Designing Optimal Object Detection Networks for Detecting Damages to Canola Kernels

by

Angshuman Thakuria

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

in partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE

Department of Biosystems Engineering

University of Manitoba

Winnipeg

THESIS ABSTRACT

Canola is an essential Canadian crop that generated a revenue of $14.4 B from its export in the 2022 fiscal year. To consistently export the best quality canola and set the correct pricing that truly reflects the grade, it is crucial to invest in reliable, high speed, and accurate grading technologies. Leveraging the current progress in Artificial Intelligence algorithms, this research proposes a comprehensive end-to-end system to detect damage to canola kernels and grade them. The proposed system comprises an accelerated sample preparation setup, custom-built specialized AI models, and an edge-AI microprocessor for in-field use. This thesis mainly focuses on the development of specialized neural networks to accurately detect damaged canola seeds as it is the most complex part of the system. Two foundational object detection networks, You Only Look Once (YOLO) version 5 and version 7 were optimized to be compatible with resource-constrained hardware environments. The aim of developing the two optimal networks was to mitigate trade-offs among speed, accuracy, cost, and model size, thereby creating a more balanced and optimized network. Several architectural design options were explored to compress the network structure of the two models in terms of size and cost yet retain or improve the metrics of accuracy and inference speed. The results from the design choices indicate that reconstructing YOLOv5 with ShuffleNet as the backbone reduces its size and cost and increases the inference speed but negatively affects the detection accuracy. Replacing the Convolutional Blocks present in the Spatial Pyramid Pooling Cross Stage Partial and all the Efficient Layer Aggregation Network modules of YOLOv7 with the Ghost Convolutional Network and adding two Convolutional Block Attention Modules improves both mean average precision metric compared to the parent model and reduces the size and cost. To prepare the samples faster, a semi-automatic option was adopted and its effect on dataset quality and the overall time required was also studied. This study is a first step toward developing a commercially viable canola grading system.

ACKNOWLEDGEMENT

I am immensely grateful for the unwavering support and guidance provided by my advisor, Dr. Chyngyz Erkinbaev, throughout my master's studies. His constant encouragement and belief in me have been the cornerstones of my academic journey. My heartfelt appreciation extends to my esteemed committee members, Dr. Danny Mann, and Dr. Arkady Major, for their invaluable contributions. Their thoughtful insights, constructive feedback, and dedication to academic excellence have played a pivotal role in shaping the outcome of this thesis. To my parents, I extend profound thanks for their enduring support and encouragement. Their love and belief in my potential have been a beacon of strength, motivating me to overcome challenges and pursue excellence. The completion of this master's program is a testament to the collective support and encouragement I have received from these exceptional individuals. I am deeply thankful for the mentorship, guidance, and familial support that have enriched my academic journey beyond measure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PUBLICATIONS AND CONFERENCES

Chapters 2 and 3 of the thesis are based on the papers published in the peer-reviewed journal, Smart Agricultural Technology. The papers are reprinted with permission. Chapter 2 is based on Paper 1 and the results are presented in Conference 1 and Conference 2. Chapter 3 is based on Paper 2 and the results are presented in Conference 3.

Paper 1: Thakuria, Angshuman, and Chyngyz Erkinbaev. "Improving the network architecture of YOLOv7 to achieve real-time grading of canola based on kernel health." *Smart Agricultural Technology* 5 (2023): 100300. (https://doi.org/10.1016/j.atech.2023.100300)

Paper 2: Thakuria, Angshuman, and Chyngyz Erkinbaev. "Real-Time Canola Damage Detection: An End-to-End Framework with Semi-Automatic Crusher and Lightweight ShuffleNetV2_YOLOv5s." *Smart Agricultural Technology* (2024): 100399. (https://doi.org/10.1016/j.atech.2024.100399)

Conference 1: Thakuria, Angshuman and Chyngyz Erkinbaev. "A Computer Vision System for Grading Canola Kernels in Real-Time". CSBE 2023, Lethbridge, Alberta, Canada, July 23-26, 2023 (oral).

Conference 2: Thakuria, Angshuman and Chyngyz Erkinbaev. "Damage Detection of Canola Seeds Using a Deep Learning-based Target Detection Algorithm". Food Technology and Research Day at RCFTR, Winnipeg, Canada, November 1, 2023 (poster).

Conference 3: Thakuria, Angshuman and Chyngyz Erkinbaev. "A Network Compaction Strategy to Reduce the Cost of Detection Networks Deployed in Edge-AI Devices: A Case Study for Grading Canola, 15th International Congress on Agricultural Mechanization and Energy in Agriculture – ANKAgEng 2023, Antalya, Türkiye, October 29 – November 1, 2023 (oral).

AUTHOR CONTRIBUTIONS

Chapter 2: The contents of this chapter are reproduced with permission from the self-work of the published article in a peer-reviewed journal.

Thakuria, A., & Erkinbaev, C. (2023). Improving the network architecture of YOLOv7 to achieve real-time grading of canola based on kernel health. Smart Agricultural Technology, 5, 100300. https://doi.org/10.1016/j.atech.2023.100300.

- **Angshuman Thakuria** (Thesis Author)
  Methodology [Lead], Data collection [Lead], Formal analysis [Lead], Software [Lead], Project administration [Lead], Figures & Tables Preparation [Lead], First draft preparation [Lead], Writing - review & editing [Equal].
- **Chyngyz Erkinbaev**
  Conceptualization [Lead], Supervision [Lead], Funding acquisition [Lead], Resources [Lead], Writing - review & editing [Equal].

Chapter 3: The contents of this chapter are reproduced with permission from the self-work of the published article in a peer-reviewed journal.

Thakuria, A., & Erkinbaev, C. (2024). Real-time canola damage detection: An end-to-end framework with semi-automatic crusher and lightweight ShuffleNetV2_YOLOv5s. Smart Agricultural Technology, 7, 100399. https://doi.org/10.1016/j.atech.2024.100399

- **Angshuman Thakuria** (Thesis Author)
  Methodology [Lead], Data collection [Lead], Formal analysis [Lead], Software [Lead], Project administration [Lead], Figures & Tables Preparation [Lead], First draft preparation [Lead], Writing - review & editing [Equal].
- **Chyngyz Erkinbaev**
  Conceptualization [Lead], Supervision [Lead], Funding acquisition [Lead], Resources [Lead], Writing - review & editing [Equal].

CHAPTER 1. Overview of the thesis

1.1. Background

A single canola kernel can be classified as either heated, immature, or sound. The current grading technique is a manual one, where the grader crushes all the seeds to visually examine the color of the cotyledon. This technique is both time-consuming and laborious. Based on the color of the cotyledon, the seed condition is identified. There are two types of damaged seed conditions which affect the grade of the canola, viz., heated, and immature. A perfectly healthy canola kernel has a dark yellow colored cotyledon, while for heat damaged, it is brown, and for immature seeds, the cotyledon is green in color. As the cotyledon is covered with a thick black seed coat, it is necessary to crush them before identifying the damage. The Canadian Grain Commission has set three grades for Canola: No. 1 Canada, No. 2 Canada, and No. 3 Canada. To be classified as Grade 1, the total damage should be less than 5%. The total damage tolerated for the subsequent grades is 12% and 25%, respectively. Apart from the need to set the correct price, it is crucial to correctly set the grade of Canola as a lower grade of canola with a higher damaged seed percentage affects the oil extraction process and increases the cost and the overall energy required for processing.

Thus, grading is a critical first step in the entire canola supply chain. However, this step is often overlooked and therefore there is no major innovation in this field. This step can however be automated by adopting computer vision techniques to specifically identify damaged canola kernels. The utilization of computer vision models will allow the automatic identification of the three classes of seed conditions. Moreover, the adoption of a mechanical canola crusher will reduce the time required to prepare the samples, and the combined system will reduce the problems of mis-grading, labour-intensive operation, and operator fatigue.

In recent years, much of the development in computer vision algorithms is centred toward autonomous driving, surveillance, remote sensing, or healthcare, where companies like Tesla, OpenAI, Microsoft, Meta, and Alphabet are the key innovators. The models that are developed are seldom trained using agricultural datasets. Popular open-source datasets such as ImageNet, which is used to train and benchmark classification models, or the Microsoft COCO, which is used to train and benchmark object detection models have thousands of classes ranging from people to vehicles, hence they are popular for the above-mentioned traditional tasks. However, despite

having thousands of classes, none of them contain any specialized agricultural classes. Hence, most of the studies in agriculture utilize off-the-shelf models developed for other use cases and fine-tuning on a small dataset with outdated or unoptimized models, which hampers the growth and development of production-ready AI systems catering to the agricultural sector.

1.2. Objectives

Thus, the key objective of this research is to develop specialized AI systems using two approaches, Model-centric and Data-centric. The process of detecting damages to canola and grading them is used as a case study. This study is a first step towards building the groundwork for developing an end-to-end canola grading system using advanced computer vision algorithms. This thesis reports the attainment of the following milestones:

1. Automating the process of damaged canola seed identification and grading using various computer vision algorithms.
2. Developing an object detection network based on a combined channel and spatial attention mechanism.
3. Exploring several network compaction strategies to create models with drastically reduced model size, model parameters and computational cost.
4. Evaluating the performance of a semi-automatic machine to prepare the samples in terms of time and resulting dataset quality.
5. Evaluating the performance of two classes of hardware platforms in terms of inferencing at the edge.

1.3. Structure of the thesis

This thesis is a grouped manuscript thesis (sandwich thesis). It comprises of four Chapters, Chapter 1 and Chapter 4 are the introductory and concluding chapters, respectively. Chapter 2 and Chapter 3 are based on two published manuscripts. The contents of each chapter are summarized as follows:

Chapter 1 gives a general introduction to damages present in canola and how they are graded. This chapter lists the objectives of the research and provides the structure of the thesis.

Chapter 2 is based on the manuscript titled “Improving the network architecture of YOLOv7 to achieve real-time grading of canola based on kernel health”, which is published in the journal,

Smart Agricultural Technology. It reports the development of an object detection network (YOLOv7_Ours) designed for the task of grading canola kernels. In this paper, more stress is given to the algorithmic aspect of network development and the mathematics behind the neural networks and the computer vision algorithms. This approach of developing AI systems where more focus is given to the development of a model is called Model-centric AI. The YOLOv7_Ours model comprises four different neural networks and computer vision algorithms, (i) an attention mechanism (Convolutional Block Attention Mechanism), which helps the model to focus only on the relevant features, i.e., the color of the cotyledon, the (ii) substitution of standard convolutional layers with more efficient and less computationally expensive Ghost convolution operations in key modules of the YOLOv7 model, (iii) a multi-object tracking algorithm (ByteTrack) to assign a unique identity number to all the detections, and (iv) a line-cross counting algorithm that counts all the unique detections, once it crosses a predetermined line. The network reported in this chapter focuses more on improving the mean average precision metric and the other metrics take a backstage. The optimal architecture is finalized based on an ablation experiment. Objectives 1, 2, and 3 were achieved in this paper.

Chapter 3 is based on the manuscript titled, "Real-Time Canola Damage Detection: An End-to-End Framework with Semi-Automatic Crusher and Lightweight ShuffleNetV2_YOLOv5s", which is published in the journal, Smart Agricultural Technology. It reports a completely different approach to designing AI systems. Here more focus is given on the dataset used and the effect of different dataset quality and data annotation on model performance. This approach to creating AI systems is called Data-centric AI, which is a rapidly growing route for developing cost-effective AI systems. In this chapter, two methods of dataset creation, created using the traditional approach and semi-automatic machine, were compared in terms of (i) the time required, and (ii) the quality of the dataset quantified by the performance of an object detection network with the same hyperparameters and training conditions. This chapter also describes the reason behind canola damage and why this is a critical issue. Furthermore, this chapter also compares the third component of the automated grading system, i.e., the microprocessor. Two classes of platforms, CPU-based, and GPU-based hardware were evaluated based on their outputted inference speed. The design philosophy that influenced the ShuffleNetV2_YOLOv5s model was to optimize inference speed, model size and computational cost. In terms of network compaction, the ShuffleNetV2_YOLOv5s model has a completely restructured backbone, where

the standard CSPDarkNet53 backbone was replaced with the ShuffleNetV2 convolutional neural network, as the primary feature extractor. This reconstruction allowed the ShuffeNetV2_YOLOv5s model to have a lower model size, lower model parameters, and a lower Floating-Point Operations per Second value. Objectives 1, 3, 4 and 5 were achieved in this paper.

Chapter 4 concludes the thesis by highlighting the major contributions of this research. It also describes the shortcomings and suggests follow-up studies to negate the limitations. The goal of this thesis is to propagate the development of an industrial-grade version of the grading unit with the prospects for replacing the current system as well as to build a unified grading system for different types of seeds. The Python codes used to create the two networks are shared in the appendix section for the follow-up studies. For reproducibility purposes, it is important to mention that all the models were built using the PyTorch framework, and trained on three classes of GPUs, Tesla T4, Tesla V100, and Tesla A100, accessed via Google Colab Pro +.

CHAPTER 2. Improving the network architecture of YOLOv7 to achieve real-time grading of canola based on kernel health

This chapter is based on the manuscript published in the journal "Smart Agricultural Technology" entitled 'Improving the network architecture of YOLOv7 to achieve real-time grading of canola based on kernel health' and is reprinted with permission.

2.1 Abstract

The occurrence of heated and immature canola kernels caused by excessive drying and frost damage is undesired by grain buyers due to low oil yield and diminished market value. The current grading process is visually examining each kernel's endosperm colour and counting the damaged seeds. As this process is time-consuming, the current study proposes an automated grading technique based on multi-object detection, tracking, and counting. The detection task was achieved via an improved YOLOv7 network (YOLOv7_ours) to increase its performance in accurately identifying small objects and decrease its computational cost and size; by adding two convolutional block attention modules and substituting convolutional layers with ghost layers in all the Efficient Layer Aggregation Networks modules, and in the Spatial Pyramid Pooling Cross Stage Partial module present in YOLOv7. The detection weights were fed to the ByteTrack multiple object tracker to track the detections frame by frame in a video feed. The unique identities generated by the tracker for each detected object of interest were then used to count the number of defects using a line cross algorithm. The mean average precision (mAP@0.5) obtained after training the YOLOv7_ours model was 1.02% better, and its cost and size were 32.1% and 37.1% lower than the baseline YOLOv7 model. In a test video, the tracking model achieved a multi-object tracking accuracy of 84.8% and the counting accuracy was determined to be 93.9%. This three-stage model can be readily deployed in an edge device for accurate and real-time grading of canola kernels by grain buyers.

Keywords: YOLOv7; Convolutional Block Attention Mechanism; Multi-Object Tracking; Canola; Quality Control; Automated Grading

2.2. Introduction

Canada's canola industry has experienced impressive growth, culminating in a production of 18.2 million metric tonnes in 2022, which in turn generated a substantial export revenue of 14.4 billion dollars [1]. To effectively support this ongoing expansion and preserve Canada's reputation for producing superior-quality canola, it is crucial that no compromises are made when it comes to the quality of exported seeds. To ensure this, continuous enhancements and modernization of quality control processes are essential. However, the existing technique employed for quality control relies predominantly on manual processes, which poses several challenges. The rapid growth in throughput has amplified the time-consuming nature of these procedures, making them increasingly inefficient and prone to sampling errors. As a result, there is a pressing need to explore and implement more efficient and reliable automated methods for quality control in the canola industry.

Automated quality control based on computer vision technology has gained more acceptance in agricultural and allied fields due to the rapid advancements in vision algorithms that are more accurate and faster than expert human inspectors or graders. Especially, in the recent years, the use of object detection models such as the You Only Look Once (YOLO) series of one-stage detectors [2] are thoroughly explored to replace expert humans in tasks involving simultaneous classification and localization such as detecting broken corn kernels using the YOLOv4-tiny model [3], grading sugar beet vegetables based on the mechanical damages present on them [4], detecting and estimating the load of orange fruits in the orchard under varying light conditions using the YOLOv4 model [5], detecting mold on food using the YOLOv5 model [6], and detecting crop weeds using the YOLOv7 model [7].

However, these studies utilized the models as is, which were not originally developed for agricultural applications. Due to the inherent complexities such as variability, scale, data diversity, and dynamic conditions present in agri-food systems, employing these models without any improvements often poses challenges in performing the tasks at hand with higher accuracy and speed. Therefore, in recent years, several studies have been conducted to improve these models based on the versatile use cases considering all scenarios prevalent in farms and grading floors such as occlusion conditions, small objects, varying lighting conditions, and hardware limitations by adding improvements such as integration of attention mechanisms, the inclusion of additional modules and layers, or modifying the loss function to improve the accuracy, and pruning the

redundant channels and layers of the original network, or replacing certain modules or parts of the network with more efficient models to decrease the size and the computational complexity (cost) of the developed models.

Of all the methods available to improve the accuracy, the addition of attention mechanisms [8] is gaining interest because of the ease of integration in the network architecture of the YOLO model. For instance, to improve the detection accuracy while detecting small targets such as weed plants, Wang et al. (2022) developed the YOLO-CBAM model, a YOLOv5 model integrated with a Convolutional Block Attention Mechanism (CBAM) to detect the invasive weed *Solanum rostratum* Dunal. Similarly, Chu et al. (2023) added the Efficient Channel Attention mechanism to improve the YOLOv5 models' performance in detecting granary pests. Ma et al. (2023) introduced the YOLOv5-lotus model which attained a 0.7% higher mean Average Precision (mAP) than the base YOLOv5 model due to the inclusion of a coordinate attention mechanism. These studies validate that the integration of attention mechanism modules in the network architecture improves the detection results of the base YOLO models. The final goal of developing these models is to deploy them to field applications. But unlike the domains of autonomous driving and robotics, where the advancement in hardware is at par with the advancements in software, the agri-food industry still relies on deploying the models on edge-computing devices such as the NVIDIA Jetson Nano [12]. As a result, there is a constraint on how big and complex these models can be. Several studies have been conducted to counter this issue where bigger models are compacted to fit in these low-powered devices. For instance, Shang et al. (2023) replaced the convolutional layers in the neck region of the YOLOv5s model with the Ghost module and the backbone with ShuffleNetv2. As a result, the size of the original YOLOv5s model was reduced from 13.70 Mb to 0.61 Mb and the inference speed increased from 31.55 frames per second (FPS) to 86.31 FPS. Similarly, Chen et al. (2022) replaced all the Convolutional + Batch Normalization + SiLU (CBS) modules present in the YOLOv7 network with ghost modules to reduce the cost of the network from 105.1 Giga-Floating Point Operations per Second (GFLOPs) to 76.40 GFLOPs.

