
    saved_frame_count = 0

    while success:
        if count % frame_interval == 0:
            frame_filename = os.path.join(output_folder, f"frame_{saved_frame_count:04d}.jpg")
            cv2.imwrite(frame_filename, frame)
            print(f"Saved {frame_filename}")
            saved_frame_count += 1
        success, frame = video_capture.read()
        count += 1

    video_capture.release()
    print(f"Frame extraction completed for {video_path}.")

# Define the list of video paths
video_paths = [
    "D:/2705/SDXC/DCIM/100MEDIA/DJI_0026.MP4",
    "D:/2705/SDXC/DCIM/100MEDIA/DJI_0027.MP4",
    "D:/2705/SDXC/DCIM/100MEDIA/DJI_0028.MP4",
    "D:/2705/SDXC/DCIM/100MEDIA/DJI_0095.MP4",
    "D:/2705/SDXC/DCIM/100MEDIA/DJI_0107.MP4"
]

# Base output folder
base_output_folder = "F:/Final_year_project/yolov8-env/frames"

# Extract frames from each video
for video_path in video_paths:
    video_name = os.path.splitext(os.path.basename(video_path))[0]
    output_folder = os.path.join(base_output_folder, video_name)
```

```
    extract_frames(video_path, output_folder, interval=2)
```

test detection on detectron2, test2.py:

```
import detectron2
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2 import model_zoo
from detectron2.utils.visualizer import Visualizer, ColorMode
from detectron2.data import MetadataCatalog
import cv2
import os

def setup_cfg():
    # Create a config object
    cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml"))

    # Absolute path to your trained model weights (fix the filename if needed)
    cfg.MODEL.WEIGHTS = os.path.abspath(r"F:\Mango Fruit Detection
project\Detectron2\Local_dataset_code\Local Dataset DETECTRON2
Results\resnet50_adamw\model_final.pth")

    # Set confidence threshold (adjust as needed)
    cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1

    # Get metadata (class names) from your training data
    mango_metadata = MetadataCatalog.get("mango_train")
    cfg.DATASETS.TEST = ("mango_test",)

    return cfg, mango_metadata

def detect_mangoes(predictor, image_path, metadata):
    print(f"Trying to load image from: {image_path}")
    # Load the image with error handling
    im = cv2.imread(image_path)

    if im is None:
        print(f"Error: Could not read image '{image_path}'.")
        print("Please check if the file exists and is in a supported format (JPG, PNG, etc.).")
        return  # Exit the function if image couldn't be loaded

    # Run inference/prediction
    outputs = predictor(im)

    # Visualization
    v = Visualizer(im[:, :, ::-1], metadata=metadata, scale=1.0,
instance_mode=ColorMode.SEGMENTATION)
    instances = outputs["instances"].to("cpu")
    boxes = instances.pred_boxes if instances.has("pred_boxes") else None
    scores = instances.scores if instances.has("scores") else None

    if boxes is not None and scores is not None:
        for box, score in zip(boxes, scores):
            box = box.numpy().tolist()
```

```
            score = float(score)
            label = f"mango: {score:.2f}"
            v.draw_box(box, edge_color="red", line_style="-")
            v.draw_text(label, (box[0], box[1]), color="red", font_size=15)

    result_image = v.output.get_image()[:, :, ::-1]

    # Save the image instead of displaying
    output_file = "output_image2.jpg"
    cv2.imwrite(output_file, result_image)
    print(f"Detection results saved to: {output_file}")

    # Uncomment the following lines if you want to display the image
    # cv2.imshow("Detection Results", result_image)
    # cv2.waitKey(0)  # Wait for key press before closing
    # cv2.destroyAllWindows()  # Close all OpenCV windows

if __name__ == "__main__":
    cfg, mango_metadata = setup_cfg()
    predictor = DefaultPredictor(cfg)

    # Update the image path
    image_path = r"F:\Mango Fruit Detection project\Detectron2\CQUniversity_dataset (need_to be_fixed)\Mango Fruit Detection.v3i.coco\train\azure_rgb_5fps_2500expo_182_png.rf.c0724bc74fc3c28eeaa9f73d0fa0d0f2.jpg"  # Absolute path to your test image

    detect_mangoes(predictor, image_path, mango_metadata)
```

video detection detection code, process_video_detectron2.py:

```
import detectron2
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2 import model_zoo
from detectron2.utils.visualizer import Visualizer, ColorMode
from detectron2.data import MetadataCatalog
import cv2
import os

def setup_cfg():
    # Create a config object
    cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))

    # Absolute path to your trained model weights (fix the filename if needed)
    cfg.MODEL.WEIGHTS = os.path.abspath(r"F:\Mango Fruit Detection project\Video detection\model_final.pth")

    # Set confidence threshold (adjust as needed)
    cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1

    # Get metadata (class names) from your training data
    mango_metadata = MetadataCatalog.get("mango_train")
    cfg.DATASETS.TEST = ("mango_test",)
```

```
    return cfg, mango_metadata

def detect_mangoes_on_frame(predictor, frame, metadata):
    # Run inference/prediction
    outputs = predictor(frame)

    # Visualization
    v = Visualizer(frame[:, :, ::-1], metadata=metadata, scale=1.0,
instance_mode=ColorMode.SEGMENTATION)
    instances = outputs["instances"].to("cpu")
    boxes = instances.pred_boxes if instances.has("pred_boxes") else None
    scores = instances.scores if instances.has("scores") else None

    if boxes is not None and scores is not None:
        for box, score in zip(boxes, scores):
            box = box.numpy().tolist()
            score = float(score)
            label = f"mango: {score:.2f}"
            v.draw_box(box, edge_color="red", line_style="-")
            v.draw_text(label, (box[0], box[1]), color="red", font_size=15)

    result_frame = v.output.get_image()[:, :, ::-1]
    return result_frame

def process_video(input_video_path, output_video_path, predictor, metadata):
    # Open the video file
    cap = cv2.VideoCapture(input_video_path)

    # Get video properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Define the codec and create VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))

    # Process each frame
    frame_number = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # Detect mangoes on the current frame
        processed_frame = detect_mangoes_on_frame(predictor, frame, metadata)

        # Write the processed frame to the output video
        out.write(processed_frame)

        frame_number += 1
        print(f"Processed frame {frame_number}/{total_frames}")

    # Release everything if job is finished
    cap.release()
    out.release()
    print(f"Detection results saved to: {output_video_path}")
```

```
if __name__ == "__main__":
    cfg, mango_metadata = setup_cfg()
    predictor = DefaultPredictor(cfg)

    # Update the video paths
    input_video_path = r"F:\Mango Fruit Detection project\Video detection\DJI_0095.MP4"  # Absolute path to your input video
    output_video_path = r"F:\Mango Fruit Detection project\Video detection\output_video2.mp4"  # Absolute path to save the output video

    process_video(input_video_path, output_video_path, predictor, mango_metadata)
```

Now yolo models was used for three dataset for other datasets change the paths, starting yolov8 50 epochs.py default model:

```
import os
from ultralytics import YOLO

def train_model():
    # Paths to dataset
    train_images_dir = r"F:\Final_year_project\collected dataset\third time\train\images"
    val_images_dir = r"F:\Final_year_project\collected dataset\third time\valid\images"
    test_images_dir = r"F:\Final_year_project\collected dataset\third time\test\images"
    train_labels_dir = r"F:\Final_year_project\collected dataset\third time\train\labels"
    val_labels_dir = r"F:\Final_year_project\collected dataset\third time\valid\labels"
    test_labels_dir = r"F:\Final_year_project\collected dataset\third time\test\labels"

    # Ensure directories exist
    for directory in [train_images_dir, val_images_dir, test_images_dir, train_labels_dir, val_labels_dir, test_labels_dir]:
        if not os.path.exists(directory):
            print(f"Error: {directory} does not exist.")
            exit(1)

    # Define the data.yaml configuration for YOLOv8
    data_yaml_content = f"""
    train: {train_images_dir}
    val: {val_images_dir}
    test: {test_images_dir}
    nc: 1
    names: ['mango']
    """

    # Write data.yaml file
    data_yaml_path = r"F:\Final_year_project\collected dataset\third time\data.yaml"
    with open(data_yaml_path, 'w') as f:
        f.write(data_yaml_content)

    # Initialize a new YOLOv8 model
    model = YOLO('yolov8n.pt')  # Using a YOLOv8 pre-trained model for initialization

    # Train the model for 50 epochs
    model.train(data=data_yaml_path, epochs=50, imgsz=640, batch=16, name='third_time_yolov8_mango_model', project=r'F:\Final_year_project\collected dataset\third time')

    print("Training completed!")
```

```
if __name__ == '__main__':
    train_model()
```

train_with_early_stopping.py:

```
from ultralytics import YOLO

def train_model():
    # Load the YOLO model
    model = YOLO('yolov8n.pt')