However, the scope of all these studies was limited to only the detection of the target objects using the improved models, but to be useful for real-world applications, the developed models must also have functionalities such as counting and tracking. Therefore, apart from just detecting the objects of interest, it is also important to find insights from them. Recently, several studies have focussed on gathering insights on agri-food systems by tracking the detections and

counting them. For instance, Shen et al. (2023) detected clusters of grapes using an improved YOLOv5s model, and then tracked the detections using the SORT algorithm and counted the unique targets using a line-cross algorithm. Similarly, Yang et al. (2022) tracked cotton seedlings using the DeepSORT algorithm and counted them with an R^2 of 0.96. To detect and track dairy cows to locate their real-time position, Zheng et al. (2023) developed the YOLO-BYTE model which improved the YOLOv7 model for the detection task and the ByteTrack tracker for the tracking part. Likewise, Zhang et al. (2023) detected fry's using YOLOv5n and improved the SORT algorithm to track and count them dynamically.

The approach of grading canola using computer vision technology introduced in this paper combines all three tasks of detection, tracking, and counting. To the best of our knowledge, no previous studies have been conducted to grade canola utilizing deep learning and computer vision. Therefore, the objective of this study is to grade canola kernels based on the total number of damaged seeds (heated and immature seeds) present in a batch of 500 kernels detected using an improved YOLOv7 network, tracking the detections on a video feed using the ByteTrack multi-object tracker, and counting them together using a region-of-interest line-cross algorithm. Based on the total counts (heated + immature), the grade of canola will be assigned the grade of either No. 1 Canada, No. 2 Canada, or No. 3 Canada, if it contains 5%, 12%, or 25% total damages to them, respectively [19]. This study lays the groundwork for developing a comprehensive end-to-end computer vision system that holds the potential to replace the current standard approach.

2.3. Materials and Methods

2.3.1 Dataset Production and Data Pre-Processing

To prepare the dataset, 50 g of pre-cleaned canola seeds sampled from a larger bulk was crushed using a semi-automatic canola crusher machine. The completion of this process yielded 50 paper strips (dimensions 38.5 (l) × 4.8 (b) cm) each containing 500 crushed canola seeds. A total of 1622 green seeds (6.49% of 25,000) and 485 heated seeds (1.94% of 25,000) were identified in the strips that were spread non-uniformly across the strips. Out of the 50 strips, 6 strips had no damaged seeds and hence strips containing a higher percentage of damaged seeds were transplanted into those to make the distribution even and retain the size of the training examples. A camera (12 MP, F1.6 lens, with OIS) was used to capture the image of the strips under

controlled indoor lighting (that mimicked an industrial grading floor) and constant camera conditions from a height of 30 cm. The fetched RGB images (1024 × 1024 pixels) were then divided into training (80%) and validation sets (20%). The same camera configurations were used to shoot a 16-second-long video (30 FPS, 1080 × 1920) using a completely different strip, which was used to test the counting performance of the model. Fig. 2.1 shows a typical sample of crushed canola images acquired for the study. Fig. 2.2 shows the difference between immature, heated, and sound canola seeds. As the detection task is set as a supervised machine learning problem, the ground truth labels are required to be fed to the model for training, hence LabelImg an open-source image annotation tool was used to manually draw tight rectangular bounding boxes over all the objects of interest falling into the two classes, immature and heated (Fig. 2.3). The annotated image files which contained 50 images of strips with 2107 annotated objects were saved in YOLO format, which included the class and the four coordinates (x, y, w, h) (where, x,y are the coordinates of the top-left point of the bounding box, w,h are the width and height of the bounding box), for further analysis. To increase the robustness of the model when deployed to function in real-world cases, the variation of images used in the training dataset was increased by the image augmentation method, scaling by ±20%, translating by ±20%, flipping with a probability of 0.5, and mosaicking. These augmentations increased the number of training examples by 35% using image processing operations and further improved the model's generalization ability and prevented possible bias present in the dataset which might not be obvious to humans who created the dataset. These augmentation techniques are present by default in the YOLOv7 pipeline, and the values were determined from previous experience.

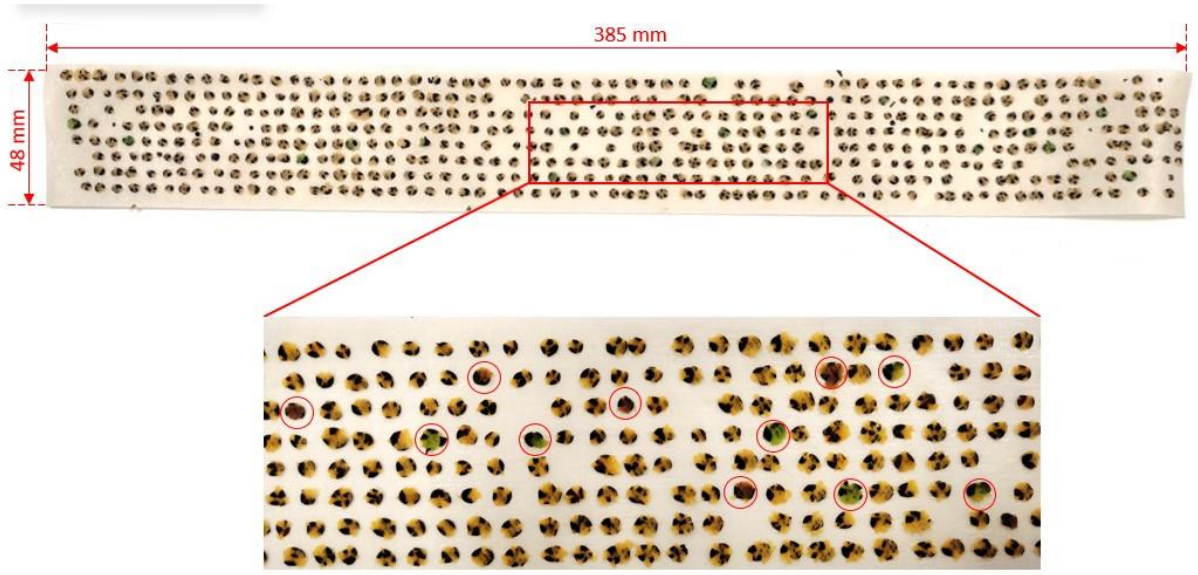


Fig. 2. 1. An example of an image present in the dataset used for training. The enlarged image below shows the presence of immature and heated seeds encircled in red.

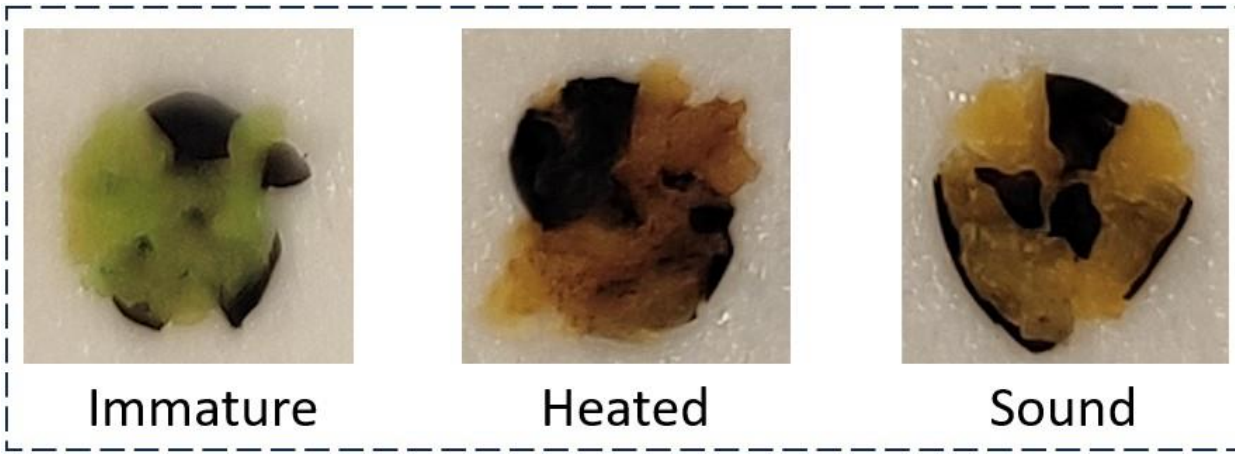


Fig. 2. 2. The color of the cotyledon of immature, heated and sound canola kernels are green, dark brown, and light brown, respectively. Seeds with green and dark brown cotyledon are categorized as damaged seeds.

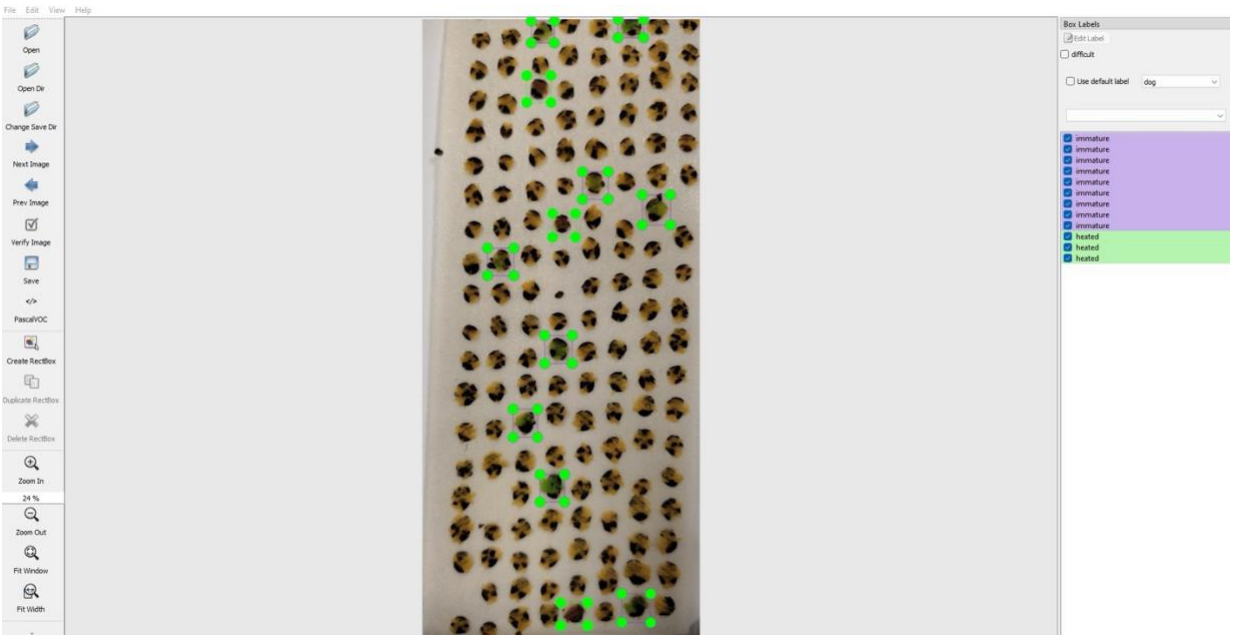


Fig. 2. 3. Data labeling using LabelImg graphical user interface. Tight bounding boxes for the two classes were drawn over the damages present to set the ground truth.

2.3.2. YOLOv7

The YOLOv7 network architecture (Fig. 2.4) features an improved architecture, modules and optimization methods introduced to improve accuracy without increasing the inference cost [20]. In terms of architecture change, the major changes are: (1) Extended efficient layer aggregation networks (E-ELAN) based on the efficient layer aggregation network (ELAN) architecture and (2) Model scaling. The addition of E-ELAN improved the ability of the model to learn more diverse features, improved the use of parameters and calculations, and improved the learning ability of the network without damaging the original gradient path of the ELAN architecture by using expand, shuffle, and merge cardinality [20]. With regards to model scaling, the compound scaling method introduced with YOLOv7 is used to develop larger models by maintaining the properties of the basic model on which the scaled model is based such as the original design and structure. Like the previous YOLO models, there are four parts to the YOLOv7 model - input, backbone, neck, and head, where each part in the YOLOv7 network has different functions (Fig. 2.4).

Images at the fixed resolution of 640 × 640 pixels were inputted in batches of 8 into the backbone of the YOLOv7 network as the YOLOv7 model performs better at this input resolution compared to 320 × 320 pixels and 512 × 512 pixels (Table 2.2). The backbone region of the model is responsible for extracting features and creating a feature map and is made up of four Convolutional + Batch Normalization + Sigmoid Linear Activation function (CBS) modules ($CBS_{1\text{-}4}$), four ELAN modules ($ELAN_{1\text{-}4}$), and three Maxpooling (MP1) modules ($MP1_{1\text{-}3}$). The CBS modules (Fig. 2.4a) which are the building blocks of ELAN, ELAN_W, MP, and SPPCSPC modules present in the network architecture are built using a standard convolution layer with one kernel, "same" padding, and a stride of 1, along with 2D batch normalization and SiLU activation function. The first four CBS modules in the backbone are connected in series and the output tensor from the fourth and final CBS module (CBS_4) is then fed to the first of the four ELAN ($ELAN_1$) modules present in the backbone. The ELAN module (Fig. 2.4b), the primary feature extractor used in YOLOv7, comprises seven CBS modules and an output concatenation unit. In the ELAN module, the first two CBS module, CBS_1_{ELAN} and CBS_2_{ELAN} gets their input tensor from the previous layer. The output tensor from the CBS_1_{ELAN} module is then fed to four successive CBS modules (CBS_3to6_{ELAN}) connected in series, and the output from CBS_6_{ELAN} along with the output of CBS_1_{ELAN}, CBS_2_{ELAN}, CBS_4_{ELAN} is then concatenated and fed to CBS_7_{ELAN}. The MP1 (Fig. 2.4d) module is made up of three CBS modules and a Max Pooling unit. Both the max pooling unit and the first CBS module (CBS_1_{MP1}) receive input from CBS_7_{ELAN}. The output of the max-pooling unit is fed to CBS_2_{MP1}, the output of CBS_1_{MP1} is fed to CBS_3_{MP1} and the output from $CBS\_1,3_{MP1}$ is concatenated. In the backbone, $MP1_1$, $MP1_2$ and $MP1_3$ are connected after $ELAN_1$, $ELAN_2$ and $ELAN_3$, respectively. The neck connects the backbone with the head, where it concatenates the feature maps extracted by the backbone to input it into the head. The function of the head is to output the final detections along with the bounding boxes. In YOLOv7, the head consists of one Spatial Pyramid Pooling Cross Stage Partial (SPPCSPC) module, four ELAN_W modules ($ELAN\_W_{1\text{-}4}$), four concatenation units, four standalone CBS ($CBS_{1\text{-}4}$) modules, two MP2 modules ($MP2_{1\text{-}2}$) and three Represented Convolution (RepConv) modules ($RepConv_{1\text{-}3}$). The SPPCSPC module (Fig. 2.4e) is the first module in the head region, and it receives the input tensor from $ELAN_4$ present in the backbone. The SPPCSPC module is a Cross Stage Partial Network with a Spatial Pyramid Pooling (SPP) block instead of a Dense block as in YOLOv5, whose function is to obtain multiscale region features using the max pool and

concatenation operations and is present in the head to solve the issue of duplicate gradient information, thereby decreasing the computation cost of the YOLOv7 network, along with improving the detection speed and accuracy. The ELAN_W (Fig. 2.4c) module present in the head differs from the ELAN module present in the backbone as the output from all the six CBS modules is concatenated compared to only four in ELAN. Two out of the four standalone CBS modules have shortcut connections from $ELAN_2$ and $ELAN_3$. These shortcut connections enable the network to reuse the features from the shallower regions of the network to the deeper parts of the network. The architecture of the two MP2 modules present are the same as the MP1 modules, except the concatenation block in $MP2_1$ gets an additional input from $ELAN\_W_1$ and the concatenation block in $MP2_2$ gets an additional input from the SPPCSPC module. The output tensors of $ELAN\_W_2$, $ELAN\_W_3$, and $ELAN\_W_4$ are fed to $RepConv_1$, $RepConv_2$, and $RepConv_3$, respectively, for final detections. The final detection tensor includes the predicted class probability, the objectness score, and the bounding boxes for all the target objects in the input image [20].

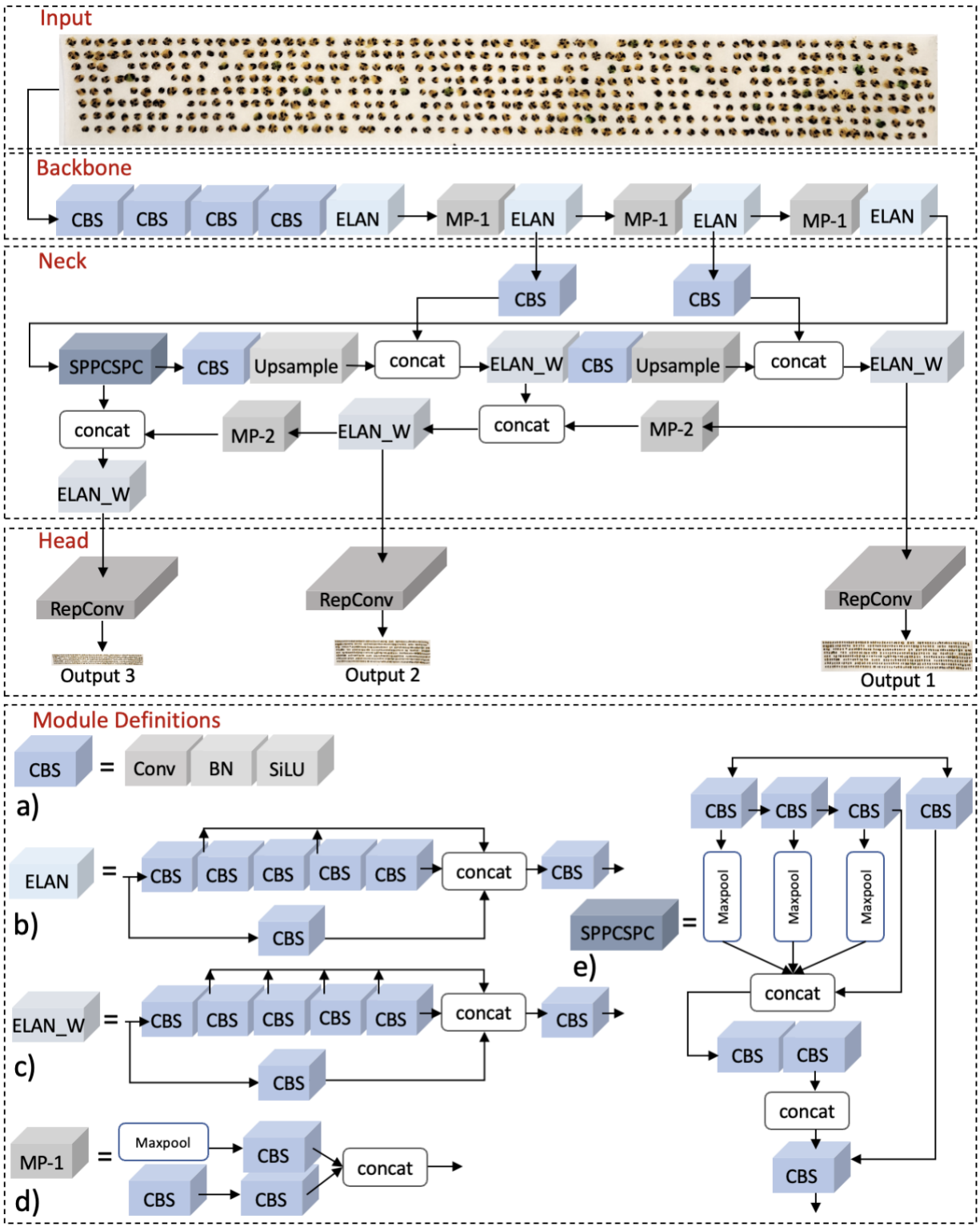


Fig. 2. 4. Architecture of the base YOLOv7 model. (a) Structure of Convolution + Batch Normalization (BN) + Sigmoid Linear Unit (SiLU) (CBS) module, (b) Structure of Efficient

Layer Aggregation Network (ELAN) module, (c) Structure of ELAN_W module, (d) Structure of MP module, and (e) Structure of Spatial Pyramid Pooling Cross Stage Partial (SPPCSPC) module.

2.3.3. YOLOv7_ours: An Enhanced YOLOv7 network for detecting Damaged Canola Kernels

Just like an expert human grader, it is hypothesized that the network will also focus on the cotyledon region to distinguish between healthy and damaged canola kernels. But as shown in Fig. 2.3 the ground-truth bounding box includes the seed coat, which is an irrelevant feature as it has the same appearance (dark-brown colour) irrespective of the health condition of the kernel and therefore it doesn't account for the classification process. However, drawing a smaller bounding box that only includes the pixels corresponding to the cotyledon region is not a viable option as the rectangular bounding box cannot fit the entire cotyledon due to its irregular shape and doing so will lead to a loss of valuable information and further decrease the size of the already small targets leading to a significant rise in the complexity of the task. Thus, to selectively focus on the relevant feature necessary for classification, i.e., the cotyledon colour and suppress the unimportant ones, i.e., the seed coat pixels, an attention mechanism - Convolutional Block Attention Module is integrated into the network architecture of the base YOLOv7 model. The attention mechanism, which found its inspiration in human vision can filter out irrelevant information while processing information to make decisions. This integration will result in improved accuracy and inference speed, albeit at the cost of increased parameters and FLOPS, which is countered by replacing standard convolutional layers in the CBS module with more efficient "Ghost convolutional layers". The addition of Ghost convolutional layers changed the standard *conv layers* with ghost *conv layers* but retained the batch normalization and SiLU activation function. This change from CBS to GBS converted the other modules into their Ghost counterparts - GhostELAN, GhostELAN_W, GhostMP, and GhostSPPCSPC as GBS became the building block of these modules. Thus, the two improvements made to the network architecture as shown in Fig. 2.5 are *(1) the Addition of a CBAM attention mechanism*, and *(2) the Substitution of*

standard modules with lighter-weight Ghost modules. The construction of these two modules is discussed in more detail in section 2.3.3.1. and section 2.3.3.2.

Once the two improvements are finalized, the next step is to determine the number of modules needed and their placement within the network architecture. To address these questions, two comparative studies were conducted to determine the location for the CBAM module and the modules that should be converted to their ghost counterpart. Regarding the first improvement, determining the optimal location for integrating the CBAM module depends on several factors, such as the dataset, target-object type, and network architecture complexity. A common approach is to employ the attention mechanism after the convolutional layers in the backbone or before other modules, enabling the network to selectively pass down the most relevant features. To identify the best location, we added the CBAM module in several positions within the network at varying quantities and compared the resulting performance against that of the original YOLOv7 architecture. Based on the results of this comparative study (Table 2.3), the attention module was placed after CBS_1 and CBS_2 as shown in Fig. 2.5. This position allowed the deeper network to focus on and pass down only the most relevant features. With regards to the second improvement, it was found that replacing all the standard modules with their ghost counterpart slightly degrades the performance due to the substituted ghost convolution layers discarding some information by using cheap filters on a subset of the input feature map. Thus, a comparative experiment (Table 2.4) was conducted to find the modules whose substitution will not affect the performance, and based on it GBS modules were substituted with standard CBS in the SPPCSPC and all the ELAN and ELAN_W modules (Fig. 2.5).

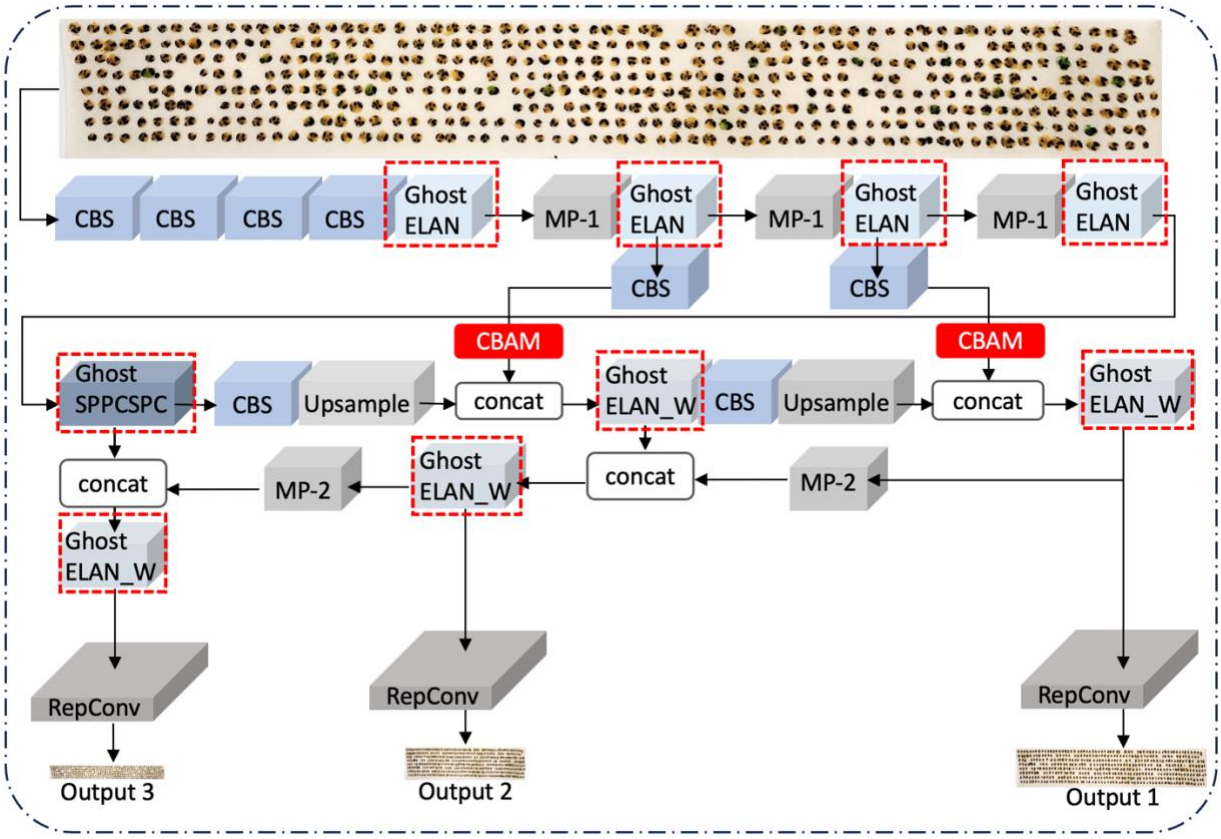


Fig. 2. 5. Structure of the improved YOLOv7 network. The position of the two Convolutional Block Attention Modules was finalized based on a comparative experiment with five other options (Table 2.3). The modules highlighted in red are substituted with ghost modules (Table 2.4).

2.3.3.1. Construction of Convolutional Block Attention Module (CBAM)

CBAM (Fig. 2.6a) is a type of attention mechanism developed by [21]. Attention mechanisms are used in computer vision networks to selectively focus on the most relevant and significant features and suppress the unimportant ones, thus improving the overall performance of the original network [8]. Unlike other attention mechanisms that consider either the channel or spatial axis, CBAM uses both, which allows it to focus on 'what' the relevant information is and 'where' that relevant information is in the image [21].

Given an input feature map $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$, where C is the number of channels, and H and W are the height and width of the input data, respectively, the refined output feature map **F''** is

computed by inferring a one-dimensional channel attention map $\mathbf{M_c} \in \mathbb{R}^{C \times 1 \times 1}$, and a two-dimensional spatial attention map $\mathbf{M_s} \in \mathbb{R}^{1 \times H \times W}$. To do so, it uses a channel attention module and a spatial attention module arranged sequentially, the former before the latter [21]. The channel attention module (Fig. 2.6b) captures the channel-wise dependencies in the feature maps, i.e., it decides on 'what' to focus given an image. To compute the channel-attention map, two intermediate feature maps, F^c_{avg} and F^c_{max} are obtained from the input feature F by average-pooling and max-pooling operations. These two maps are then fed to a common multi-layer perceptron (MLP) network with one hidden layer. The resulting output vectors are then element-wise summed and normalized with a sigmoid function. Mathematically it can be written as:

$$M_c(F) = \sigma\,(MLP(AvgPool(F)) + MLP(MaxPool(F))) \quad (1)$$

Where σ represents the sigmoid function.

The obtained channel attention map is then element-wise multiplied (denoted by ⊗) with the input feature map $\boldsymbol{F}$ to generate a channel-refined feature map $\boldsymbol{F'}$ (Eq. 2)

$$F' = M_c(F) \otimes F \quad (2)$$

The Spatial Attention Module (Fig. 2.6c) captures the spatial dependencies, i.e., 'where' to focus given an image [21]. To generate the 2D spatial attention map, first, the channel-refined feature map $\boldsymbol{F'}$ is pooled using max-pooling and average-pooling along its channel axis. The two resultant 2D pooled maps, $F^s_{avg} \in \mathbb{R}^{1 \times H \times W}$ and $F^s_{max} \in \mathbb{R}^{1 \times H \times W}$ are concatenated and forwarded to a convolution layer with a kernel size of 7 ($f^{7\times7}$). The resultant map $M_s \in \mathbb{R}^{H \times W}$ is then normalized using a sigmoid function. Mathematically, this operation can be summarized as:

$$M_s(F') = \sigma\,(f^{7\times7}([AvgPool(F'); MaxPool(F')])) \quad (3)$$

Finally, using this spatial attention map, the output of CBAM, i.e., the refined feature map F" is computed using Eq. 4:

$$F'' = M_s(F') \otimes F' \quad (4)$$

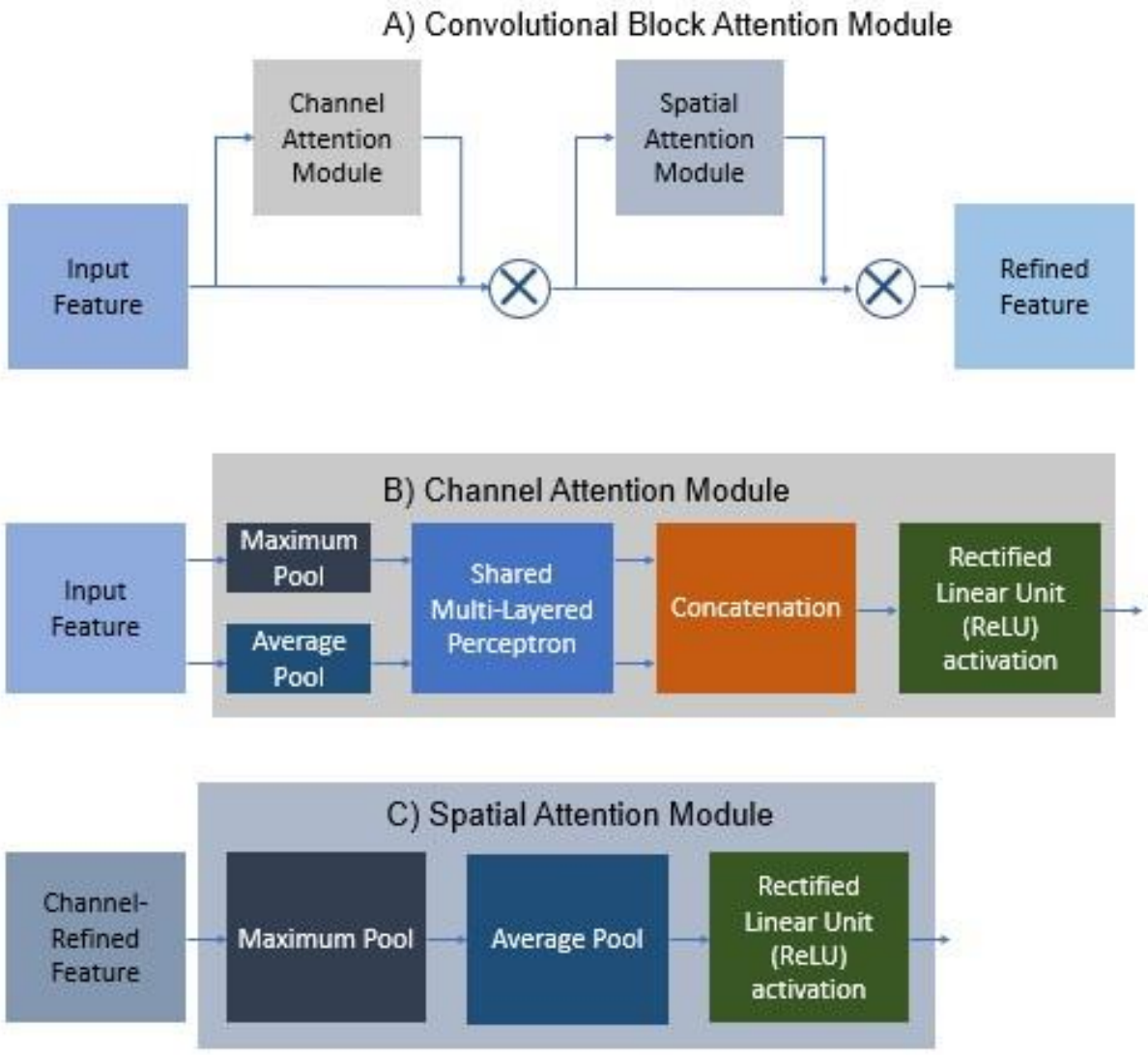


Fig. 2. 6. Structure of (a) Convolutional Block Attention Mechanism. (b) Channel Attention Module. (c) Spatial Attention Module.

2.3.3.2. Construction of Ghost Modules

It is known that the convolution operation between filters and the input data can output pairs of feature maps that contain almost similar information, and these outputted feature maps can contain similar information. Generating these feature maps one by one using a large number of filters costs a lot in terms of FLOPS, but they are crucial as they play a significant role in increasing the accuracy of the network. To decrease the number of filters required to generate those redundant feature maps, Ghost modules [22] use the ghost convolution operation Fig. 2.7. The ghost

convolution works as follows: Instead of using ‘n’ filters to generate ‘n’ feature maps, a fewer number of filters ‘m’ (where m ≤ n) is used to generate ‘m’ feature maps. These intrinsic feature maps are then used to generate ‘s’ feature maps using a cheap linear operation ϕ, to output n = m.s feature maps (Eq. 5). These ‘s’ feature maps can be called the ‘ghosts’ of the intrinsic feature maps as they contain almost the same information but generated without using the same number of filters, thereby reducing the overall cost (in terms of GFLOPs) of the network [22].

$$y_{ij} = \phi_{i,j}(y'_i), for\ i = 1, \dots, m, j = 1, \dots, s \quad (5)$$

Where, y_{ij} represents the feature map generated after the ghost operation ϕ.

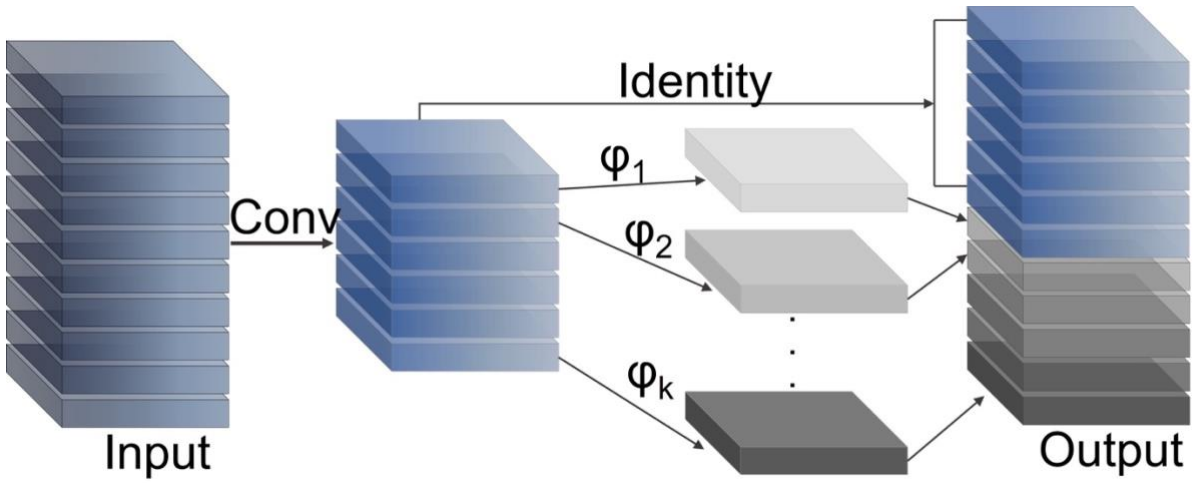


Fig. 2. 7. Schematic diagram of ghost convolution. A few intrinsic feature maps are generated using standard convolution after which linear transformations are applied to them to generate the rest of the ‘ghost’ feature maps.