    # Train the model with early stopping
    model.train(
        data='F:/Final_year_project/collected dataset/third time/data.yaml',  # Path to the data.yaml file
        epochs=200,  # Maximum number of epochs
        patience=10,  # Number of epochs with no improvement after which training will be stopped
        batch=16,  # Batch size
        imgsz=640,  # Image size
        save=True,
        project='F:/Final_year_project/collected dataset/third time/early_stopping_training',  # Directory to save the training results
        name='early_stopping_yolov8_mango_model',  # Name of the training run
    )

if __name__ == '__main__':
    import multiprocessing
    multiprocessing.freeze_support()
    train_model()
```

augmentated.py:

```
from ultralytics import YOLO

def train_model():
    # Load the YOLO model
    model = YOLO('yolov8n.pt')

    # Train the model with early stopping and augmentation
    model.train(
        data='F:/Final_year_project/collected dataset/third time/data.yaml',  # Path to the data.yaml file
        epochs=200,  # Maximum number of epochs
        patience=10,  # Number of epochs with no improvement after which training will be stopped
        batch=16,  # Batch size
        imgsz=640,  # Image size
        save=True,
        project='F:/Final_year_project/collected dataset/third time/early_stopping_training',  # Directory to save the training results
        name='early_stopping_yolov8_mango_model',  # Name of the training run
        hsv_h=0.015,  # Hue augmentation (fraction)
        hsv_s=0.7,    # Saturation augmentation (fraction)
        hsv_v=0.4,    # Value augmentation (fraction)
        degrees=10.0,  # Image rotation (degrees)
        translate=0.1,  # Image translation (fraction)
        scale=0.5,    # Image scale (fraction)
        shear=2.0,    # Image shear (degrees)
        fliplr=0.5,   # Horizontal flip (probability)
        mosaic=0.5,   # Mosaic augmentation (probability)
```

```
        mixup=0.2,    # Mixup augmentation (probability)
    )

if __name__ == '__main__':
    import multiprocessing
    multiprocessing.freeze_support()
    train_model()
```

SGD_optimizer_with_augmentation.py

```
from ultralytics import YOLO

def train_model_with_augmentation():
    # Load the YOLO model
    model = YOLO('yolov8n.pt')

    # Train the model with early stopping, augmentation, and SGD optimizer
    model.train(
        data='F:/Final_year_project/collected dataset/third time/data.yaml',  # Path to the data.yaml file
        epochs=200,  # Maximum number of epochs
        patience=10,  # Number of epochs with no improvement after which training will be stopped
        batch=16,  # Batch size
        imgsz=640,  # Image size
        save=True,
        project='F:/Final_year_project/collected dataset/third time/early_stopping_training',  # Directory to save the training results
        name='early_stopping_yolov8_mango_model_sgd_aug',  # Name of the training run
        optimizer='SGD',  # Specify the SGD optimizer
        lr0=0.01,  # Initial learning rate
        momentum=0.9,  # Momentum for SGD
        weight_decay=0.0005,  # Weight decay
        hsv_h=0.015,  # Hue augmentation (fraction)
        hsv_s=0.7,    # Saturation augmentation (fraction)
        hsv_v=0.4,    # Value augmentation (fraction)
        degrees=10.0,  # Image rotation (degrees)
        translate=0.1,  # Image translation (fraction)
        scale=0.5,    # Image scale (fraction)
        shear=2.0,    # Image shear (degrees)
        fliplr=0.5,   # Horizontal flip (probability)
        mosaic=0.5,   # Mosaic augmentation (probability)
        mixup=0.2,    # Mixup augmentation (probability)
    )

if __name__ == '__main__':
    import multiprocessing
    multiprocessing.freeze_support()
    train_model_with_augmentation()
```

SGD_optimizer_without_augmentation.py

```
from ultralytics import YOLO

def train_model_without_augmentation():
    # Load the YOLO model
    model = YOLO('yolov8n.pt')

    # Train the model with early stopping and SGD optimizer without augmentation
```

```
    model.train(
        data='F:/Final_year_project/collected dataset/third time/data.yaml',  # Path to the data.yaml
file
        epochs=200,  # Maximum number of epochs
        patience=10,  # Number of epochs with no improvement after which training will be stopped
        batch=16,  # Batch size
        imgsz=640,  # Image size
        save=True,
        project='F:/Final_year_project/collected dataset/third time/early_stopping_training',  #
Directory to save the training results
        name='early_stopping_yolov8_mango_model_sgd_no_aug',  # Name of the training run
        optimizer='SGD',  # Specify the SGD optimizer
        lr0=0.01,  # Initial learning rate
        momentum=0.9,  # Momentum for SGD
        weight_decay=0.0005  # Weight decay
    )

if __name__ == '__main__':
    import multiprocessing
    multiprocessing.freeze_support()
    train_model_without_augmentation()
```

YOLOv10, default.py:

```
import os
import urllib.request
from ultralytics import YOLO

def setup_directories(base_dir):
    """
    Ensure necessary directories exist for training, validation, and testing images and labels.
    """
    directories = [
        os.path.join(base_dir, "train", "images"),
        os.path.join(base_dir, "valid", "images"),
        os.path.join(base_dir, "test", "images"),
        os.path.join(base_dir, "train", "labels"),
        os.path.join(base_dir, "valid", "labels"),
        os.path.join(base_dir, "test", "labels"),
    ]

    for directory in directories:
        if not os.path.exists(directory):
            print(f"Error: {directory} does not exist.")
            return False
    return True

def create_data_yaml(base_dir):
    """
    Create the data.yaml configuration file for YOLOv10.
    """
    data_yaml_content = f"""
    train: {os.path.join(base_dir, 'train', 'images')}
    val: {os.path.join(base_dir, 'valid', 'images')}
    test: {os.path.join(base_dir, 'test', 'images')}
    nc: 1
    names: ['mango']
    """
```

```
    data_yaml_path = os.path.join(base_dir, "data.yaml")
    try:
        with open(data_yaml_path, 'w') as f:
            f.write(data_yaml_content)
    except IOError as e:
        print(f"Error writing data.yaml file: {e}")
        return None
    return data_yaml_path

def download_weights(weights_dir):
    """
    Download the pre-trained YOLOv10 weights.
    """
    os.makedirs(weights_dir, exist_ok=True)
    url = "https://github.com/THU-MIG/yolov10/releases/download/v1.0/yolov10n.pt"
    weight_path = os.path.join(weights_dir, "yolov10n.pt")
    urllib.request.urlretrieve(url, weight_path)
    return weight_path

def train_model():
    base_dir = r"F:\Final_year_project\collected dataset\third time"
    output_dir = os.path.join(base_dir, "yolov10 results")
    os.makedirs(output_dir, exist_ok=True)

    if not setup_directories(base_dir):
        return

    data_yaml_path = create_data_yaml(base_dir)
    if not data_yaml_path:
        return

    weights_dir = os.path.join(base_dir, "weights")
    weight_path = download_weights(weights_dir)

    # Initialize YOLOv10 model with pre-trained weights
    model = YOLO(weight_path)

    # Train the model
    model.train(data=data_yaml_path, epochs=50, imgsz=640, batch=16,
name='yolov10_mango_model', project=output_dir)

    print("Training completed!")

if __name__ == '__main__':
    train_model()
```

earlystopping_augmentated.py:

```
import os
import urllib.request
from ultralytics import YOLO
import multiprocessing

def setup_directories(base_dir):
    """
    Ensure necessary directories exist for training, validation, and testing images and labels.
    """
    directories = [
        os.path.join(base_dir, "train", "images"),
```

```
        os.path.join(base_dir, "valid", "images"),
        os.path.join(base_dir, "test", "images"),
        os.path.join(base_dir, "train", "labels"),
        os.path.join(base_dir, "valid", "labels"),
        os.path.join(base_dir, "test", "labels"),
    ]

    for directory in directories:
        if not os.path.exists(directory):
            print(f"Error: {directory} does not exist.")
            return False
    return True

def create_data_yaml(base_dir):
    """
    Create the data.yaml configuration file for YOLOv10.
    """
    data_yaml_content = f"""
    train: {os.path.join(base_dir, 'train', 'images')}
    val: {os.path.join(base_dir, 'valid', 'images')}
    test: {os.path.join(base_dir, 'test', 'images')}
    nc: 1
    names: ['mango']
    """

    data_yaml_path = os.path.join(base_dir, "data.yaml")
    try:
        with open(data_yaml_path, 'w') as f:
            f.write(data_yaml_content)
    except IOError as e:
        print(f"Error writing data.yaml file: {e}")
        return None
    return data_yaml_path

def download_weights(weights_dir):
    """
    Download the pre-trained YOLOv10 weights.
    """
    os.makedirs(weights_dir, exist_ok=True)
    url = "https://github.com/THU-MIG/yolov10/releases/download/v1.0/yolov10n.pt"
    weight_path = os.path.join(weights_dir, "yolov10n.pt")
    urllib.request.urlretrieve(url, weight_path)
    return weight_path

def train_model():
    base_dir = r"F:\Final_year_project\collected dataset\third time"
    output_dir = os.path.join(base_dir, "early_stopping_training")
    os.makedirs(output_dir, exist_ok=True)

    if not setup_directories(base_dir):
        return

    data_yaml_path = create_data_yaml(base_dir)
    if not data_yaml_path:
        return

    weights_dir = os.path.join(base_dir, "weights")
    weight_path = download_weights(weights_dir)
```