2.3.4. Detection Model Training

To train the detection network, the annotated damaged canola kernel training set was used to fine-tune the two YOLOv7 models via transfer learning using the pretrained YOLOv7_training.pt weights in a server using an NVIDIA Tesla V100 Tensor Core GPU. All the models were constructed by the deep learning framework PyTorch and were pretrained on the Microsoft Common Objects in Context (MS COCO) dataset which has 328,000 images divided into 80 classes. The hyperparameters for both the base and improved model were kept constant

and are given in Table 2.1. After the YOLOv7 model was trained with the damaged canola kernel dataset, its result was used as a baseline to further improve the original network's capability.

2.3.5. Counting the Damages by a Tracking-by-Detection Paradigm

Counting by solely relying on a detection model is unreliable for our use case as there is a high probability the model might register multiple counts of the same damaged kernel when it appears in successive frames. As the application of grading based on total damages present is critical of the number of counts of damaged seeds and has a stringent threshold to determine the market value of the seeds, duplicate counts will result in a higher number of false positive cases thereby decreasing the potency and reliability of the model for commercial use. Therefore, the counting approach should be based on an algorithm that is not solely based on detections. A reliable technique is thus by tracking each damage throughout its lifetime in the video using an object tracking algorithm until the counter is updated. Multi-object trackers (MOT) which work on the tracking-by-detection paradigm track multiple objects-of-interest by detecting the objects in each time frame 't', associating these detections with existing objects in the previous time frame, 't - 1', and then predicting the locations of these objects in the next time frame, 't+1', and is repeated for every frame in the video sequence to track the objects over time. Foundational MOT algorithms like SORT [23] use a Kalman filter [24] for predicting and updating the state of the object and the Hungarian algorithm [25] for associating the objects. The output of the MOT are bounding boxes with a unique ID generated for each object which helps identify the objects in cases when the detections are lost due to flickering, motion blur, or occlusions. However, these models can be prone to ID switching, hence, to evaluate the accuracy of the MOT, the multi-object tracking accuracy (MOTA) metric is used (Eq. 6).

$$MOTA = 1 - \frac{\sum_t FN_t + FP_t + IDS_t}{\sum_t GT_t} \quad (6)$$

Where, FN, FP, IDS, and GT represent false negatives, false positives, ID switches, and ground truth counts, respectively.

Table 2. 1. The specifics of the hyperparameters that were used for training the YOLOv7 and the YOLOv7_ours model.

Hyperparameters	**Values**

Learning rate	0.01
Momentum	0.937
Batch Size	8
Epochs	150
Optimizer	SGD
IoU training threshold	0.20
Anchors per output layer	3

2.3.5.1. Construction of ByteTrack

ByteTrack is a state-of-the-art real-time MOT algorithm with a MOTA of 80.3 in the MOT17 test set [26] and was therefore used as a tracker to generate unique identities of the defects present in a test video. It proposes a novel data association metric, BYTE, which utilizes every detection box, unlike other MOT algorithms which only keep the detection boxes with confidence scores above a threshold [26]. ByteTrack is constructed as follows:

(1) The detection boxes outputted by the YOLOv7_ours model having a confidence score 'τ' ≥ 0.6 is stored into D_{high} and boxes with $\tau < 0.6$ are stored into D_{low}.

(2) A Kalman Filter is then used to estimate the new location in the present frame of each track T. This was done recursively in two steps: prediction using the priori estimates and a motion model, and update step. A constant linear motion model was utilized for the study as the defects travelled in a straight line and an eight-dimensional state vector $x = (h, v, a, c, h', v', a', c')$ was used, where, (h, v, a, c) represents the horizontal and vertical coordinate, aspect ratio, and height of the centre of the detection box, respectively, and it's derivatives are represented by (h', v', a', c'). The prediction was made using Eq. 7.

$$\begin{cases} \hat{x}_t = S\hat{x}_{t-1} \\ E_t = SE_{t-1}S^T + N_{t-1} \end{cases} \quad (7)$$

where $\hat{x}_t$ is the estimated state of the damaged seed at the current frame t, S is the state transition matrix, $\hat{x}_{t-1}$ is the state of the seeds at the previous frame t-1, E_t is the estimated

error covariance matrix at the current frame t, E_{t-1} is the error covariance matrix at the previous frame t-1, and N_{t-1} is the noise covariance matrix at frame t-1. In the next step, the movement state of the damaged seeds is updated using Eq. 8.

$$\begin{cases} G_{t+1} = E_t O^T (O E_t O^T + M)^{-1} \\ l_{t+1} = C_{t+1} - O\hat{x}_t \\ \hat{x}_{t+1} = \hat{x}_t + G_{t+1} l_{t+1} \\ E_{t+1} = (I - G_{t+1} O) E_t \end{cases} \quad (8)$$

Where G_{t+1} is the Kalman gain, M, and O are the observation noise and the observation matrix, respectively, l_{t+1} is the residual between the observed position of the damaged seed and its estimated position at the next frame t + 1, C_{t+1} is the new observed position of the seeds at the next frame, I represent identity matrix, and $\hat{x}_{t+1}$ is the estimated update of the state of the damaged canola seeds at the next frame.

(3) Data association is performed between D_{high} and all track T, followed by the Hungarian algorithm to complete the matchings. Tracks and detections which were unmatched in this step are stored in T_{remain} and D_{remain}, respectively. Following the first association, A second association is performed between low confidence detections stored in D_{low} and the remaining tracks T_{remain}.

(4) All the unmatched tracks are stored in $T_{unmatched}$ and the unmatched detection boxes with $\tau < 0.2$ are deleted. The unmatched tracks that remain unmatched after the second association are kept in T_{lost}. New tracks are then started from D_{remain} post first association. This results in an output containing the bounding box along with the identity of the tracks T in the frame t.

2.3.5.2. Line-cross counting algorithm for counting the targets

Once the target objects present in each of the four test videos were detected and tracked, the counting functionality was activated via a line-cross algorithm. The algorithm was developed in such a way that it can count the objects irrespective of the direction of the objects in the video. This was done to provide flexibility with the camera movement in an industrial setting and to validate the results of counting in one direction with the results of counting in another direction.

The two target classes were counted together as the process to determine the grade as either No. 1 Canada, No. 2 Canada, or No. 3 Canada is based on the total damages present [19].

The counter works as follows:

(1) Once a target object is detected within the video frame, the coordinates of its bounding box are recorded for each subsequent frame until it crosses a predetermined line.
(2) A line is drawn in the video feed at coordinates (0, 1750) and (1080, 1750).
(3) Two counters are initialized (c_out) and (c_in) based on the direction of the moving targets.
(4) The counter (c_out) is updated if the coordinate of a tracked object is greater than the coordinate of the set line.
(5) The counter (c_in) is updated if the coordinate of a tracked object is lesser than the coordinate of the set line.
(6) The two counters are activated until the video ends, or all the target objects cross the set line.

2.3.6. Evaluation of Detection and Counting Performance

Precision (P), Recall (R), F1 Score, Mean Average Precision (mAP), and Frames per second (FPS) are the metrics that were used to evaluate the detection performance of the models. The P value was calculated using Eq. 9. The R value describes the sensitivity of the object detection model and was calculated using Eq. 10. The F1 Score is the harmonic mean between the precision and the recall value (Eq. 11). The mAP value represents the accuracy of the object detection model and is computed by calculating the area under the precision-recall curve (Eq. 12). The mAP value at 0.5 IoU (Intersection over union) was used to evaluate the performance of our model. Mean detection time includes the time taken by the network to detect an object and it includes both the time taken for non-maximal suppression and for making the inference and FPS is calculated by taking its inverse (Eq. 13).

$$Precision\ (P) = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{9}$$

$$Recall\ (R) = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{10}$$

$$F1_{Score} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{11}$$

$$mAP = \frac{1}{2}\sum_{m=0}^{1}\int_{0}^{1} P_m(R_m)dR_m \quad (12)$$

$$FPS = \frac{1}{mDT} = \frac{1}{\frac{1}{M}\sum_{i=1}^{M} t_i} \quad (13)$$

To evaluate the accuracy of counting (P_c) the damages in the video, Eq. 14 was used.

$$P_c = \left(1 - \frac{|N_g - N_a|}{N_g}\right) \times 100\% \quad (14)$$

Where N_a represents the estimated count, N_g represents the actual count. The ground truth was determined by a trained person by manually counting the seeds.

2.4. Results and Discussion

2.4.1. Baseline model performance with varying input image resolution

Table 2.2 outlines the performance of the baseline model at three different input image resolutions. Three inputs with a fixed image resolution of 320 × 320, 512 × 512, and 640 × 640, were fed to the network in batches of 8 to determine the best input image resolution at which the model will show the most optimal performance in terms of mean average precision (mAP@0.5 IoU). It is observed that the model gets better at accurately detecting the target object when the input resolution is higher, therefore as the input resolution increased, so did the mAP. This trend is in accordance with the results reported by Parico and Ahamed, (2021), where their YOLOv4 model trained to detect pear fruits with high-resolution images showed better mAP than the models trained with low-resolution images. This is because when the image resolution is higher, the clarity of the target objects in the image is enhanced and the model can learn its inherent features better compared to when the target objects are pixelated in low-resolution input images. However, as the model had to learn more features associated with the ground truth with higher resolution images, the GPU memory consumption also increased and in turn, it took more time to train. The time required to train the three different models for 150 epochs ranged from 0.28 hours to 0.52 hours and the mAP varied from 0.96 to 0.97, respectively. The model size was almost the same, as they were essentially identical except for the variation in input image resolution. Although the difference in mAP between YOLOv7-320px and YOLOv7-640px is less than 1%, the

improvements carried out henceforth were based on the 640-pixel input resolution as any improvement is significant.

2.4.2. Model Improvement Results and Ablation Analysis

2.4.2.1. Impact of CBAM Integration on Network Performance

The Convolutional Block Attention Module was added in the network architecture of the YOLOv7 model so that it can selectively learn the most important features, i.e., the color of the cotyledon and ignore the irrelevant features present in the ground truth boxes. To determine the optimal placement and number of these module(s) to be integrated into the network, five potential options were carefully evaluated and identified, each aimed at identifying the most effective configuration for the successful incorporation of these modules. The goal was to find the most optimal option which can result in the highest performance metrics in terms of F1 Score and mAP and the lowest computational cost and complexity in terms of model size and GFLOPs. The result of this comparative experiment with the five shortlisted options is shown in Table 2.3. In these shortlisted options, all the CBAM modules were integrated into the backbone of the YOLOv7 network, and the number of these units ranged from one to four. The first option involved the insertion of four CBAM units, three located before each of the three MP1 modules, and one before the SPPCSPC module. The second option proposed the placement of a single unit before the SPPCSPC module. The third option proposed the placement of two units before the two standalone CBS modules. The fourth option involved placing two units after the standalone CBS modules (Fig. 2.5), while the fifth option was similar to the fourth option, but with adding an extra unit before the SPPCSPC module. Through comparative analysis of these options, it was found that the model size and the cost had no relation with the number of CBAM modules, and the addition of more units didn't necessarily improve the performance of the YOLOv7 model. For instance, option 2 with a single CBAM module was 5% larger in size than option 4 with two CBAM units, but its F1 score was 1.7% lower. Also, option 1 with four CBAM units had 1.12% lower mAP than option 2 which had a single unit. However, as expected, the integration of all these additional modules did increase the computational cost of the model from 105.1 GFLOPs to 105.4 – 112.9 GFLOPs. As outlined in Table 2.3, results indicate that YOLOv7_CBAM4 emerged as the best YOLOv7 + CBAM model as when compared to the baseline YOLOv7 model, there was a 1.12% improvement

in mAP and only 0.28% and 0.4% increment in the cost and model size, respectively. This option also resulted in the highest mAP, and lowest model size and cost compared to other options.

Table 2. 2. Performance metrics, time required to train, and size of the baseline YOLOv7 models at different input image resolutions.

Model Name	P (%)	R (%)	F1 Score (%)	mAP@0.5 (%)	Training Time (h)	Size (Mb)
YOLOv7-320px	96.4	96.3	96.3	96.8	0.287	74.7
YOLOv7-512px	96.6	96.3	96.4	97.2	0.425	74.7
YOLOv7-640px	96.7	96.5	96.6	97.6	0.524	74.8

2.4.2.2. Results of Replacing Standard Convolution with Ghost Convolution

However, as seen in Table 2.3, the computational cost rose noticeably due to the addition of the CBAM modules, hence a network compaction strategy was devised. As a part of this strategy, Ghost modules were included in the network to make the baseline YOLOv7 model and the YOLOv7_CBAM4 model more compact while retaining or further improving the original model's performance metrics in terms of mAP, F1 Score, Size, and cost (GFLOPs). The performance of the YOLOv7 model, when different modules in it are substituted with their 'ghost' counterparts, is tabulated in Table 2.4. It was observed that substituting the convolutional layers in only the SPPCSPC module, the ELAN modules, and the ELAN_W modules improved the mAP by 0.7%, and drastically reduced the cost and model size by 32.1% and 37.1%, respectively in comparison to the baseline YOLOv7 model. Furthermore, substituting all the conv layers present in the original modules with this module decreases the performance of the model, for instance, compared to the baseline model the precision, recall, and inference speed dropped by 1.03%, 1.14%, and 18.55%, respectively. Therefore, even though the option of replacing all modules with Ghost conv layers resulted in the lowest size and cost amongst the other options, only ELAN, ELAN_W, and SPPCSPC modules were changed as this was the best model in terms of mAP@0.5 and the second-best model in terms of size and cost.

2.4.2.3. Combined Effect of All Model Improvements on Performance

These two improvements – YOLOv7_CBAM4 and YOLOv7_ELAN + ELAN_W + SPPCSPC (sections 2.4.2.1 and 2.4.2.2) were combined to construct the enhanced YOLOv7_ours model. Table 2.5 outlines the ablation experiment conducted to show the effect of each improvement in enhancing the performance of the baseline model. These two improvement strategies increased the mAP and F1 Score by 1.02% and 0.31%, respectively, while decreasing the model size and GFLOPs by 37.08%, and 32.08%, respectively, compared to the off-the-shelf YOLOv7 model. However, the inference speed of the improved model decreased by 5 FPS, but as it is still higher than the real-time requirement of 24 FPS, it is a justified trade-off due to the improvements obtained in all the other metrics. However, the results in Table 2.5 also reveal that among the four models, the YOLOv7_ours model only leads in the precision metric and ranks second in other metrics such as mAP, size, and cost. Due to the addition of two CBAM modules that passed down refined feature maps comprising of relevant information, the YOLOv7_CBAM4 model, as hypothesized, attained the highest mAP, and due to the network compaction strategy implemented to decrease the number of filters required to generate feature maps, the YOLOv7_ELAN + ELAN_W + SPPCSPSC model attained the lowest model size and the lowest model cost. But the former model also cost and weighed the most, while the later model ranked only third in terms of mAP, hence the improved model with only 0.1% lower mAP, only 0.58% higher size, and only 0.39% higher cost than the first ranked models is arguably the most balanced model. As a result, the improved model when deployed in an edge-AI device will have lower hardware limitations due to the reduced computational complexity, will be easier to deploy due to the reduced size, and will be more accurate in detecting damaged canola kernels.

Table 2. 3. Comparison of model performance with Convolutional Block Attention Module (CBAM) at different locations within the model

CBAM Option	P (%)	R (%)	F1 Score (%)	mAP@0.5 (%)	Size (Mb)	FPS	GFLOPs
YOLOv7_CBAM1	96.4	96.9	96.6	97.2	84.5	38	112.9
YOLOv7_CBAM2	94.4	97.2	95.7	98.3	79.0	52	108.5
YOLOv7_CBAM3	96.2	95.7	95.9	97.4	84.2	46	112.7

YOLOv7_CBAM4	97.1	97.7	97.4	98.7	75.1	50	105.4
YOLOv7_CBAM5	96.3	98.1	97.2	97.7	79.3	50	108.8

Table 2. 4. Comparison of model performance when different modules with standard convolution are replaced with ghost convolution.

Module(s)	P (%)	R (%)	F1 Score (%)	mAP@0.5 (%)	Size (Mb)	FPS	GFLOPs
All Modules	95.7	95.4	95.5	97.5	45.3	44	61.3
All ELAN	96.5	98.1	97.3	98.2	66.3	51	86.7
All ELAN_W	97.0	96.9	96.9	97.7	67.1	51	97.3
SPPCSPC	97.1	97.8	97.4	97.5	67.3	52	102.1
All MP	96.8	97.4	97.1	98.2	69.5	52	98.7
ELAN + ELAN_W	96.5	97.3	96.8	98.2	58.6	50	78.8
ELAN + ELAN_W + SPPCSPC	97.1	97.1	97.1	98.3	51.1	50	75.8

2.4.3. Improved Detection Model Performance Comparison

To highlight that our developed model is the best among its counterparts in detecting damages to canola kernels, the performance of YOLOv7_ours was compared with other leading networks in terms of F1 score and mAP@0.5. To ensure fairness and accuracy of comparison, all models including those used for comparison with YOLOv7_ours (YOLOv5 - YOLOv8) were trained for 150 epochs under the same training environment, in the same dataset, and using the recommended hyperparameters for each model. The results of these comparisons (Table 2.6) outlined that in terms of both mAP@0.5 IoU and F1 score the YOLOv7_ours model outperformed all the other models, including the newer version of YOLO; YOLOv8 by 0.81% in mAP@0.5 and 1.03% in F1 Score. However, YOLOv8 reported the highest precision value of 98.1% which was

the highest among all the models including the improved model. In terms of F1 score, the YOLOv7_ours is 1.14%, 0.31%, and 1.03% better than YOLOv5m, YOLOv7, and YOLOv8m, respectively. These results demonstrate that incorporating systemic enhancements into the network architecture consistently outperforms off-the-shelf models, regardless of their version, in tasks they have been trained to perform.

Fig. 2.8 compares the PR curves obtained from the best model (YOLOv7_ours) and the second-best model (YOLOv8) in terms of mAP@0.5. The PR curve is useful to illustrate the trade-off between precision and recall at various classification thresholds. The two PR curves indicate that in the immature class, YOLOv7_ours is 0.50% better and in the heated class it is 1.13% better. Additionally, the greater area-under-curve in the PR curve of the YOLOv7_ours model indicates that the improved model is able to provide an accurate and reliable prediction for positive instances while minimizing false positives. Furthermore, the final detections made on a selected random test image out of eight such images by four different models (YOLOv5, YOLOv7, YOLOv8, YOLOv7_ours) are visualized in Fig. 2.9. It can be observed that the improved model didn't report any false positive or false negative detections. While the YOLOv5 model failed to detect one heated seed and classified and localized three sound seeds as heated. The YOLOv7 model made one false positive detection where it detected one sound seed as heated and the YOLOv8 made one false negative detection where it failed to detect one heated seed present in the test image #8. It is interesting to observe that all the false positive cases were seen only in the heated class. It is because the off-the-shelf models got confused between the feature of the seed coat and the feature of the cotyledon of heated seeds and detected them as heated. This indicates that the addition of two CBAM modules in the improved model helped it to selectively focus only on the cotyledon region, preventing it from making the same mistake.

Table 2. 5 Evaluation results of ablation experiment.

YOLOv7	CBAM	Ghost Conv	P (%)	R (%)	F1 Score (%)	mAP@0.5 (%)	Size (Mb)	FPS	GFLOPs
✓			96.7	96.5	96.6	97.6	74.8	53	105.1
✓	✓		97.1	97.7	97.4	98.7	75.1	50	105.4
✓		✓	97.1	97.1	97.1	98.3	51.1	50	75.8

✓	✓	✓	97.2	96.6	96.9	98.6	51.4	48	76.1

Table 2. 6. Performance of other detection models compared to YOLOv7_ours.

Model	P (%)	R (%)	F1 Score (%)	mAP@0.5 (%)
YOLOv5m	95.8	95.8	95.8	96.9
YOLOv7	96.7	96.5	96.6	97.6
YOLOv8m	98.1	93.9	95.9	97.8
YOLOv7_ours	97.2	96.6	96.9	98.6

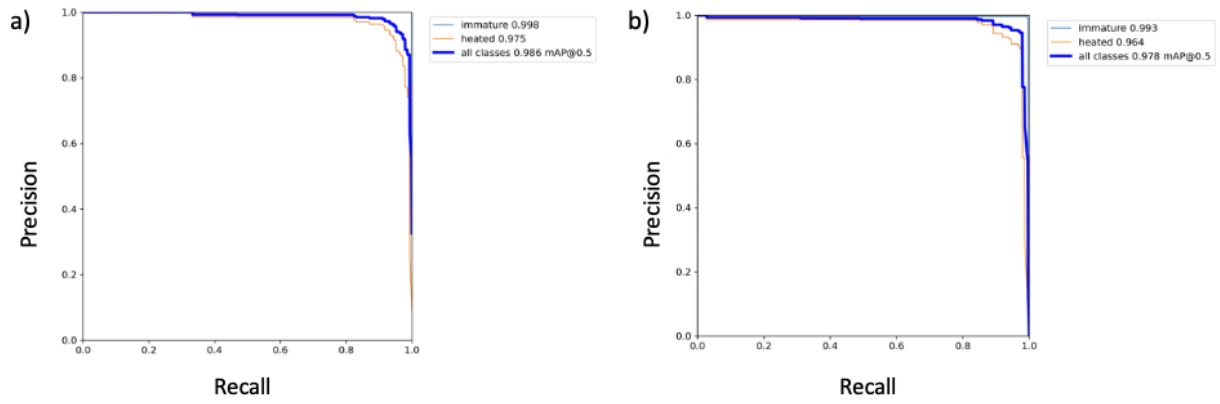


Fig. 2. 8. The Precision-Recall curve of (a) YOLOv7_ours, (b) YOLOv8.

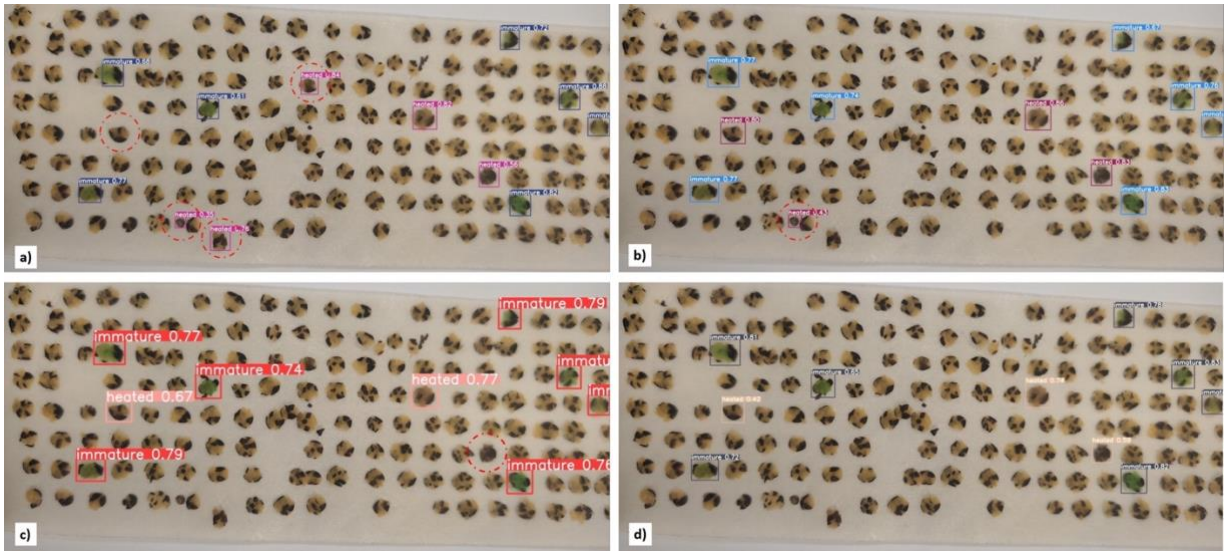


Fig. 2. 9. Detections made by (a) YOLOv5m, (b) YOLOv7, (c) YOLOv8m, and (d) YOLOv7_ours models on the same representative test image T8. The false positives and false negatives are encircled in red.

2.4.4. Tracking and Counting Results

This section highlights the tracking and counting results in the test video chosen to analyze the performance of the ByteTrack multi-object tracker and the line-cross counting algorithm. In the selected test video, three cases of Identity (ID) switching, and one case each of false positive and false negative detection were observed. The ground truth was determined to be 33 damages, so the MOTA achieved in this video by the ByteTrack tracker was calculated to be 84.8% by using Eq. 6, which for comparison is 5.49% higher than the MOTA achieved by it in the MOT17 test set [26]. Out of the three ID switching's observed, two were observed in the same immature seed and one was observed in a heated seed (Fig. 10). In the 1st frame, the immature seed was assigned the ID of 413 but in the 37th frame this ID was lost and in the 41st frame the first switch to ID 417 was observed. After maintaining the track for 33 frames, the detection was again lost in the 74th frame, finally, a new ID of 430 was assigned to the same immature seed in the 81st frame. In the other case of the heated seed, the ID of 434 was assigned to it in the 93rd frame where it first appeared in the video. The detection was lost in the 115th frame and 69 frames later (184th frame) a new ID

of 460 was assigned to it. In the second ID switching instance of the immature seed, the ID was lost after 33 frames as the detector was unable to detect the damaged seed with a confidence score above the set threshold of 0.35, but once the detection was reclaimed, the tracker was also unable to associate the two detections in the 73rd frame and in the 81st frame as the same object. This might be because of the similar appearance and motion patterns of the detected immature seed. This hypothesis also holds for the second instance of ID switching in the heated seed.

As for the performance of the counter, ID switching didn't directly affect the algorithmic count as the counter was developed such that it is updated only when the bounding box of the detected damaged seed crosses the pre-determined line. The two seeds which were initially identified as 413 and 434 crossed the line as 430 and 460, respectively. Hence these three cases of ID switching did not affect the count. These three instances also justify having an extra step for counting the detections as opposed to counting solely based on the number of unique IDs generated, which would have resulted in three more than the actual count. Fig. 2.11 shows the state of the counter in four different situations, a) initially when no objects cross the line, b) when ID 430 crosses it, c) when ID 460 crosses it, and finally d) at the end of the video with only eight objects left to cross. The final count is observed to be 33 which is equal to the actual number of damages present in it. Using Eq. 14, the counting accuracy can be calculated to be 100%, but this value is misleading as the original formula doesn't account for false positive and false negative detections. There was one false positive bounding box that crossed the line which made up for the false negative case. So, accounting for the false and missed detections the counting accuracy in this test video can be adjusted to be 93.9%. In this test strip, the total damage was therefore 6.6%, and it can be assigned the grade of No. 2 Canada.

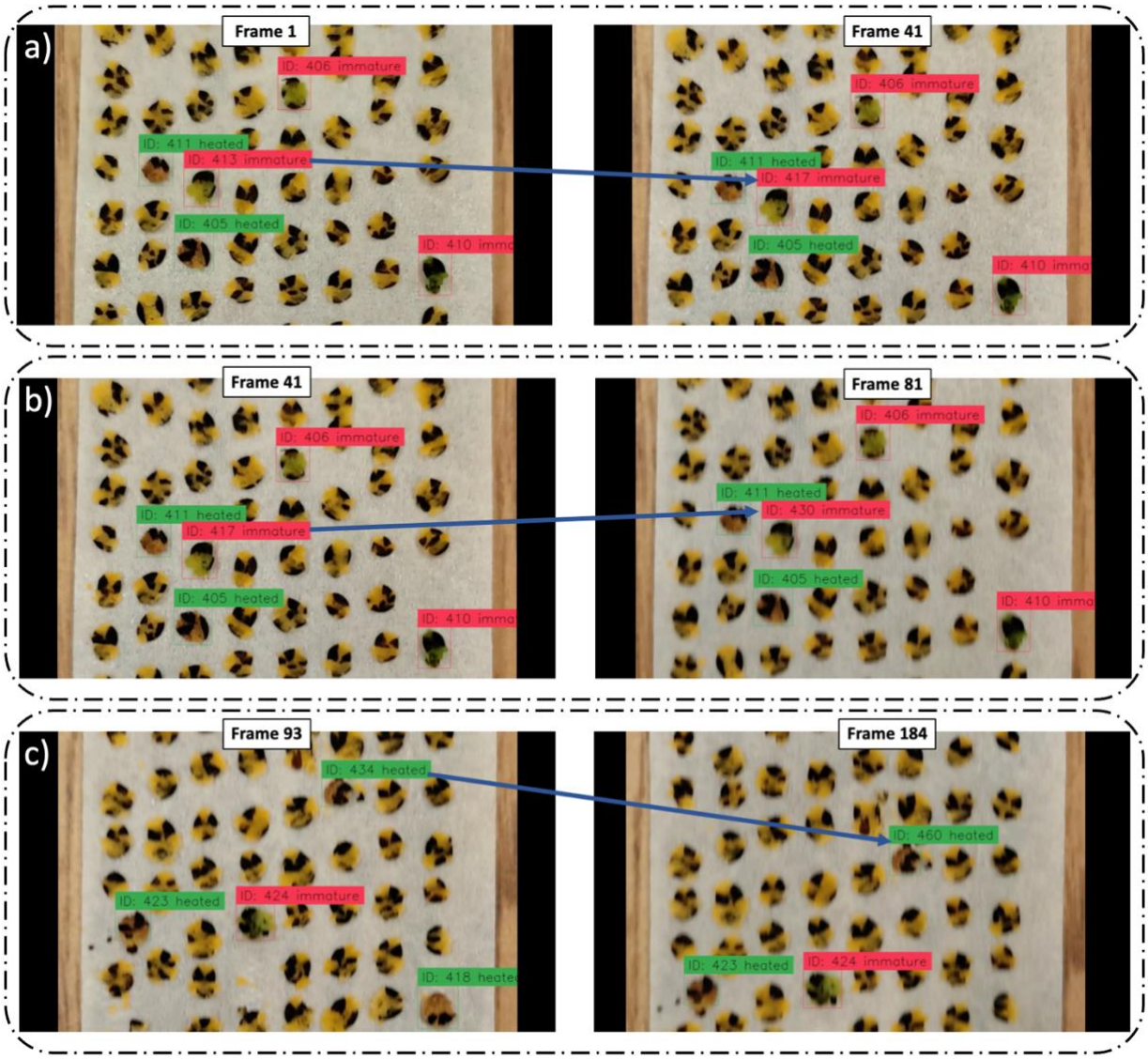


Fig. 2. 10. The three instances of identity switching observed in the test video while tracking the detected objects.

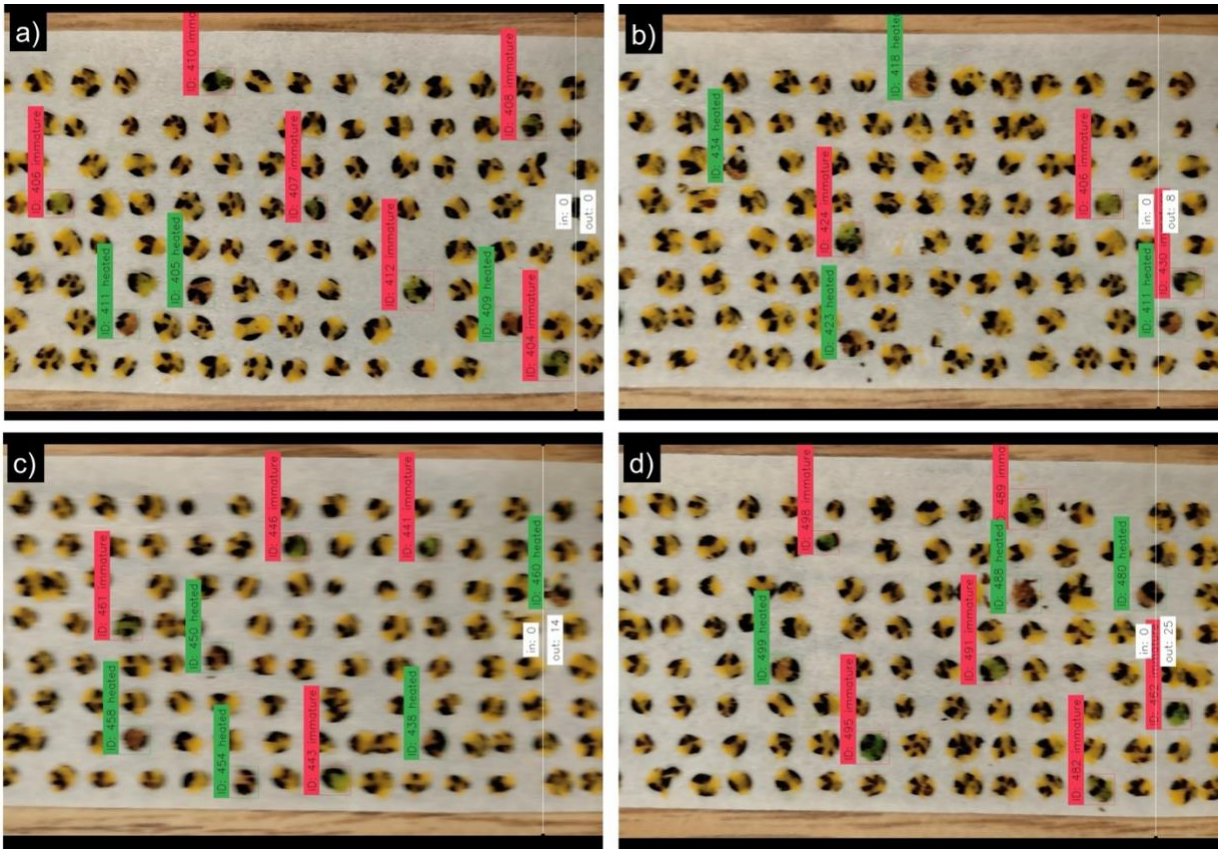


Fig. 2. 11. The state of the counter when (a) no object has crossed it yet, (b) when the object with ID 430 has crossed it, (c) when the object with ID 460 has crossed it, and (d) before the rest of the eight objects are left to cross it.

2.5. Conclusion

Current canola quality control methods still rely on a manual grading process, which is time-taking, laborious, and prone to errors. To address this issue, the study focussed on developing a model for a computer vision system that has the potential to replace the current laboratory process. By automating the grading process, the system could increase efficiency, reduce labor costs, and ensure consistent and accurate grading, ultimately contributing to the continued success of the canola industry. This process was done in three steps, which include detecting the damages present using an improved single-shot object detection model based on YOLOv7, tracking the

detected objects with the ByteTrack MOT, and counting the total number of damaged seeds to set the grade of the canola using a line-cross counting algorithm. As the target objects were tiny and were mostly covered with seed coat revealing only a small portion of the relevant feature, cotyledon, base models couldn't perform well in the detection task, and hence two improvements were performed, viz. addition of two CBAM modules and replacement of the standard convolution layers with ghost convolution layers in all the ELAN, ELAN_W, and SPPCSPC modules present in the YOLOv7 network. These two improvements resulted in 1.02% higher mAP@0.5, 0.31% higher F1 Score, 37.1% lower model size, and 32.1% lower model cost when compared to the baseline model. The ByteTrack tracker was able to track all the detected targets with a MOTA of 84.8% and the accuracy achieved by the counter in the same video was 93.9%.

References

[1] Statistics Canada, "Statistics Canada," 2022. https://www.statcan.gc.ca/en/start (accessed Jun. 01, 2023).

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Jun. 2015, [Online]. Available: http://arxiv.org/abs/1506.02640

[3] X. Li, Y. Du, L. Yao, J. Wu, and L. Liu, "Design and experiment of a broken corn kernel detection device based on the yolov4-tiny algorithm," *Agriculture (Switzerland)*, vol. 11, no. 12, Dec. 2021, doi: 10.3390/agriculture11121238.

[4] A. Nasirahmadi, U. Wilczek, and O. Hensel, "Sugar beet damage detection during harvesting using different convolutional neural network models," *Agriculture (Switzerland)*, vol. 11, no. 11, Nov. 2021, doi: 10.3390/agriculture11111111.

[5] H. Mirhaji, M. Soleymani, A. Asakereh, and S. Abdanan Mehdizadeh, "Fruit detection and load estimation of an orange orchard using the YOLO models through simple approaches in different imaging and illumination conditions," *Comput Electron Agric*, vol. 191, Dec. 2021, doi: 10.1016/j.compag.2021.106533.

[6] F. Jubayer *et al.*, "Detection of mold on the food surface using YOLOv5," *Curr Res Food Sci*, vol. 4, pp. 724–728, Jan. 2021, doi: 10.1016/j.crfs.2021.10.003.

[7] I. Gallo, A. U. Rehman, R. H. Dehkordi, N. Landro, R. La Grassa, and M. Boschetti, "Deep Object Detection of Crop Weeds: Performance of YOLOv7 on a Real Case Dataset from UAV Images," *Remote Sens (Basel)*, vol. 15, no. 2, Jan. 2023, doi: 10.3390/rs15020539.