```
    # Initialize YOLOv10 model with pre-trained weights
    model = YOLO(weight_path)

    # Train the model with early stopping and augmentation
    model.train(
        data=data_yaml_path,
        epochs=200,  # Maximum number of epochs
        patience=10,  # Number of epochs with no improvement after which training will be stopped
        batch=16,
        imgsz=640,
        save=True,
        project=output_dir,
        name='early_stopping_yolov10_mango_model',
        hsv_h=0.015,  # Hue augmentation (fraction)
        hsv_s=0.7,    # Saturation augmentation (fraction)
        hsv_v=0.4,    # Value augmentation (fraction)
        degrees=10.0,  # Image rotation (degrees)
        translate=0.1,  # Image translation (fraction)
        scale=0.5,    # Image scale (fraction)
        shear=2.0,    # Image shear (degrees)
        fliplr=0.5,   # Horizontal flip (probability)
        mosaic=0.5,   # Mosaic augmentation (probability)
        mixup=0.2,    # Mixup augmentation (probability)
    )

    print("Training completed!")

if __name__ == '__main__':
    multiprocessing.freeze_support()
    train_model()
```

SGD_optimizer_with_augmentation.py:

```
import os
import urllib.request
from ultralytics import YOLO
import multiprocessing

def setup_directories(base_dir):
    """
    Ensure necessary directories exist for training, validation, and testing images and labels.
    """
    directories = [
        os.path.join(base_dir, "train", "images"),
        os.path.join(base_dir, "valid", "images"),
        os.path.join(base_dir, "test", "images"),
        os.path.join(base_dir, "train", "labels"),
        os.path.join(base_dir, "valid", "labels"),
        os.path.join(base_dir, "test", "labels"),
    ]

    for directory in directories:
        if not os.path.exists(directory):
            print(f"Error: {directory} does not exist.")
            return False
    return True

def create_data_yaml(base_dir):
```

```
    """
    Create the data.yaml configuration file for YOLOv10.
    """
    data_yaml_content = f"""
    train: {os.path.join(base_dir, 'train', 'images')}
    val: {os.path.join(base_dir, 'valid', 'images')}
    test: {os.path.join(base_dir, 'test', 'images')}
    nc: 1
    names: ['mango']
    """

    data_yaml_path = os.path.join(base_dir, "data.yaml")
    try:
        with open(data_yaml_path, 'w') as f:
            f.write(data_yaml_content)
    except IOError as e:
        print(f"Error writing data.yaml file: {e}")
        return None
    return data_yaml_path

def download_weights(weights_dir):
    """
    Download the pre-trained YOLOv10 weights.
    """
    os.makedirs(weights_dir, exist_ok=True)
    url = "https://github.com/THU-MIG/yolov10/releases/download/v1.0/yolov10n.pt"
    weight_path = os.path.join(weights_dir, "yolov10n.pt")
    urllib.request.urlretrieve(url, weight_path)
    return weight_path

def train_model_with_augmentation():
    base_dir = r"F:\Final_year_project\collected dataset\third time"
    output_dir = os.path.join(base_dir, "early_stopping_training")
    os.makedirs(output_dir, exist_ok=True)

    if not setup_directories(base_dir):
        return

    data_yaml_path = create_data_yaml(base_dir)
    if not data_yaml_path:
        return

    weights_dir = os.path.join(base_dir, "weights")
    weight_path = download_weights(weights_dir)

    # Initialize YOLOv10 model with pre-trained weights
    model = YOLO(weight_path)

    # Train the model with early stopping, augmentation, and SGD optimizer
    model.train(
        data=data_yaml_path,
        epochs=200,  # Maximum number of epochs
        patience=10,  # Number of epochs with no improvement after which training will be stopped
        batch=16,  # Batch size
        imgsz=640,  # Image size
        save=True,
        project=output_dir,
        name='early_stopping_yolov10_mango_model_sgd_aug',  # Name of the training run
```

```
        optimizer='SGD',  # Specify the SGD optimizer
        lr0=0.01,  # Initial learning rate
        momentum=0.9,  # Momentum for SGD
        weight_decay=0.0005,  # Weight decay
        hsv_h=0.015,  # Hue augmentation (fraction)
        hsv_s=0.7,    # Saturation augmentation (fraction)
        hsv_v=0.4,    # Value augmentation (fraction)
        degrees=10.0,  # Image rotation (degrees)
        translate=0.1,  # Image translation (fraction)
        scale=0.5,    # Image scale (fraction)
        shear=2.0,    # Image shear (degrees)
        fliplr=0.5,   # Horizontal flip (probability)
        mosaic=0.5,   # Mosaic augmentation (probability)
        mixup=0.2,    # Mixup augmentation (probability)
    )

    print("Training completed!")

if __name__ == '__main__':
    multiprocessing.freeze_support()
    train_model_with_augmentation()
```

Now gui file, main one Gui.py:

```
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog, ttk
from PIL import Image, ImageTk, ImageDraw, ImageFont
import time
import os

# Importing your custom modules
from translations import translations
from splash_screen import show_splash_screen
from toolbar import create_menu
from themes import Theme
from icons_buttons import Icons
from map_view import display_map_in_app
from video_utils import (
    select_video_file, process_selected_video, display_video_frame,
    detect_mangoes_in_video, save_annotated_video, play_video,
    pause_video, stop_video, play_next_frame
)
from image_utils import process_selected_image, display_image, annotate_image,
detect_mangoes_in_image, save_annotated_image
from ultralytics import YOLO

class MangoVisionApp:
    def __init__(self, root):
        self.root = root
        self.current_lang = "en"
        self.model1 = self.load_model(r'F:\Mango Fruit Detection
project\YOLO\CQUniversity_dataset\CQUniversity_dataset_results\yolov10\early_stopping_augment
ation\early_stopping_yolov10_mango_model\weights\best.pt')
        self.model2 = self.load_model(r'F:\Mango Fruit Detection
project\YOLO\Local_dataset\local_dataset_results\yolov10_results\early_stopping_yolov10_mango_
model_sgd_no_aug\weights\best.pt')
        self.current_model = self.model1
```

```
        self.confidence_threshold = 0.5
        self.summary = []
        self.current_gps_info = None
        self.is_video = False
        self.video_frames = []
        self.video_playing = False
        self.video_cap = None
        self.frame_interval = 2  # seconds

         # Add image_path to track the selected image path
        self.image_path = None

        # Theme and Icons
        self.theme = Theme()
        self.icons = Icons('F:\\Mango Fruit Detection project\\GUI\\icons\\')

        # Splash Screen
        show_splash_screen(self.root)

        # Initialize UI
        self.initialize_ui()

    def load_model(self, model_path):
        """Load the YOLO model from the specified path."""
        return YOLO(model_path)

    def get_text(self, key):
        """Get the translated text for the given key."""
        return translations[self.current_lang][key]

    def switch_language(self, lang):
        """Switch the application language."""
        self.current_lang = lang
        self.update_text()
        if self.summary:
            self.display_summary(self.summary)

    def update_text(self):
        """Update all text in the UI to match the current language."""
        self.root.title(self.get_text("title"))
        self.title_label.config(text=self.get_text("title"))
        self.subtitle_label.config(text=self.get_text("subtitle"))
        self.select_button.config(text=self.get_text("select_media"))
        self.detect_button.config(text=self.get_text("detect_mangoes"))
        self.save_button.config(text=self.get_text("save_results"))
        self.view_map_button.config(text=self.get_text("view_on_map"))
        self.clear_button.config(text=self.get_text("clear"))
        self.date_label_lbl.config(text=self.get_text("date"))
        self.time_label_lbl.config(text=self.get_text("time"))
        self.latitude_label.config(text=self.get_text("latitude"))
        self.longitude_label.config(text=self.get_text("longitude"))
        self.status_var.set(self.get_text("ready"))
        self.credit_label.config(text=self.get_text("created_by"), fg=self.theme.credit_text_color)
        self.model_menu.entryconfig(0, label=self.get_text("model_1"))
        self.model_menu.entryconfig(1, label=self.get_text("model_2"))
        self.threshold_menu.entryconfig(0, label=self.get_text("confidence_threshold"))
        self.image_frame.config(text=self.get_text("selected_media"))
        self.gps_frame.config(text=self.get_text("gps_coordinates"))
```

```
        self.datetime_frame.config(text=self.get_text("date_time"))
        self.summary_frame.config(text=self.get_text("detection_summary"))

    def initialize_ui(self):
        """Initialize the main UI components."""
        self.root.title(self.get_text("title"))
        self.root.geometry("1000x600")
        self.root.resizable(True, True)
        self.root.iconphoto(True, tk.PhotoImage(file='F:\\Final_year_project\\yolov8-
env\\my_yolov8_app\\icons\\mango.png'))

        self.theme.apply_theme(self.root)