[8] M.-H. Guo *et al.*, "Attention Mechanisms in Computer Vision: A Survey," Nov. 2021, doi: 10.1007/s41095-022-0271-y.

[9] Q. Wang, M. Cheng, S. Huang, Z. Cai, J. Zhang, and H. Yuan, "A deep learning approach incorporating YOLO v5 and attention mechanisms for field real-time detection of the invasive weed Solanum rostratum Dunal seedlings," *Comput Electron Agric*, vol. 199, Aug. 2022, doi: 10.1016/j.compag.2022.107194.

[10] J. Chu, Y. Li, H. Feng, X. Weng, and Y. Ruan, "Research on Multi-Scale Pest Detection and Identification Method in Granary Based on Improved YOLOv5," *Agriculture (Switzerland)*, vol. 13, no. 2, Feb. 2023, doi: 10.3390/agriculture13020364.

[11] J. Ma, A. Lu, C. Chen, X. Ma, and Q. Ma, "YOLOv5-lotus an efficient object detection method for lotus seedpod in a natural environment," *Comput Electron Agric*, vol. 206, Mar. 2023, doi: 10.1016/j.compag.2023.107635.

[12] Y. Zhang, J. Yu, Y. Chen, W. Yang, W. Zhang, and Y. He, "Real-time strawberry detection using deep neural networks on embedded system (rtsd-net): An edge AI application," *Comput Electron Agric*, vol. 192, Jan. 2022, doi: 10.1016/j.compag.2021.106586.

[13] Y. Shang, X. Xu, Y. Jiao, Z. Wang, Z. Hua, and H. Song, "Using lightweight deep learning algorithm for real-time detection of apple flowers in natural environments," *Comput Electron Agric*, vol. 207, p. 107765, Apr. 2023, doi: 10.1016/j.compag.2023.107765.

[14] J. Chen, H. Liu, Y. Zhang, D. Zhang, H. Ouyang, and X. Chen, "A Multiscale Lightweight and Efficient Model Based on YOLOv7: Applied to Citrus Orchard," *Plants*, vol. 11, no. 23, Dec. 2022, doi: 10.3390/plants11233260.

[15] L. Shen *et al.*, "Real-time tracking and counting of grape clusters in the field based on channel pruning with YOLOv5s," *Comput Electron Agric*, vol. 206, Mar. 2023, doi: 10.1016/j.compag.2023.107662.

[16] H. Yang *et al.*, "Multi-object tracking using Deep SORT and modified CenterNet in cotton seedling counting," *Comput Electron Agric*, vol. 202, Nov. 2022, doi: 10.1016/j.compag.2022.107339.

[17] Z. Zheng, J. Li, and L. Qin, "YOLO-BYTE: An efficient multi-object tracking algorithm for automatic monitoring of dairy cows," *Comput Electron Agric*, vol. 209, Jun. 2023, doi: 10.1016/j.compag.2023.107857.

[18] H. Zhang, W. Li, Y. Qi, H. Liu, and Z. Li, "Dynamic fry counting based on multi-object tracking and one-stage detection," *Comput Electron Agric*, vol. 209, Jun. 2023, doi: 10.1016/j.compag.2023.107871.

[19] Canadian Grain Commission, "Canadian Grain Commission," Jul. 29, 2022. https://www.grainscanada.gc.ca/en/grain-quality/official-grain-grading-guide/10-canola-rapeseed/primary-grade-determinants-tables.html (accessed Jun. 01, 2023).

[20] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," Jul. 2022, [Online]. Available: http://arxiv.org/abs/2207.02696

[21] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional Block Attention Module,” Jul. 2018, [Online]. Available: http://arxiv.org/abs/1807.06521

[22] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “GhostNet: More Features from Cheap Operations,” Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.11907

[23] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple Online and Realtime Tracking,” Feb. 2016, doi: 10.1109/ICIP.2016.7533003.

[24] G. Welch and G. Bishop, “An Introduction to the Kalman Filter.” [Online]. Available: http://www.cs.unc.edu/~gb

[25] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Research Logistics*, vol. 52, no. 1, pp. 7–21, Feb. 2005, doi: 10.1002/nav.20053.

[26] Y. Zhang *et al.*, “ByteTrack: Multi-Object Tracking by Associating Every Detection Box,” Oct. 2021, [Online]. Available: http://arxiv.org/abs/2110.06864

[27] A. I. B. Parico and T. Ahamed, “Real time pear fruit detection and counting using yolov4 models and deep sort,” *Sensors*, vol. 21, no. 14, Jul. 2021, doi: 10.3390/s21144803.

CHAPTER 3. Real-time canola damage detection: An end-to-end framework with semi-automatic crusher and lightweight ShuffleNetV2_YOLOv5s

This chapter is based on the manuscript published in the journal "Smart Agricultural Technology" entitled 'Real-time canola damage detection: An end-to-end framework with semi-automatic crusher and lightweight ShuffleNetV2_YOLOv5s', reprinted with permission.

3.1 Abstract

The current method employed in detecting damages present in canola kernels is inefficient and time-consuming. The sample preparation process is laborious and the judgment between sound and damaged seeds using visual methods is prone to errors as they are small and indistinguishable. To date, there is no hardware and supporting software solution that can expedite this time-consuming yet crucial task. This study demonstrates an end-to-end damage detection framework that has two components: a semi-automatic machine to crush the canola seeds to prepare the samples rapidly for detecting the damages and a supporting object detection algorithm based on a compressed YOLOv5s model. The lightweight detection model was deployed in an Edge-AI device and integrated with the crusher machine to detect the damages in real-time. Two Convolutional Neural Network (CNN) architectures, viz. ShuffleNetV2 and MobileNetV3 were compared in terms of model parameters, computational cost, and model size to replace the standard CSPDarknet53 CNN backbone present in YOLOv5s. The best-performing model was deployed into a Jetson Nano embedded device for real-time inferencing. Results indicate that in terms of model size, the ShuffleNetV2_YOLOv5s model was 56.5% and 80.4% smaller than the MobileNetV3_YOLOv5s and the baseline YOLOv5s models, respectively. While, in terms of computational cost, the ShuffleNetV2_YOLOv5s model was 64.1% and 99.5% less expensive than the MobileNetV3_YOLOv5s and the baseline YOLOv5s models, respectively. This study demonstrated an end-to-end framework for detecting damages in canola supported by a real-time and lightweight damage detection model for automated grading applications in the agricultural industry.

Keywords: Canola quality; Real-Time Inferencing; Object Detection; Network Compaction; Edge-AI

3.2. Introduction

Canada is the largest producer and exporter of canola in the world. The canola price is largely influenced by the quality of the harvest, which relates to the total number of damages present in them. The two main types of damages present in canola are heated and immature seed conditions which can only be determined based on the appearance of their cotyledon, viz. dark brown colored cotyledon for heat damaged and green colored cotyledon for immature seed conditions. The standard practice for determining these damages is to first crush the canola seeds using a hand-held roller and make decisions based on the observation made by trained personnel.

These two damage conditions adversely affect the overall value and price of the canola kernels as they affect the subsequent downstream process of oil extraction and oil purification. The oil derived from immature seeds results in a darker shade due to the presence of chlorophyll. To remove the undesired pigments, processors have to treat it with more amount of clay. Due to this treatment, an additional filtration step is required which leads to a lower final yield and an increased processing cost. Furthermore, immature seeds are also less stable in storage and have lower oil content per tonne. The enzyme *pheophorbide-a-oxygenase* is an important enzyme responsible for the degradation of chlorophyll during the maturation process of canola seeds [1]. However, during the seed development process, exposure to frost or hot dry weather, or hot air from the fan during storage disrupts the proper functioning of the enzymes which are only active in temperatures above 5°C and seed moisture content above 20%. These extreme conditions before harvest or due to improper storage conditions in silos can denature the enzymes, completely stagnating the maturity of the seeds. The high temperature is also the main reason for the formation of heated seeds. Even though the causes of immature and heated seed formations are known, there are no ways to reverse these conditions.

Therefore, detecting the damaged canola seeds is a crucial step performed by grain buyers and elevators to correctly assess the value of the canola before export. The canola is graded based on the total percentage of damage into three classes (i.e., 0 – 5%, 5 – 12%, 12 – 25%). The current system in place for sample preparation is by utilizing a hand-held roller. The seeds are crushed, and the damages are detected based on visual judgment by referring to a color reference chart. Moreover, the traditional approach for damage detection is laborious and time-consuming due to

the manual sample preparation step and is also prone to subjectivity and other human errors due to the element of decision-making relying solely on human vision.

The current system, therefore, is not a true reflection of the current progress made in automation technology, especially the advancements made in the utilization of Artificial Intelligence models in agriculture, especially computer vision algorithms for abnormality and damage detection in agri-food systems. Several studies have been conducted to detect diseases, defects, and damages in agricultural products such as grains [2], crop leaves [3], fruits [4], and vegetables [5] using single-stage object detection algorithms such as the You Only Look Once (YOLO) [6] family of models. For instance, in a study to detect Fusarium head blight infection in corn grains, [7] utilized the YOLOv5s algorithm which resulted in a mAP@0.5 of 99%. Similarly, [8] used the YOLOv5 model to detect three types of fungal damage in rice kernels with a mAP@0.5 of 89.26%, 91.15%, and 90.19%, respectively. All these studies indicate that the YOLO series of algorithms are suitable techniques for detecting damages and other abnormal conditions in agricultural products. However, the YOLO models without any network-compaction modifications are unsuitable for deployment in resource-constrained portable devices such as Intel Neural Computing Stick, NVIDIA Jetson Nano, or Raspberry Pi due to them being incompatible with these devices because of their higher model size and having a higher amount of Floating-Point Operations per Second (FLOPs) also referred to as the computational cost of models. Therefore, the off-the-shelf object detection models used in the above studies, which provide an excellent base for model development, are not capable of being properly deployed in these low-powered and resource-constrained hardware environments. Thus comes the need to develop lighter models in terms of cost and size by modifying the network architectures of these standard object detection models.

“Lightweight” models are achieved by implementing three primary types of network compaction strategies: 1. Model Compression, 2. Knowledge Distillation, and 3. Modification of Network Structures [9]. In recent years, a handful of studies in the agricultural domain, have been conducted to achieve this feat using one of the three available techniques. For instance, to decrease the size and cost of the YOLOv7 model for detecting damages to canola kernels, [10] developed the YOLOv7_Ours model by replacing the convolutional layers present in the ELAN and SPPCSPC modules with ghost layers, which enabled the model to have a 32.1% lower model size and 37.1% lower model cost. In a study to develop a tea leaf detection model for deployment in a

Raspberry Pi 4 device, [11] reduced the model size of the baseline YOLOv5 model from 334 MB to 90.1 MB by restructuring the backbone using the ShuffleNetV2 feature extractor followed by channel pruning. Similarly, [12] used the ShuffleNetV2 model in the backbone of the YOLOv5s model to detect sugarcane aphids, which resulted in a 92.5% size reduction and 81.6% cost reduction. The study, [13] replaced the backbone of the YOLOv5 model with the GhostNet CNN architecture to detect agricultural pests, which resulted in a 42% reduction in cost and a 39% reduction in size. Furthermore, the MobileNetV3 CNN was used as a backbone replacer of the YOLOv7 model in a study conducted by [14] to detect rice pests. All these studies indicated that restructuring the network architecture with efficient and lightweight CNNs such as the MobileNetV3, ShuffleNetV2, and GhostNet decreases the model size and cost for compatible deployment in embedded platforms. Although a few studies have been conducted on compacting the pre-existing neural networks with the goal of deploying them in mobile devices, there are not enough studies that explicitly demonstrate the end-to-end application of using these developed models.

Therefore, the first objective of this study was to develop a complete end-to-end computer vision pipeline for detecting damages to canola seeds. There are three steps in the first objective (i) Development of two novel datasets (D1 and D2) containing damaged canola seeds using two sample preparation methods (manual crushing and crushing based on a semi-automatic unit), (ii) Training a YOLOv5s model to detect the two damage conditions followed by making improvements in the neural network architecture to make it suitable for deployment in resource-constrained and portable devices. This was achieved by replacing the backbone, which acts as the primary feature extractor, with lighter convolutional neural network architectures especially designed for mobile devices or low-compute power devices, (iii) Deploying the developed models into a low-cost GPU-based edge device (NVIDIA Jetson Nano) for smart and real-time canola quality prediction. The second objective was to perform a comparative study between the two sample preparation methods (manual and semi-automatic canola crushing). The time required to crush the canola seeds and the effect of the crushing methods on the canola damage detection performance was investigated.

3.3. Materials and Methods

3.3.1. Sample preparation: Hand-Held Roller and Semi-Automatic Machine

First, a sample set containing 20 strips of paper consisting of 500 crushed canola kernels was created using the traditional canola crushing method using a hand-held roller. The hand-held roller was purchased from Dimo's Labtronics (Winnipeg, MB) and it includes a platform with 500 perforations to place the canola seeds for crushing. To crush the canola using this method, a handful of seeds was gently placed on the perforated platform, followed by a masking tape which was placed directly above the platform with the seeds, the glue side facing inwards. The hand-held roller was then firmly rolled to crush the seeds. Each of these strips contained 500 crushed canola seeds including the three seed conditions. In total, 380 immature seeds (3.80% of the total 10,000 crushed seeds) and 143 heated seeds (1.43%) were identified by a trained person.

Second, an equal number of strips (20) containing the crushed seeds were created using a canola-crushing machine. The semi-automatic canola crusher machine (Devloo Canola Crusher, Manitoba, Canada) uses aluminum and stainless steel as its material of construction and it has four primary parts – frame, seed hopper, and crush and pickup wheel. The machine needs four rotations of the crank to crush 500 kernels of canola. The canola crushing unit works as follows: On the first rotation of the crank, the seeds fed onto the hopper get transferred to the rotating pickup wheel. The tape which is attached to the upper part of the machine acts as a carrier for the seeds and passes the crushing rollers in the successive rotations of the crank. The crushed seeds attached to the tape were then visually analyzed for damage by an expert. Here, in total, a trained person identified 337 immature seeds (3.37%) and 219 heated seeds (2.19%).

3.3.2. Image Acquisition, Annotation, and Augmentation to Create the Datasets

The first step before training the detection model is to create the dataset comprising all three classes: i.e., sound kernels, heated kernels, and immature kernels. Contrary to the popular approach of presenting each class in separate images for model training and object detection workflow, in this study all three classes were presented together in a single image. Even though this approach made the model training complex, this method was followed to match the sample preparation guidelines for grading canola (Section 3.2.1).

A standard camera (Sony CMOS, 48 MP, f/1.78, with OIS) was used to capture the image of each strip under constant lighting conditions, where each strip was placed directly below an overhead 20W broadband light and from a constant height of 30 cm. The dataset resulting from the first method of sample creation was named D1 and the dataset corresponding to the second method of sample preparation was named D2. In total, there were 240 images (each strip was divided into three and then they were captured in four different backgrounds) in each dataset based on the method of creation. For the task of annotating each object of interest present in the image, the Makesense AI tool was used to create the ground-truth bounding boxes. The validation of the annotation process was conducted in the presence of an expert grader, according to the guidelines issued by the Canadian Grain Commission. To minimize fatigue and to prevent wrong annotations, the annotation process was done in a span of six days, two hours each day. Regular breaks of five minutes were taken after every thirty minutes of work. The utmost care was taken to prevent false annotation while preparing the ground truth for model training as wrong ground truth can negatively affect the training results (Section 3.4.). Furthermore, to increase the size of the training data, the image augmentation techniques of horizontal and vertical flip, image scaling, image shearing, image rotation, and image mosaicing were applied to each image present in both datasets. These image augmentation operations were achieved using the open-source Albumentation library. The two separate datasets (D1 and D2) were created to check the effect of model performance based on the method of sample creation and also as a means to compare these two methods (Fig 3.7). It is hypothesized that models trained on D2 will outperform models trained on D1 due to the higher quality of the dataset and the non-touching nature of the crushed canola kernels. The difference between the two datasets is shown in Fig.3.1. Both of these datasets were split in a ratio of 8:2 to create the training and validation sets and were used separately henceforth. In total, there were 1079 instances before augmentation, out of which 717 instances belonged to the immature class and 362 instances were of the heated category. The augmentation process further increased the number of each instance by 40%.

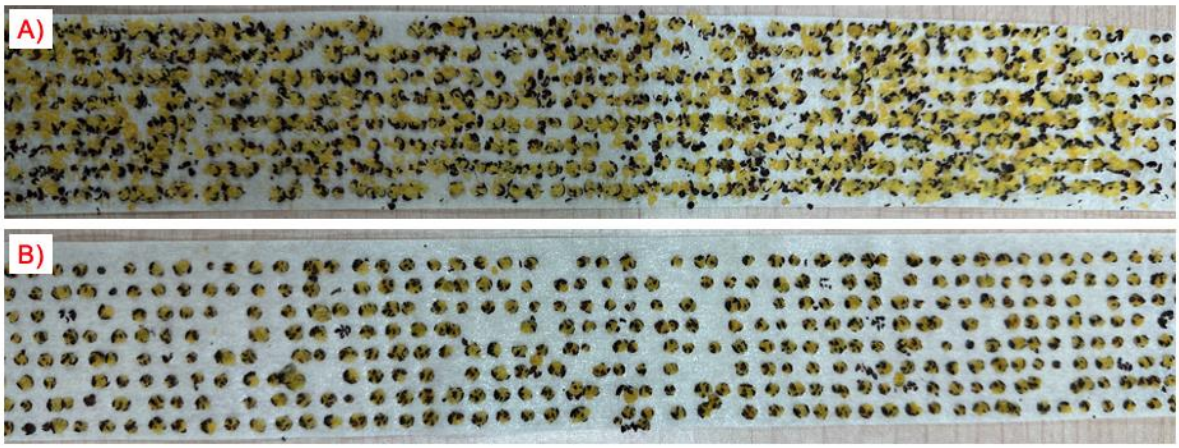


Fig. 3. 1. The two imaging datasets created from the two methods of sample preparation. (A) traditional hand-held roller, (B) Semi-automatic canola crushing machine.

3.3.3. Sample Preparation Method Comparison

The two methods used in preparing the samples and subsequently for creating the dataset for this study (D1 and D2) were compared based on two parameters: 1) the performance of an object detection model for detecting the target objects and 2) the average time taken to prepare the samples. For the first comparison task, an object detection model was trained on the two sets of datasets separately. Secondly, the time required to create 15 strips of samples using both methods by an expert was measured using a stop clock. The best method of sample preparation is reported in Section 3.4.1.