        # Create a main container
        self.main_container = tk.Frame(self.root, bg=self.theme.bg_color)
        self.main_container.pack(fill=tk.BOTH, expand=True)

        # Add scrollbar
        self.v_scrollbar = ttk.Scrollbar(self.main_container, orient=tk.VERTICAL)
        self.v_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        # Create a canvas
        self.canvas = tk.Canvas(self.main_container, bg=self.theme.bg_color,
yscrollcommand=self.v_scrollbar.set)
        self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Configure scrollbar
        self.v_scrollbar.config(command=self.canvas.yview)

        # Create a frame inside the canvas
        self.scrollable_frame = tk.Frame(self.canvas, bg=self.theme.bg_color)
        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: self.canvas.configure(
                scrollregion=self.canvas.bbox("all")
            )
        )

        # Add the new frame to a window in the canvas
        self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")

        create_menu(self)
        self.create_status_bar()
        self.create_main_layout()

    def create_status_bar(self):
        """Create the status bar at the bottom of the window."""
        self.status_var = tk.StringVar()
        self.status_bar = tk.Label(self.scrollable_frame, textvariable=self.status_var, bd=1,
relief=tk.SUNKEN, anchor=tk.W, bg=self.theme.bg_color)
        self.status_bar.pack(side=tk.BOTTOM, fill=tk.X)
        self.status_var.set(self.get_text("ready"))

        self.credit_label = tk.Label(self.scrollable_frame, text=self.get_text("created_by"),
font=("Helvetica", 10), bg=self.theme.bg_color, fg=self.theme.credit_text_color)
        self.credit_label.pack(side=tk.BOTTOM, pady=5)

    def create_main_layout(self):
```

```
        """Create the main layout with top and main frames."""
        self.top_frame = tk.Frame(self.scrollable_frame, bg=self.theme.bg_color, padx=10, pady=10)
        self.top_frame.pack(side=tk.TOP, fill=tk.X)

        self.main_frame = tk.Frame(self.scrollable_frame, bg=self.theme.bg_color)
        self.main_frame.pack(fill=tk.BOTH, expand=True)

        self.title_label = tk.Label(self.top_frame, text=self.get_text("title"), font=self.theme.title_font,
bg=self.theme.bg_color)
        self.title_label.pack(side=tk.TOP, pady=5)

        self.subtitle_label = tk.Label(self.top_frame, text=self.get_text("subtitle"),
font=self.theme.subtitle_font, bg=self.theme.bg_color)
        self.subtitle_label.pack(side=tk.TOP, pady=5)

        self.create_left_frame()
        self.create_right_frame()

    def create_left_frame(self):
        """Create the left frame containing image and control buttons."""
        self.left_frame = tk.Frame(self.main_frame, bg=self.theme.bg_color)
        self.left_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        self.image_frame = tk.LabelFrame(self.left_frame, text=self.get_text("selected_media"), padx=10,
pady=10, bg=self.theme.frame_bg_color, font=self.theme.label_font, bd=2, relief=tk.GROOVE)
        self.image_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.gps_frame = tk.LabelFrame(self.left_frame, text=self.get_text("gps_coordinates"), padx=10,
pady=10, bg=self.theme.frame_bg_color, font=self.theme.label_font, bd=2, relief=tk.GROOVE)
        self.gps_frame.pack(fill=tk.X, pady=5, padx=10)

        self.latitude_label = tk.Label(self.gps_frame, text=self.get_text("latitude"),
font=self.theme.label_font, bg=self.theme.frame_bg_color)
        self.latitude_label.grid(row=0, column=0, sticky=tk.W, padx=5, pady=5)
        self.latitude_entry = tk.Entry(self.gps_frame, font=self.theme.entry_font, state='readonly',
width=20)
        self.latitude_entry.grid(row=0, column=1, padx=5, pady=5)

        self.longitude_label = tk.Label(self.gps_frame, text=self.get_text("longitude"),
font=self.theme.label_font, bg=self.theme.frame_bg_color)
        self.longitude_label.grid(row=0, column=2, sticky=tk.W, padx=5, pady=5)
        self.longitude_entry = tk.Entry(self.gps_frame, font=self.theme.entry_font, state='readonly',
width=20)
        self.longitude_entry.grid(row=0, column=3, padx=5, pady=5)

        self.view_map_button = tk.Button(self.gps_frame, text=self.get_text("view_on_map"),
image=self.icons.map_icon, compound="top", command=self.view_on_map,
font=self.theme.button_font, bg=self.theme.button_color, fg=self.theme.button_text_color,
state=tk.DISABLED)
        self.view_map_button.grid(row=0, column=4, padx=5, pady=5, rowspan=2)

        self.datetime_frame = tk.LabelFrame(self.left_frame, text=self.get_text("date_time"), padx=10,
pady=10, bg=self.theme.frame_bg_color, font=self.theme.label_font, bd=2, relief=tk.GROOVE)
        self.datetime_frame.pack(fill=tk.X, pady=5, padx=10)

        self.date_label_lbl = tk.Label(self.datetime_frame, text=self.get_text("date"),
font=self.theme.label_font, bg=self.theme.frame_bg_color)
        self.date_label_lbl.grid(row=0, column=0, sticky=tk.W, padx=5, pady=5)
```

```
        self.date_label = tk.Entry(self.datetime_frame, font=self.theme.entry_font, state='readonly', width=20)
        self.date_label.grid(row=0, column=1, padx=5, pady=5)

        self.time_label_lbl = tk.Label(self.datetime_frame, text=self.get_text("time"), font=self.theme.label_font, bg=self.theme.frame_bg_color)
        self.time_label_lbl.grid(row=0, column=2, sticky=tk.W, padx=5, pady=5)
        self.time_label = tk.Entry(self.datetime_frame, font=self.theme.entry_font, state='readonly', width=20)
        self.time_label.grid(row=0, column=3, padx=5, pady=5)

        self.button_frame = tk.Frame(self.left_frame, bg=self.theme.bg_color)
        self.button_frame.pack(fill=tk.X, pady=5)

        self.create_placeholder_image()

        self.image_label = tk.Label(self.image_frame, bg=self.theme.frame_bg_color, image=self.placeholder_image)
        self.image_label.pack(padx=10, pady=10)

        self.progress_bar = ttk.Progressbar(self.image_frame, orient=tk.HORIZONTAL, length=640, mode='determinate')
        self.progress_bar.pack(side=tk.BOTTOM, padx=5, pady=5)

        self.create_buttons()

    def create_placeholder_image(self):
        """Create a placeholder image with the text 'No Preview Available'."""
        self.placeholder_image = Image.new('RGB', (640, 360), color=(192, 192, 192))
        draw = ImageDraw.Draw(self.placeholder_image)
        font = ImageFont.load_default()  # Use default PIL font
        text = "No Preview Available"
        text_bbox = draw.textbbox((0, 0), text, font=font)
        textwidth = text_bbox[2] - text_bbox[0]
        textheight = text_bbox[3] - text_bbox[1]
        width, height = self.placeholder_image.size
        x = (width - textwidth) / 2
        y = (height - textheight) / 2
        draw.text((x, y), text, font=font, fill=(255, 255, 255))
        self.placeholder_image = ImageTk.PhotoImage(self.placeholder_image)

    def create_buttons(self):
        """Create and place the control buttons."""
        commands = {
            'select': self.select_media,
            'detect': self.process_image_for_mango_detection,
            'save': self.save_detection_results,
            'play': self.play_video,
            'pause': self.pause_video,
            'stop': self.stop_video,
            'clear': self.clear_data,
        }

        self.select_button = tk.Button(self.button_frame, text=self.get_text("select_media"), image=self.icons.upload_icon, compound="top", command=commands['select'], font=self.theme.button_font, bg=self.theme.button_color, fg=self.theme.button_text_color)
        self.detect_button = tk.Button(self.button_frame, text=self.get_text("detect_mangoes"), image=self.icons.detect_icon, compound="top", command=commands['detect'],
```

```
font=self.theme.button_font, bg=self.theme.button_color, fg=self.theme.button_text_color,
state=tk.DISABLED)
        self.save_button = tk.Button(self.button_frame, text=self.get_text("save_results"),
image=self.icons.save_icon, compound="top", command=commands['save'],
font=self.theme.button_font, bg=self.theme.button_color, fg=self.theme.button_text_color,
state=tk.DISABLED)
        self.clear_button = tk.Button(self.button_frame, text=self.get_text("clear"),
image=self.icons.clear_icon, compound="top", command=commands['clear'],
font=self.theme.button_font, bg="red", fg=self.theme.button_text_color)

        self.select_button.grid(row=0, column=0, padx=5, pady=5)
        self.detect_button.grid(row=0, column=1, padx=5, pady=5)
        self.save_button.grid(row=0, column=2, padx=5, pady=5)
        self.clear_button.grid(row=0, column=3, padx=5, pady=5)

        # Create video control buttons but initially hide them
        self.video_controls_frame = tk.Frame(self.image_frame, bg=self.theme.frame_bg_color)
        self.video_controls_frame.pack(fill=tk.X, padx=5, pady=5)

        self.play_button = tk.Button(self.video_controls_frame, text=self.get_text("play_video"),
image=self.icons.play_icon, compound="left", command=commands['play'],
font=self.theme.button_font, bg=self.theme.button_color, fg=self.theme.button_text_color,
state=tk.DISABLED)
        self.pause_button = tk.Button(self.video_controls_frame, text=self.get_text("pause_video"),
image=self.icons.pause_icon, compound="left", command=commands['pause'],
font=self.theme.button_font, bg=self.theme.button_color, fg=self.theme.button_text_color,
state=tk.DISABLED)
        self.stop_button = tk.Button(self.video_controls_frame, text=self.get_text("stop_video"),
image=self.icons.stop_icon, compound="left", command=commands['stop'],
font=self.theme.button_font, bg=self.theme.button_color, fg=self.theme.button_text_color,
state=tk.DISABLED)

        self.play_button.pack(side=tk.LEFT, padx=5, pady=5)
        self.pause_button.pack(side=tk.LEFT, padx=5, pady=5)
        self.stop_button.pack(side=tk.LEFT, padx=5, pady=5)

        # Hide the video control buttons initially
        self.video_controls_frame.pack_forget()

    def create_right_frame(self):
        """Create the right frame containing the detection summary."""
        self.right_frame = tk.Frame(self.main_frame, bg=self.theme.bg_color)
        self.right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        self.summary_frame = tk.LabelFrame(self.right_frame, text=self.get_text("detection_summary"),
padx=10, pady=10, bg=self.theme.frame_bg_color, font=self.theme.label_font, bd=2,
relief=tk.GROOVE)
        self.summary_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.summary_canvas = tk.Canvas(self.summary_frame, bg=self.theme.frame_bg_color)
        self.summary_scrollbar = ttk.Scrollbar(self.summary_frame, orient="vertical",
command=self.summary_canvas.yview)
        self.summary_inner_frame = tk.Frame(self.summary_canvas, bg=self.theme.frame_bg_color)

        self.summary_inner_frame.bind(
            "<Configure>",
            lambda e: self.summary_canvas.configure(
                scrollregion=self.summary_canvas.bbox("all")
```

```
            )
        )

        self.summary_canvas.create_window((0, 0), window=self.summary_inner_frame, anchor="nw")
        self.summary_canvas.configure(yscrollcommand=self.summary_scrollbar.set)

        self.summary_canvas.pack(side="left", fill="both", expand=True)
        self.summary_scrollbar.pack(side="right", fill="y")

    def select_media(self):
        """Open a file dialog to select an image or video file."""
        file_path = filedialog.askopenfilename(filetypes=[("Media files",
"*.jpg;*.jpeg;*.png;*.mp4;*.avi")])
        if file_path:
            if file_path.lower().endswith(('.mp4', '.avi')):
                self.status_var.set(self.get_text("loading_video"))
                process_selected_video(self, file_path)
            else:
                self.status_var.set(self.get_text("loading_image"))
                process_selected_image(self, file_path)

    def process_image_for_mango_detection(self):
        """Process the selected image or video for mango detection."""
        self.status_var.set(self.get_text("detecting_mangoes"))
        start_time = time.time()
        if self.is_video:
            detect_mangoes_in_video(self)
        else:
            detect_mangoes_in_image(self)
        end_time = time.time()
        detection_time = end_time - start_time
        self.status_var.set(f"{self.get_text('detection_complete')}
{self.get_text('detection_time')}{detection_time:.2f} seconds")

    def display_summary(self, summary):
        self.clear_summary()
        if summary:
            if self.is_video:
                summary_text = f"Total Mangoes Detected: {len(summary)}"
                tk.Label(self.summary_inner_frame, text=summary_text, font=self.theme.label_font,
bg=self.theme.frame_bg_color, fg="black").pack(anchor='w')
            else:
                for item in summary:
                    summary_text = f"Mango {item['Mango No.']}: {item['Confidence Score']}"
                    tk.Label(self.summary_inner_frame, text=summary_text, font=self.theme.label_font,
bg=self.theme.frame_bg_color, fg="black").pack(anchor='w')
                summary_text = f"Total Mangoes Detected: {len(summary)}"
                tk.Label(self.summary_inner_frame, text=summary_text, font=self.theme.label_font,
bg=self.theme.frame_bg_color, fg="black").pack(anchor='w')
        else:
            tk.Label(self.summary_inner_frame, text=self.get_text("no_mangoes_detected"),
font=self.theme.label_font, bg=self.theme.frame_bg_color, fg="black").pack(anchor='w')

    def save_detection_results(self):
        """Save the detection results."""
        try:
            if self.is_video:
```

```
            save_path = filedialog.asksaveasfilename(defaultextension=".mp4", filetypes=[("MP4 files",
"*.mp4"), ("AVI files", "*.avi")])
            if save_path:
                save_annotated_video(self, save_path)
        else:
            save_path = filedialog.asksaveasfilename(defaultextension=".jpg", filetypes=[("JPEG files",
"*.jpg"), ("PNG files", "*.png")])
            if save_path:
                save_annotated_image(self, save_path)
        if save_path:
            messagebox.showinfo(self.get_text("save_successful"),  f"{self.get_text('results_saved_to')}
{save_path}")
            self.status_var.set(self.get_text("results_saved"))
    except Exception as e:
        messagebox.showerror(self.get_text("error"), str(e))

def view_on_map(self):
    """View the detected GPS coordinates on a map."""
    try:
        if self.current_gps_info:
            lat = self.current_gps_info['Latitude']
            lon = self.current_gps_info['Longitude']
            display_map_in_app(self.root, lat, lon)
    except Exception as e:
        messagebox.showerror(self.get_text("error"), str(e))

def clear_data(self):
    self.image_label.config(image=self.placeholder_image)
    self.image_label.image = self.placeholder_image
    self.clear_exif_data()
    self.detect_button.config(state=tk.DISABLED)
    self.save_button.config(state=tk.DISABLED)
    self.clear_summary()
    self.status_var.set(self.get_text("data_cleared"))
    self.video_controls_frame.pack_forget()  # Hide video controls


def clear_exif_data(self):
    """Clear EXIF data displayed in the UI."""
    self.latitude_entry.config(state=tk.NORMAL)
    self.latitude_entry.delete(0, tk.END)
    self.longitude_entry.config(state=tk.NORMAL)
    self.longitude_entry.delete(0, tk.END)
    self.date_label.config(state=tk.NORMAL)
    self.date_label.delete(0, tk.END)
    self.time_label.config(state=tk.NORMAL)
    self.time_label.delete(0, tk.END)

    self.latitude_entry.insert(0, "No Data")
    self.longitude_entry.insert(0, "No Data")
    self.date_label.insert(0, "No Data")
    self.time_label.insert(0, "No Data")

    self.latitude_entry.config(state='readonly')
    self.longitude_entry.config(state='readonly')
    self.date_label.config(state='readonly')
    self.time_label.config(state='readonly')
```

```
    self.view_map_button.config(state=tk.DISABLED)

def clear_summary(self):
    """Clear the detection summary displayed in the UI."""
    for widget in self.summary_inner_frame.winfo_children():
        widget.destroy()

# GUI.py
def display_exif_data(self, exif_data):
    """Display EXIF data in the UI."""
    self.latitude_entry.config(state=tk.NORMAL)
    self.latitude_entry.delete(0, tk.END)
    self.longitude_entry.config(state=tk.NORMAL)
    self.longitude_entry.delete(0, tk.END)
    self.date_label.config(state=tk.NORMAL)
    self.date_label.delete(0, tk.END)
    self.time_label.config(state=tk.NORMAL)
    self.time_label.delete(0, tk.END)

    date, time = "No Data", "No Data"
    if exif_data['Date and Time']:
        date_time = exif_data['Date and Time'].values
        if date_time:
            date, time = date_time.split()
    self.date_label.insert(0, date)
    self.time_label.insert(0, time)
    self.date_label.config(state='readonly')
    self.time_label.config(state='readonly')

    self.current_gps_info = None
    if exif_data and exif_data["GPS Info"]:
        gps_info = exif_data["GPS Info"]
        self.latitude_entry.insert(0, gps_info['Latitude'])
        self.longitude_entry.insert(0, gps_info['Longitude'])
        self.current_gps_info = gps_info
        self.view_map_button.config(state=tk.NORMAL)
    else:
        self.latitude_entry.insert(0, "No Data")
        self.longitude_entry.insert(0, "No Data")
        self.view_map_button.config(state=tk.DISABLED)
    self.latitude_entry.config(state='readonly')
    self.longitude_entry.config(state='readonly')


def play_video(self):
    """Play the video."""
    play_video(self)

def pause_video(self):
    """Pause the video playback."""
    pause_video(self)

def stop_video(self):
    """Stop the video playback and reset to the first frame."""
    stop_video(self)

def play_next_frame(self):
    """Play the next frame in the video."""
```

```
        play_next_frame(self)

    def display_video_frame(self, frame_idx):
        """Display a specific frame of the video."""
        display_video_frame(self, frame_idx)

    def switch_model(self, model_number):
        """Switch the detection model."""
        if model_number == 1:
            self.current_model = self.model1
            self.status_var.set(self.get_text("model_1_selected"))
        else:
            self.current_model = self.model2
            self.status_var.set(self.get_text("model_2_selected"))

    def set_confidence_threshold(self):
        new_threshold = tk.simpledialog.askfloat(self.get_text("input"),
self.get_text("enter_confidence_threshold"), minvalue=0.0, maxvalue=1.0)
        if new_threshold is not None:
            self.confidence_threshold = new_threshold
            self.status_var.set(f"{self.get_text('confidence_threshold_set')} {new_threshold}")
            self.detect_button.config(state=tk.NORMAL)  # Enable the detect button


if __name__ == "__main__":
    root = tk.Tk()
    app = MangoVisionApp(root)
    root.mainloop()
```

gps_utils.py:

```
import exifread

def extract_gps_and_datetime(image_path):
    with open(image_path, 'rb') as image_file:
        tags = exifread.process_file(image_file)
        gps_info = {}
        gps_latitude = tags.get('GPS GPSLatitude')
        gps_latitude_ref = tags.get('GPS GPSLatitudeRef')
        gps_longitude = tags.get('GPS GPSLongitude')
        gps_longitude_ref = tags.get('GPS GPSLongitudeRef')
        if gps_latitude and gps_latitude_ref and gps_longitude and gps_longitude_ref:
            lat = get_decimal_from_dms(gps_latitude.values, gps_latitude_ref.values)
            lon = get_decimal_from_dms(gps_longitude.values, gps_longitude_ref.values)
            gps_info['Latitude'] = lat
            gps_info['Longitude'] = lon
        date_time = tags.get('EXIF DateTimeOriginal')
        return {"Date and Time": date_time, "GPS Info": gps_info}

def get_decimal_from_dms(dms, ref):
    degrees = dms[0].num / dms[0].den
    minutes = dms[1].num / dms[1].den
    seconds = dms[2].num / dms[2].den
    decimal = degrees + (minutes / 60.0) + (seconds / 3600.0)
    if ref in ['S', 'W']:
        decimal = -decimal
    return decimal
```

icons_buttons.py:

```
from PIL import Image, ImageTk
import tkinter as tk

class Icons:
    def __init__(self, icons_path):
        icon_size = (30, 30)
        self.upload_icon = ImageTk.PhotoImage(Image.open(icons_path + 'cloud-upload.png').resize(icon_size, Image.LANCZOS))
        self.detect_icon = ImageTk.PhotoImage(Image.open(icons_path + 'object.png').resize(icon_size, Image.LANCZOS))
        self.save_icon = ImageTk.PhotoImage(Image.open(icons_path + 'folder-download.png').resize(icon_size, Image.LANCZOS))
        self.map_icon = ImageTk.PhotoImage(Image.open(icons_path + 'map.png').resize(icon_size, Image.LANCZOS))
        self.play_icon = ImageTk.PhotoImage(Image.open(icons_path + 'play.png').resize(icon_size, Image.LANCZOS))
        self.pause_icon = ImageTk.PhotoImage(Image.open(icons_path + 'pause.png').resize(icon_size, Image.LANCZOS))
        self.stop_icon = ImageTk.PhotoImage(Image.open(icons_path + 'stop.png').resize(icon_size, Image.LANCZOS))
        self.clear_icon = ImageTk.PhotoImage(Image.open(icons_path + 'clear.png').resize(icon_size, Image.LANCZOS))

    def create_buttons(self, frame, commands, theme):
        select_button = tk.Button(frame, text="Select Image/Video", image=self.upload_icon, compound="top", command=commands['select'], font=theme.button_font, bg=theme.button_color, fg=theme.button_text_color)
        detect_button = tk.Button(frame, text="Detect Mangoes", image=self.detect_icon, compound="top", command=commands['detect'], font=theme.button_font, bg=theme.button_color, fg=theme.button_text_color, state=tk.DISABLED)
        play_button = tk.Button(frame, text="Play", image=self.play_icon, compound="top", command=commands['play'], font=theme.button_font, bg=theme.button_color, fg=theme.button_text_color, state=tk.DISABLED)
        pause_button = tk.Button(frame, text="Pause", image=self.pause_icon, compound="top", command=commands['pause'], font=theme.button_font, bg=theme.button_color, fg=theme.button_text_color, state=tk.DISABLED)
        stop_button = tk.Button(frame, text="Stop", image=self.stop_icon, compound="top", command=commands['stop'], font=theme.button_font, bg=theme.button_color, fg=theme.button_text_color, state=tk.DISABLED)
        save_button = tk.Button(frame, text="Save Results", image=self.save_icon, compound="top", command=commands['save'], font=theme.button_font, bg=theme.button_color, fg=theme.button_text_color, state=tk.DISABLED)
        clear_button = tk.Button(frame, text="Clear", image=self.clear_icon, compound="top", command=commands['clear'], font=theme.button_font, bg="red", fg=theme.button_text_color)

        return select_button, detect_button,play_button,stop_button,pause_button, save_button, clear_button
```

image_utils.py:

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk, ImageDraw,ImageFont
import numpy as np
import cv2
import os
from gps_utils import extract_gps_and_datetime
```

```
from gps_utils import extract_gps_and_datetime

def process_selected_image(app, file_path):
    app.is_video = False
    app.video_frames = []
    app.video_cap = None
    app.image_path = file_path  # Set image_path to the selected image file path

    try:
        # Load and display the image
        image = Image.open(file_path)
        image.thumbnail((640, 360))
        img = ImageTk.PhotoImage(image)
        app.image_label.config(image=img)
        app.image_label.image = img
        app.status_var.set(app.get_text("image_loaded"))
        app.detect_button.config(state=tk.NORMAL)
        app.save_button.config(state=tk.DISABLED)

        # Extract and display GPS and datetime data
        exif_data = extract_gps_and_datetime(file_path)
        app.display_exif_data(exif_data)

    except Exception as e:
        messagebox.showerror(app.get_text("error"), str(e))
        app.status_var.set(app.get_text("error"))



def display_image(app, image_path):
    image = Image.open(image_path)
    image.thumbnail((640, 360))
    img = ImageTk.PhotoImage(image)
    app.image_label.config(image=img)
    app.image_label.image = img
    app.image_label.image_path = image_path

def annotate_image(app, image_path, valid_detections):
    image = cv2.imread(image_path)
    for box, conf in valid_detections:
        x1, y1, x2, y2 = map(int, box.tolist())
        cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(image, f"{conf:.2f}", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255,
0), 2)
    detected_image = Image.fromarray(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    detected_image.thumbnail((640, 360))
    return ImageTk.PhotoImage(detected_image)

def detect_mangoes_in_image(app):
    if not app.image_path:
        messagebox.showerror(app.get_text("error"), "No image selected.")
        return

    app.status_var.set(app.get_text("detecting_mangoes"))
    try:
        image = Image.open(app.image_path)
```