3.3.4. Selection of Object Detection Model for Damage Detection: One-stage vs Two-stage networks

There are two algorithmic approaches towards object detection, viz. via one-stage networks and via two-stage networks. The two-stage network proposes multiple regions of interest where there is a high probability of an object of interest being present. Each region is then fed to a convolutional neural network to classify if the pixel values correspond to the ground truth bounding box. True Positive cases are then assigned a class along with a bounding box indicating

the location of the object in an image. An object is said to be correctly detected if the intersection over union (IoU) value is above a certain threshold, usually 0.5 for most tasks including ours. The most popular two-stage detectors are the Region-based Convolutional Neural Network (R-CNN) families of models. The R-CNN network uses the Selective Search algorithm to propose around 2000 regions of interest in the inputted image following a CNN to extract features from each region. The output of the CNN is then fed to a Support Vector Machine classifier to classify the object located in that region and a Linear Regression model to tighten the object's bounding box. Because of the multiple steps and models involved in making a detection, the average detection speed achieved by R-CNN is 47 s per image in the COCO dataset [15]. Fast R-CNN was developed to address this issue. In Fast R-CNN [16], feature extraction using a CNN is performed before proposing regions of interest. The output is then fed to a Softmax layer instead of a support vector classifier used in R-CNN. This algorithm is succeeded by the Faster R-CNN [17], which uses an algorithm called region proposal network to propose regions. The second stage of this model is the same as Fast R-CNN. Despite the two-stage networks being more accurate, they show inadequate performance when it comes to inference speed required to make real-time detections, especially in edge-computing devices.

The YOLO series of models, which is a one-stage detector, fixes this limitation of lower inference speed by treating the task of object detection as a regression problem, where a single convolutional neural network predicts the bounding boxes and class probabilities of each object of interest in one single pass. To achieve this, each image is divided into an N x N matrix of grids, and the cell in which the object's midpoint lies is responsible for detecting that object. The bounding boxes output the coordinates of the center of the object relative to the dimension of the grid along with the width and height of the object relative to the overall image. To eliminate multiple detections by a single grid cell, non-maximal suppression is implemented [6]. The approach of treating detection as a one-stage regression problem enabled the first version of the YOLO network to perform real-time object detection at 45 frames per second (FPS) compared to only 0.5 FPS achieved by the then state-of-the-art Fast R-CNN in the COCO dataset [18]. Despite its fast inference speed, the first version of YOLO had major issues with detecting small objects in groups and it could only predict the bounding box of a single object per grid cell, thus YOLO9000 [19] was introduced with batch normalization layers which improved the mean average precision (mAP) by 2% and introduced anchor boxes which allowed for detection of five

objects per cell. For feature extraction from input images, the second version also introduced the DarkNet-19 convolutional neural network architecture as the backbone of the network. The backbone component of the YOLO model is a convolutional neural network that performs the function of extracting features from the input image. The third iteration of this model, YOLOv3 further improved the feature extraction performance by increasing the depth of the backbone model (number of layers) to 106 with the introduction of the Darknet53 neural network [20]. YOLOv4 [21] introduced an intermediate neck component using a Path Aggregation Network (PAN) with Spatial Pyramid Pooling (SPP) between the backbone and the head, which is used as a feature aggregator, responsible for collecting feature maps from the backbone. It also introduced the concepts of Bag of Freebies (BoF) and Bag of Specials (BoS). BoF uses techniques to improve the mean Average Precision (mAP) without increasing the cost of training. These techniques included adding mosaic data augmentation, introducing a new loss function (CIoU-loss), and self-adversarial training in the head, among others. On the other hand, BoS applies techniques to significantly improve the performance with a slight improvement in detection speed. These improvements were the addition of Cross-stage partial (CSP) connections in the Darknet53 backbone along with the use of Mish activation functions in the backbone and detector components. These changes improved the mAP by 10% compared to YOLOv3. YOLOv5, which was released just after YOLOv4, was the first model to use PyTorch instead of the original Darknet framework on which all the previous versions were based. One notable improvement in YOLOv5 was the introduction of AutoAnchor, which is a novel method of generating anchor boxes to improve the aspect ratio to have a better fit with the target objects. When generating new anchors, AutoAnchor first applies a KMeans function against the ground truth bounding boxes and uses the centroids outputted by the function as the initial condition for a Genetic algorithm for anchor verification and generation. In addition to that, one more improvement compared to YOLOv4 is the use of SPPF (Spatial Pyramid Pooling - Fast) instead of SPP. Both SPP and SPPF modules produce mathematically identical outputs, but the latter uses less computational resources (in terms of FLOPS) than the former. It does so by running convolutions in series and then concatenating the outputs compared to just running them in parallel.

3.3.4.1. YOLOv5s: The Baseline Canola Damage Detection Model

For detecting the damaged kernels and modifying it to build a lighter version for subsequent deployment in an embedded device, the YOLOv5s (version 6.1) object detection model was adopted. The YOLOv5s model belongs to the YOLOv5 family of models that consists of other models like nano, small, medium, large, and X-large. It is a one-stage detector developed by [22]. Similar to the previous iterations of YOLO, it consists of three parts to its architecture: backbone, neck, and head. The backbone is responsible for processing the input images and extracting hierarchical features that are used for detecting the objects of interest. YOLOv5s uses the Cross-Stage-Partial Darknet 53 (CSPDarknet53) [23] convolutional neural network model as the backbone. This CNN structure is a 53-layer deep network, where all the layers are stacked to form a hierarchy of features. The layers present in the bottom half are responsible for capturing low-level details, while the top layers are tasked with capturing more abstract and semantically meaningful features. The neck is responsible for multi-scale feature fusion and improving the contextual information fetched from the images. This is achieved via the use of the Spatial Pyramid Pooling Fast (SPPF) [24] and the Cross-Stage Partial - Path Aggregation Network (CSP - PAN) [25] architectures. The SPPF module computes spatial pooling features at multiple scales, while the PAN [26] module aggregates spatial context and channel information across feature maps to improve detection accuracy. Finally, YOLOv5s uses the detection head of YOLOv3 [20] for predicting object bounding boxes, class probabilities, and objectness scores at different scales.

3.3.5. Improved ShuffleNetV2_YOLOv5s Network Architecture

The choice of backbone architecture significantly impacts the overall model size and computational complexity. Inherently “lightweight” CNN structures such as ShuffleNet [27] and MobileNet [28] have fewer parameters and require fewer computations during both training and inference, which can lead to lower model size and lower computational complexity than CSPDarknet53. Both these enhancements are advantageous for deployment in resource-constrained devices due to storage, memory, and computational power limitations. Therefore, in this study, the backbone of the original model was redeveloped using the ShuffleNetV2 network. The ShuffleNetV2 [29] model was chosen as it is an efficient and lightweight CNN architecture designed particularly for mobile and embedded devices (Section 3.2.). The previous version, ShuffleNetV1, [27] was not chosen for this study as it was less efficient due to problems such as higher memory access cost (MAC) and low parallelism in computing, due to certain design choices

of incorporating network architectural components such as pointwise (1 x 1) group convolutions, element-wise operations such as Addition, and varying channel width due to bottleneck structures in the network. Hence, ShuffleNetV2 included reforms to counter those issues to increase parallelism and decrease memory access cost by making efficient network architecture design choices such as including a channel splitting operation after each unit to split the feature channels into two branches c-c' and c' (where c' is c/2). Out of the two branches, one branch is kept as Identity, and the other branch is constituted of three convolutions with the same input and output channels to minimize MAC. These two branches are then concatenated followed by a channel shuffle operation (Fig. 3.2a). The structure for spatial-down sampling is shown in Fig 3.2b. These two structures which are then stacked make the overall ShuffleNetV2 CNN. The ShuffleNetV2 backbone integrated with YOLOv5s network increases parallelism in model training and decreases the MAC, which is a crucial parameter while training an object detection network and makes the model training much faster and more efficient. The ShuffleNetV2_YOLOv5s model consists of a total of 16 ShuffleNetV2 units. In the backbone region, the input image is first fed to a module consisting of 3 x 3 convolutions (kernels = 6, stride = 2, padding = 2), batch normalization, a maxpool layer, and activated by the SiLU activation function. The resulting tensor is inputted to four stacked ShuffleNetV2 units, followed by another eight and four stacked units. The first two units replaced the standard C3 modules present in the original network and feature a shortcut connection to the neck region of the network to prevent feature loss. The SPPF module is retained as it is faster than the original SPP module and efficiently aggregates all the features extracted by the ShuffleNetV2 units. The details of the reconstructed YOLOv5s backbone using efficient ShuffleNetV2 CNN architecture and a side-by-side comparison are shown in Fig. 3.3 and the resulting ShuffleNetV2_YOLOv5s model is shown in Fig. 3.4.

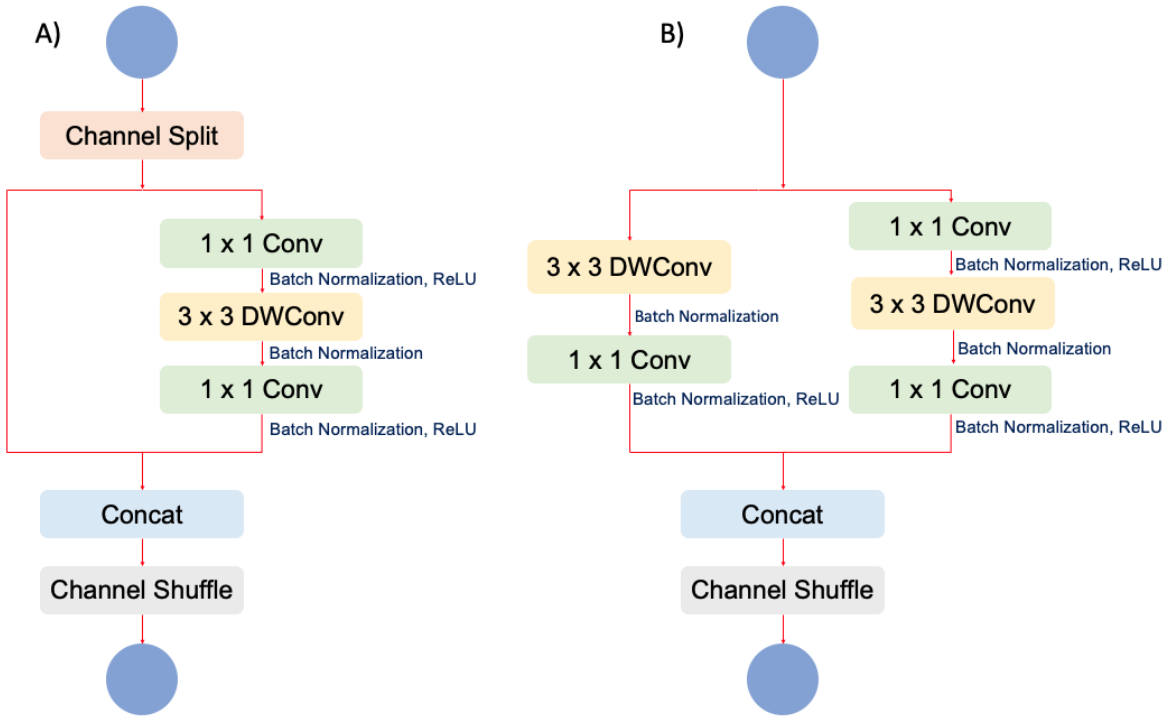


Fig. 3. 2. Architecture of ShuffleNetV2.

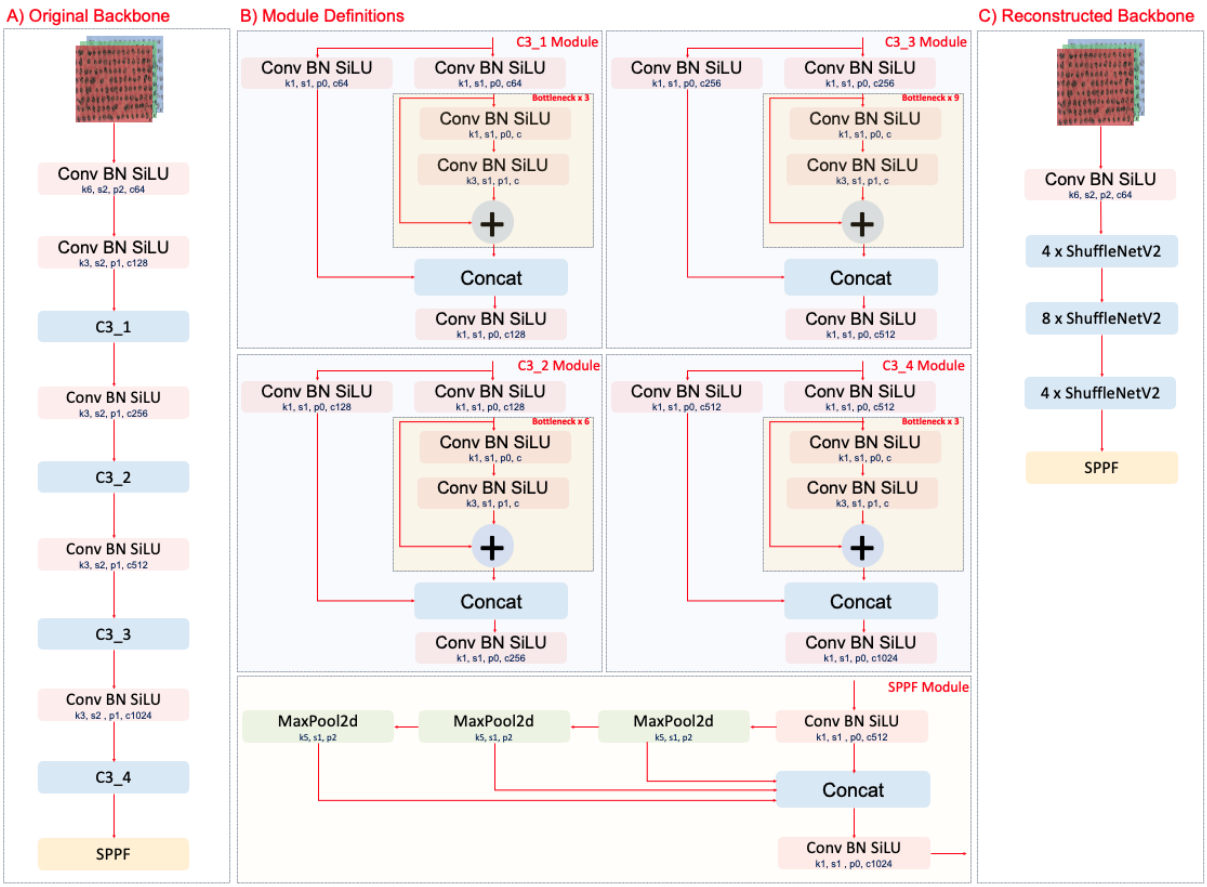


Fig. 3. 3. The comparison of the original and the reconstructed backbone architectures.

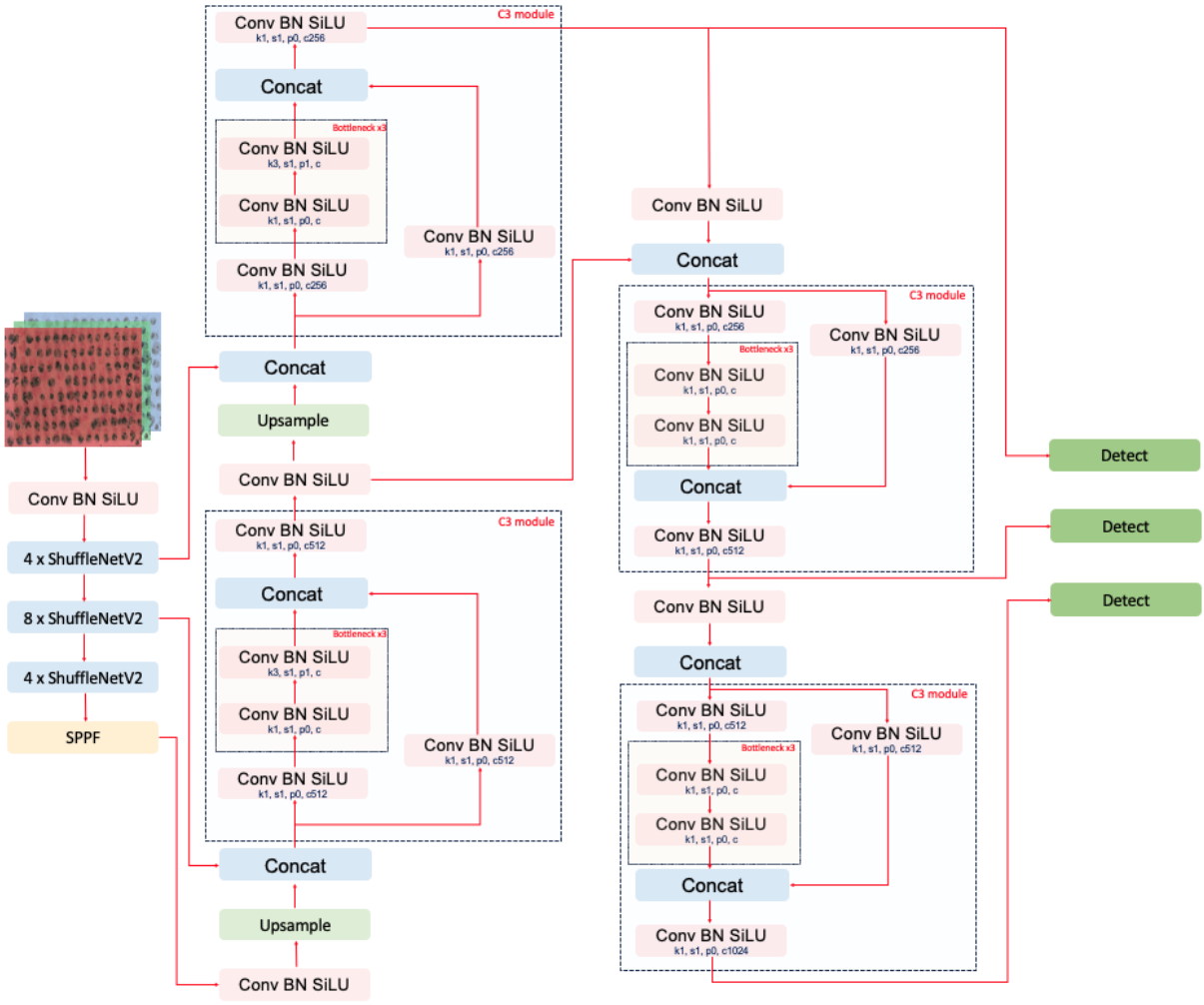


Fig. 3. 4. The structure of the ShuffleNetV2_YOLOv5s model.