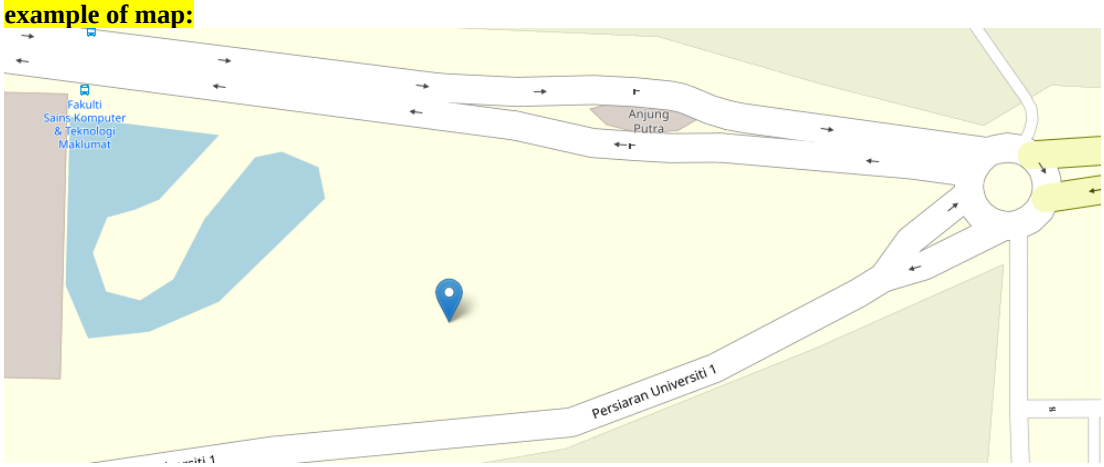
```
        original_image = image.copy()  # Keep a copy of the original image
        # Perform detection on the image using the model
        results = app.current_model.predict(source=app.image_path)
        boxes = results[0].boxes.xyxy
        confidences = results[0].boxes.conf
        valid_detections = [(box, conf) for box, conf in zip(boxes, confidences) if conf >= app.confidence_threshold]
        app.summary = [{'Mango No.': i + 1, 'Confidence Score': f"{conf:.2f}", 'Box Coordinates': box.tolist()} for i, (box, conf) in enumerate(valid_detections)]

        # Annotate the image
        draw = ImageDraw.Draw(image)
        font_path = "arial.ttf"  # Path to a .ttf font file
        font_size = 20  # Adjust font size as needed
        font = ImageFont.truetype(font_path, font_size)
        for i, (box, conf) in enumerate(valid_detections):
            x1, y1, x2, y2 = map(int, box.tolist())
            draw.rectangle([x1, y1, x2, y2], outline="red", width=6)
            label = f"{conf:.2f}"
            draw.text((x1, y1 - font_size), label, fill="red", font=font)

        # Resize the image back to fit the display size (640, 360)
        image.thumbnail((640, 360))
        img = ImageTk.PhotoImage(image)
        app.image_label.config(image=img)
        app.image_label.image = img

        app.display_summary(app.summary)
        app.status_var.set(app.get_text("detection_complete"))
        app.detect_button.config(state=tk.DISABLED)
        app.save_button.config(state=tk.NORMAL)

        app.annotated_image = original_image  # Store the annotated image
    except Exception as e:
        messagebox.showerror(app.get_text("error"), str(e))
        app.status_var.set(app.get_text("error"))


def save_annotated_image(app, save_path):
    try:
        annotated_image = app.annotated_image
        if annotated_image:
            draw = ImageDraw.Draw(annotated_image)
            try:
                font = ImageFont.truetype("arial.ttf", 20)
            except IOError:
                font = ImageFont.load_default()

            for item in app.summary:
                x1, y1, x2, y2 = map(int, item['Box Coordinates'])
                draw.rectangle([x1, y1, x2, y2], outline="red", width=6)
                label = f"{item['Confidence Score']}"
                text_bbox = draw.textbbox((0, 0), label, font=font)
                text_width = text_bbox[2] - text_bbox[0]
                text_height = text_bbox[3] - text_bbox[1]
                draw.text((x1, y1 - text_height), label, fill="red", font=font)
            annotated_image.save(save_path)
```

```
            messagebox.showinfo(app.get_text("save_successful"),      f"{app.get_text('results_saved_to')}
{save_path}")
            app.status_var.set(app.get_text("results_saved"))
        else:
            messagebox.showerror(app.get_text("error"), "No annotated image to save.")
    except Exception as e:
        messagebox.showerror(app.get_text("error"), str(e))
```

map_view.py:

```
import tkinter as tk
import folium
from tkinter import Toplevel, messagebox
from folium import Map, Marker
from io import BytesIO
import webview
import requests

def show_map(lat, lon):
    # Create a map centered around the given coordinates
    folium_map = Map(location=[lat, lon], zoom_start=12)
    Marker(location=[lat, lon], popup='Image Location').add_to(folium_map)

    # Save the map as HTML content
    map_data = BytesIO()
    folium_map.save(map_data, close_file=False)
    return map_data.getvalue().decode()

def check_internet_connection():
    try:
        requests.get("http://www.google.com", timeout=5)
        return True
    except requests.ConnectionError:
        return False

def display_map_in_app(root, lat, lon):
    if check_internet_connection():
        map_html = show_map(lat, lon)

        # Create a Toplevel window for the map
        map_window = Toplevel(root)
        map_window.title("Map View")
        map_window.geometry("800x600")

        # Create a WebView to display the map
        window = webview.create_window("Map View", html=map_html, width=800, height=600,
resizable=True)

        # Start the webview without any additional parameters
        webview.start(gui='tkinter')

        # Destroy the Tkinter Toplevel window after webview closes
        map_window.destroy()
    else:
        if messagebox.askretrycancel("No Internet Connection", "Please connect to the internet and try
again."):
            os.system('start ms-settings:network-status')
```

example of map:



splash_screen.py:

```
import tkinter as tk
from PIL import Image, ImageTk, ImageSequence

def show_splash_screen(root):
    splash = tk.Toplevel()
    splash.overrideredirect(True)

    # Load the GIF file
    gif_path = "F:\\Mango Fruit Detection project\\GUI\\icons\\mango.gif"
    gif = Image.open(gif_path)
    frames = [ImageTk.PhotoImage(img) for img in ImageSequence.Iterator(gif)]

    # Get the dimensions of the GIF
    gif_width, gif_height = gif.size
    screen_width = splash.winfo_screenwidth()
    screen_height = splash.winfo_screenheight()

    # Center the window
    x = (screen_width // 2) - (gif_width // 2)
    y = (screen_height // 2) - (gif_height // 2)
    splash.geometry(f"{gif_width}x{gif_height}+{x}+{y}")

    splash_label = tk.Label(splash, bg="#333333")
    splash_label.pack(expand=True)

    def update_frame(index):
        frame = frames[index]
        index += 1
        if index == len(frames):
            index = 0
        splash_label.config(image=frame)
        splash.after(50, update_frame, index)

    root.withdraw()
    splash.after(0, update_frame, 0)
    splash.after(3000, lambda: root.deiconify())
    splash.after(3000, splash.destroy)

    return splash
```

themes.py:

```
class Theme:
    def __init__(self):
        self.bg_color = "#f0f0f0"  # Background color
        self.button_color = "#4CAF50"
        self.button_text_color = "#ffffff"
        self.frame_bg_color = "#ffffff"
        self.title_font = ("Helvetica", 18, "bold")
        self.subtitle_font = ("Helvetica", 14)
        self.label_font = ("Helvetica", 12)
        self.button_font = ("Helvetica", 10)
        self.entry_font = ("Helvetica", 10)
        self.credit_text_color = "black"

    def apply_theme(self, root):
        root.configure(bg=self.bg_color)
```

toolbar.py:

```
import tkinter as tk
import webbrowser

def create_menu(app):
    app.toolbar = tk.Menu(app.root)
    app.root.config(menu=app.toolbar)

    app.language_menu = tk.Menu(app.toolbar, tearoff=0)
    app.toolbar.add_cascade(label=app.get_text("language"), menu=app.language_menu)
    app.language_menu.add_command(label="English", command=lambda: app.switch_language("en"))
    app.language_menu.add_command(label="Malay", command=lambda: app.switch_language("ms"))

    app.model_menu = tk.Menu(app.toolbar, tearoff=0)
    app.toolbar.add_cascade(label=app.get_text("model"), menu=app.model_menu)
    app.model_menu.add_command(label=app.get_text("model_1"), command=lambda:
app.switch_model(1))
    app.model_menu.add_command(label=app.get_text("model_2"), command=lambda:
app.switch_model(2))

    app.threshold_menu = tk.Menu(app.toolbar, tearoff=0)
    app.toolbar.add_cascade(label=app.get_text("threshold"), menu=app.threshold_menu)
    app.threshold_menu.add_command(label=app.get_text("confidence_threshold"),
command=app.set_confidence_threshold)

    app.help_menu = tk.Menu(app.toolbar, tearoff=0)
    app.toolbar.add_cascade(label=app.get_text("contact"), menu=app.help_menu)
    app.help_menu.add_command(label=app.get_text("contact"), command=lambda:
webbrowser.open("mailto:Marawandeep13@gmail.com"))
```

translations.py:

```
translations = {
    "en": {
        "title": "MangoVision",
        "subtitle": "Mango Fruit Detection From Aerial Images",
        "select_media": "Select Image/Video",
        "detect_mangoes": "Detect Mangoes",
        "save_results": "Save Results",
        "view_on_map": "View on Map",
        "clear": "Clear",
```

```
    "date": "Date:",
    "time": "Time:",
    "latitude": "Latitude:",
    "longitude": "Longitude:",
    "loading": "Please wait, processing...",
    "ready": "Ready",
    "created_by": "Created by: Marawan Eldeib ©2024",
    "model": "Model",
    "model_1": "mode1_CQUniversity Dataset",
    "model_2": "model2_Local Dataset",
    "threshold": "Threshold",
    "confidence_threshold": "Confidence Threshold",
    "image_loaded": "Image loaded successfully.",
    "video_loaded": "Video loaded successfully.",
    "detection_complete": "Detection complete.",
    "data_cleared": "Data cleared.",
    "no_mangoes_detected": "No mangoes detected.",
    "save_successful": "Save Successful",
    "results_saved_to": "Detection results and summary saved to",
    "results_saved": "Results saved successfully.",
    "error": "Error",
    "model_1_selected": "Model 1  selected.",
    "model_2_selected": "Model 2  selected.",
    "input": "Input",
    "enter_confidence_threshold": "Enter new confidence threshold (0.0 - 1.0):",
    "confidence_threshold_set": "Confidence threshold set to",
    "selected_media": "Selected Media",
    "gps_coordinates": "GPS Coordinates",
    "date_time": "Date & Time",
    "detection_summary": "Detection Summary",
    "loading_image": "Loading image...",
    "loading_video": "Loading video...",
    "detecting_mangoes": "Detecting mangoes...",
    "detection_time": "Detection Time: ",
    "play_video": "Play Video",
    "pause_video": "Pause Video",
    "stop_video": "Stop Video",
    "theme": "Theme",
    "language": "Language",
    "contact": "Contact",
    "contact_email": "For further assistance, contact [Marawan Eldeib](mailto:Marawandeep13@gmail.com).",
    "mango_no": "Mango No.",
    "confidence_score": "Confidence Score",
    "total_mangoes_detected": "Total Mangoes Detected"
  },
  "ms": {
    "title": "MangoVision",
    "subtitle": "Pengesanan Buah Mangga Dari Imej Udara",
    "select_media": "Pilih Imej/Video",
    "detect_mangoes": "Kesan Mangga",
    "save_results": "Simpan Keputusan",
    "view_on_map": "Lihat di Peta",
    "clear": "Bersihkan",
    "date": "Tarikh:",
    "time": "Masa:",
    "latitude": "Latitud:",
    "longitude": "Longitud:",
```

```
        "loading": "Sila tunggu, memproses...",
        "ready": "Sedia",
        "created_by": "Dicipta oleh: Marawan Eldeib ©2024",
        "model": "Model",
        "model_1": "Model 1 (20 epochnya)",
        "model_2": "Model 2 (40 epochnya)",
        "threshold": "Ambang",
        "confidence_threshold": "Ambang Keyakinan",
        "image_loaded": "Imej berjaya dimuatkan.",
        "video_loaded": "Video berjaya dimuatkan.",
        "detection_complete": "Pengesanan selesai.",
        "data_cleared": "Data dibersihkan.",
        "no_mangoes_detected": "Tiada mangga dikesan.",
        "save_successful": "Simpan Berjaya",
        "results_saved_to": "Keputusan pengesanan dan ringkasan disimpan ke",
        "results_saved": "Keputusan berjaya disimpan.",
        "error": "Ralat",
        "model_1_selected": "Model 1 (20 epochnya) dipilih.",
        "model_2_selected": "Model 2 (40 epochnya) dipilih.",
        "input": "Input",
        "enter_confidence_threshold": "Masukkan ambang keyakinan baru (0.0 - 1.0):",
        "confidence_threshold_set": "Ambang keyakinan ditetapkan kepada",
        "selected_media": "Media Terpilih",
        "gps_coordinates": "Koordinat GPS",
        "date_time": "Tarikh & Masa",
        "detection_summary": "Ringkasan Pengesanan",
        "loading_image": "Memuatkan imej...",
        "loading_video": "Memuatkan video...",
        "detecting_mangoes": "Mengesan mangga...",
        "detection_time": "Masa Pengesanan: ",
        "play_video": "Mainkan Video",
        "pause_video": "Jeda Video",
        "stop_video": "Hentikan Video",
        "theme": "Tema",
        "language": "Bahasa",
        "contact": "Hubungi",
        "contact_email": "Untuk bantuan lanjut, hubungi [Marawan Eldeib](mailto:Marawandeep13@gmail.com).",
        "mango_no": "Mangga No.",
        "confidence_score": "Skor Keyakinan",
        "total_mangoes_detected": "Jumlah Mangga Dikesan"
    }
}
```

Video_utils.py:

```
import cv2
import tkinter as tk
from tkinter import messagebox, filedialog
from PIL import Image, ImageTk
import os

def select_video_file(app):
    file_path = filedialog.askopenfilename(filetypes=[("Video files", "*.mp4;*.avi")])
    if file_path:
        process_selected_video(app, file_path)

def process_selected_video(app, file_path):
```

```
    app.is_video = True
    app.video_frames = []
    app.annotated_frames = []  # Add this line
    app.video_cap = cv2.VideoCapture(file_path)
    if not app.video_cap.isOpened():
        messagebox.showerror(app.get_text("error"), app.get_text("video_load_error"))
        return

    total_frames = int(app.video_cap.get(cv2.CAP_PROP_FRAME_COUNT))
    app.progress_bar["maximum"] = total_frames

    while True:
        ret, frame = app.video_cap.read()
        if not ret:
            break
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        app.video_frames.append(frame)
        app.progress_bar["value"] += 1
        app.root.update_idletasks()

    app.display_video_frame(0)
    app.status_var.set(app.get_text("video_loaded"))
    app.detect_button.config(state=tk.NORMAL)
    app.save_button.config(state=tk.DISABLED)

    # Show video control buttons
    app.video_controls_frame.pack(fill=tk.X, padx=5, pady=5)
    app.play_button.config(state=tk.NORMAL)
    app.pause_button.config(state=tk.DISABLED)
    app.stop_button.config(state=tk.DISABLED)

def display_video_frame(app, frame_idx):
    if frame_idx < len(app.video_frames):
        frame = app.video_frames[frame_idx]
        frame = Image.fromarray(frame)  # Ensure Image is imported from PIL
        frame.thumbnail((640, 360))
        img = ImageTk.PhotoImage(frame)
        app.image_label.config(image=img)
        app.image_label.image = img

def detect_mangoes_in_video(app):
    app.summary = []
    app.annotated_frames = []  # Add this line
    frame_idx = 0
    app.video_cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    total_frames = int(app.video_cap.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = app.video_cap.get(cv2.CAP_PROP_FPS)
    frame_interval_in_frames = int(fps * app.frame_interval)

    for idx in range(total_frames):
        ret, frame = app.video_cap.read()
        if not ret:
            break
        if idx % frame_interval_in_frames == 0:
            results = app.current_model.predict(source=frame)
            boxes = results[0].boxes.xyxy
            confidences = results[0].boxes.conf
```

```
            valid_detections = [(box, conf) for box, conf in zip(boxes, confidences) if conf >= app.confidence_threshold]
            frame_summary = [{'Mango No.': i + 1, 'Frame': idx, 'Confidence Score': f"{conf:.2f}", 'Box Coordinates': box.tolist()} for i, (box, conf) in enumerate(valid_detections)]
            app.summary.extend(frame_summary)
            for i, (box, conf) in enumerate(valid_detections):
                x1, y1, x2, y2 = map(int, box.tolist())
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)  # Red color
                label = f"{conf:.2f}"
                cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)  # Red color
            app.annotated_frames.append(frame)  # Store the annotated frame
            app.progress_bar["value"] = idx
            app.root.update_idletasks()

        app.display_summary(app.summary)
        app.status_var.set(app.get_text("detection_complete"))
        app.detect_button.config(state=tk.DISABLED)
        app.save_button.config(state=tk.NORMAL)
        play_video(app)  # Play the video with detections in GUI

    def save_annotated_video(app, save_path):
        # Determine the codec based on the file extension
        if save_path.endswith(".mp4"):
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        elif save_path.endswith(".avi"):
            fourcc = cv2.VideoWriter_fourcc(*'XVID')
        else:
            messagebox.showerror(app.get_text("error"), "Unsupported file format")
            return

        out = cv2.VideoWriter(save_path, fourcc, app.video_cap.get(cv2.CAP_PROP_FPS), (app.annotated_frames[0].shape[1], app.annotated_frames[0].shape[0]))
        for frame in app.annotated_frames:  # Save annotated frames
            out.write(cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))
        out.release()
        messagebox.showinfo(app.get_text("save_successful"), f"{app.get_text('results_saved_to')} {save_path}")
        app.status_var.set(app.get_text("results_saved"))


    def play_video(app):
        app.video_playing = True
        if app.video_cap.get(cv2.CAP_PROP_POS_FRAMES) == app.video_cap.get(cv2.CAP_PROP_FRAME_COUNT):
            app.video_cap.set(cv2.CAP_PROP_POS_FRAMES, 0)  # Restart if at the end
        play_next_frame(app)
        app.play_button.config(state=tk.DISABLED)
        app.pause_button.config(state=tk.NORMAL)
        app.stop_button.config(state=tk.NORMAL)

    def pause_video(app):
        app.video_playing = False
        app.play_button.config(state=tk.NORMAL)
        app.pause_button.config(state=tk.DISABLED)

    def stop_video(app):
        app.video_playing = False
```

```
    app.video_cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    app.display_video_frame(0)
    app.play_button.config(state=tk.NORMAL)
    app.pause_button.config(state=tk.DISABLED)
    app.stop_button.config(state=tk.DISABLED)

def play_next_frame(app):
    if app.video_playing and app.video_cap:
        frame_idx = int(app.video_cap.get(cv2.CAP_PROP_POS_FRAMES))
        if frame_idx < len(app.annotated_frames):
            frame = app.annotated_frames[frame_idx]
        else:
            ret, frame = app.video_cap.read()
            if ret:
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                app.annotated_frames.append(frame)
        if frame_idx < len(app.annotated_frames):
            frame = app.annotated_frames[frame_idx]
            frame = Image.fromarray(frame)
            frame.thumbnail((640, 360))
            img = ImageTk.PhotoImage(frame)
            app.image_label.config(image=img)
            app.image_label.image = img
            app.root.after(int(1000 / app.video_cap.get(cv2.CAP_PROP_FPS)), lambda:
play_next_frame(app))
        else:
            app.stop_video()
```