3.3.6. Model Training and Performance Evaluation

All the models developed for this study were trained using the same sets of hyperparameters (Epochs: 100, Batch size: 8, Image Size: 640 x 640 px, Initial and Final learning rate: 0.01, Optimizer: Stochastic Gradient Descent, and Momentum: 0.937) using an NVIDIA Tesla T4 GPU. The performance of each model was evaluated based on the metrics of Precision (Eq. 1), Recall (Eq. 2), F1 Score (Eq. 3), mean Average Precision at 0.5 Intersection over Union (mAP@0.5IoU) (Eq. 4), and Inference speed (pre-process time + inference time + non-maximal suppression time) (Eq. 5).

$$Precision\ (P) = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad \text{(Eq. 1)}$$

$$Recall\ (R) = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad \text{(Eq. 2)}$$

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad \text{(Eq. 3)}$$

$$mAP = \frac{1}{2}\sum_{m=0}^{1}\int_{0}^{1} P_m(R_m)dR_m \quad \text{(Eq. 4)}$$

$$FPS = \frac{1}{mDT} = \frac{1}{\frac{1}{M}\sum_{i=1}^{M} t_i} \quad \text{(Eq. 5)}$$

3.3.7. Deployment in an Edge-AI Device and Integration with the Canola Crusher

The Jetson Nano (Jetpack 4.6.1, Ubuntu 18.02, Python 3.8.12, Torch v0.9.0, Torchvision v0.11.1), an edge-computing device developed by NVIDIA corporation was used as the platform for deploying the developed lightweight model. This device runs on 5 watts of power and features a Quad-core ARM A57 CPU clocked at 1.46 GHz and a 128 CUDA core Maxwell GPU. Once the training of the models was complete, the weight file of the best lightweight model (Fig. 3.9) was exported to *.onnx* (open neural network exchange) format from *.pt* (PyTorch) format and then transferred to the NVIDIA Jetson Nano. Exporting the model to *.onnx* format was conducted to optimize the inference speed of the network. The deployment was conducted to demonstrate the real-time performance of the developed model in accurately detecting the damages present in the canola samples. The embedded device was seamlessly incorporated into the canola crusher apparatus. This integration facilitated the following operational workflow: Canola kernels were systematically fed to the machine via the hopper and the crank was rotated to crush the seeds. The camera was fixed in such a way that it directly captured the incoming strip with the crushed kernels. This synchronous data acquisition enabled real-time monitoring of the canola damage detection process, with the detection boxes promptly visualized and displayed on a dedicated screen for immediate observation and analysis.

3.4. Results and Discussion

3.4.1. Comparison between methods of sample preparation and its effect on model performance

The two sample preparation methods using the traditional hand-held roller and the semi-automatic canola crushing unit were compared based on the quality of the sample set prepared and the time required to prepare the samples. It was observed that the seeds were more uniformly crushed and aligned using semi-automatic crushing unit, whereas the canola seeds crushed using the hand-held roller were touching each other and some seeds were not crushed properly. Fig. 3.5 shows the difference in time required to prepare 15 sets of the sample strips from which it is evident that the machine is the superior option as it allows to processing of more samples in a lesser duration of time and hence also exhibits lesser fatigue to the operator. Moreover, processing more samples in a shorter amount of time facilitated the preparation of more samples for evaluation, which led to a better representation of the actual damage condition of the canola in bulk.

Furthermore, the base YOLOv5s model was trained on the two datasets separately. The training performance of the same model on the two datasets is visualized in Fig. 3.6. The loss and mAP curves indicate that the model trained on D2 converged faster and with less noise. Fig. 3.7 outlines the results of the performance of the YOLOv5s model in both datasets. The F1 score and mAP@0.5 metrics of the YOLOv5s model trained on D1 were 1.7% and 1.4% lower than the model trained on D2. This result can be directly attributed to the better quality of the dataset, where the same model could learn the features inherent to both the classes better due to the kernels being better aligned, uniformity of the sizes of the kernels, and due to the non-touching nature of them. Our result is in accordance with the results from the study conducted by [7] where the performance of the YOLOv5 model trained by them in two distinct sets of datasets (touching and non-touching corn kernels) was better in the case of non-touching corn kernels. The samples crushed using the hand-held roller method resulted in some non-crushed kernels as well, which the baseline model confused as the heated class due to the close resemblance in appearance between the color of the heated seeds cotyledon and the dark brown shell. Moreover, the touching-kernels also made the annotation step difficult and had a huge chance of mislabelling. It was observed that while the ground-truth bounding boxes of the image corresponding to D2 were more distinct, the bounding

boxes for instances pertaining to D1 were overlapping between each class as well with the sound samples and the seed coat. The difficulty caused by annotation also led to a lower performance metric. Overall, this divergence in model performance can be attributed to multifaceted factors intrinsic to computer vision and deep learning. The D2 dataset, characterized by well-defined and prominently visible objects-of-interest, offered a more extensive and high-quality training data source. This enabled the YOLOv5s model to glean distinctive object features more effectively during training, ultimately enhancing its object recognition capabilities. In contrast, in D1, with objects exhibiting reduced clarity due to the non-uniform diameter of the crushed seeds, introduced inherent challenges into the training process, hampering the model's ability to extract pertinent information and make accurate predictions. As the baseline model performed better in the D2 dataset, the following model training with lightweight CNN backbones was done on this dataset. These multitude of reasons led to the model trained on D2 to detect the damaged kernels better as can be seen in Fig. 3.8. The D1 trained model reported a total of four false negative cases in a representative test image, one in the immature class and three in the heated class. All these misdetections were in overlapping seeds, which clearly emerges from the difficulty in the ground-truth preparation step and thus demonstrates the importance of proper data annotation in computer vision projects.

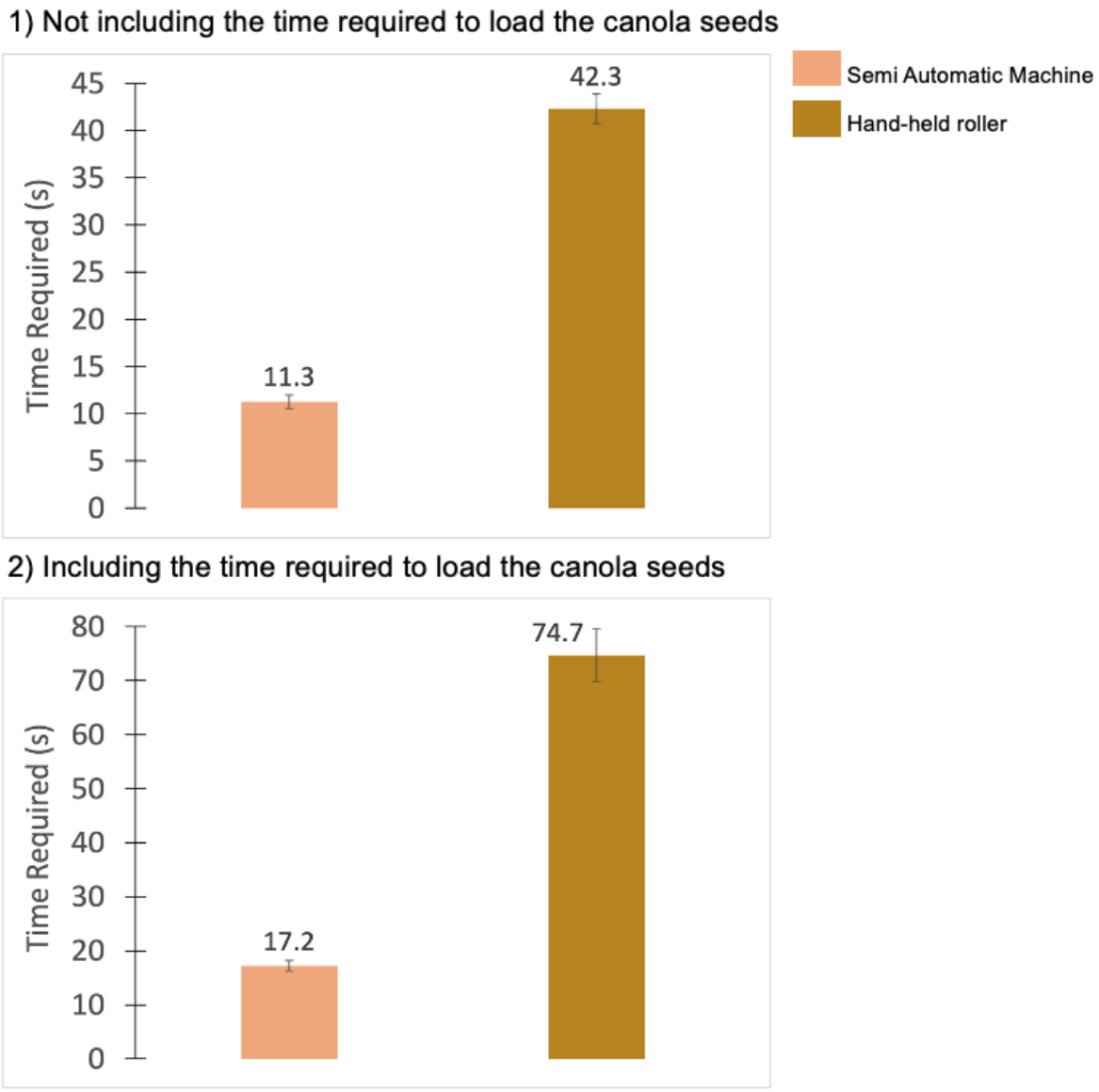


Fig. 3. 5. Comparison between the time required to prepare the crushed canola samples using hand-held roller and semi-automatic crushing machine, (A) not including the time required to load the canola seeds, (B) including the time required to load the canola seeds.

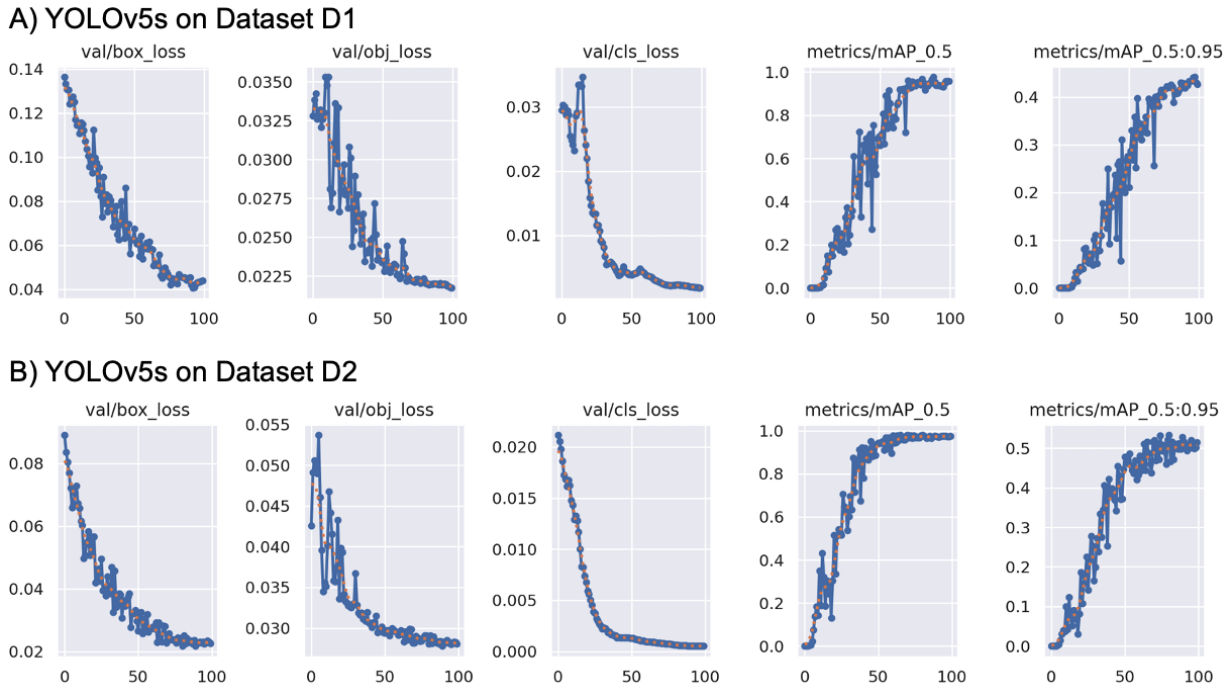


Fig. 3. 6. Difference in training metrics (Loss and mAP) for the YOLOv5s model trained on (A) dataset D1 and (B) dataset D2.

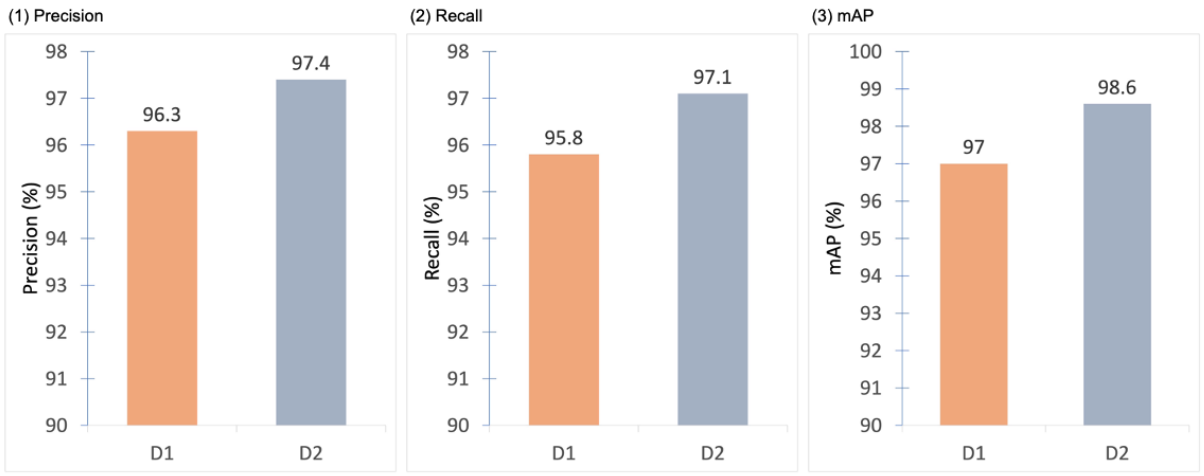


Fig. 3. 7. Performance of the base YOLOv5 model when trained on D1 and D2 datasets.

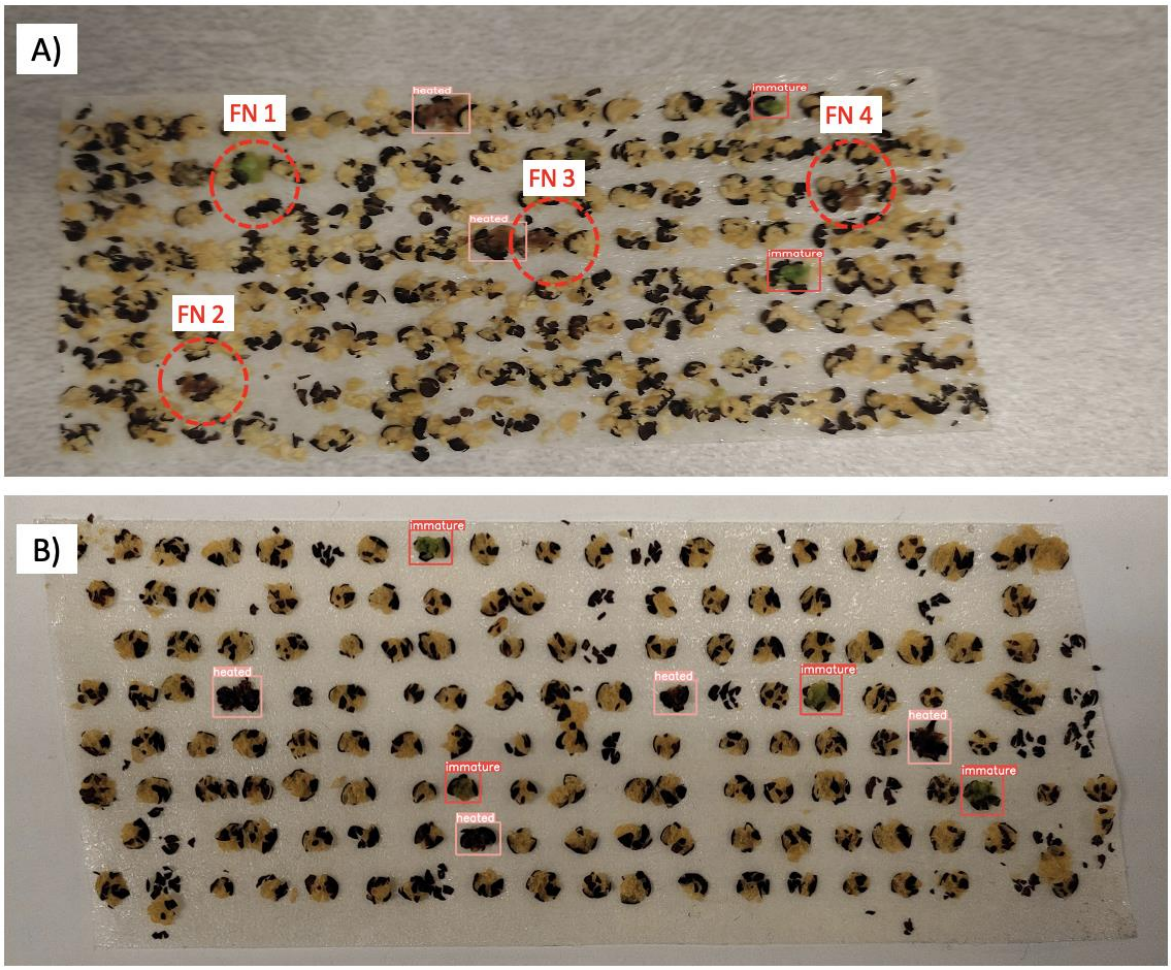


Fig. 3. 8. Detections made by the YOLOv5s model on a representative image present in A) test dataset D1, B) test dataset D2. FN stands for False Negative.

3.4.2. Comparison between the baseline model and the lightweight model

Fig. 3.9 outlines the difference in performance between the three models trained in the D2 dataset. The ShuffleNetV2_YOLOv5s model emerged as the lightweight model of choice as its size was 56.5% smaller than the MobileNetV3_YOLOv5s model and 80.4% smaller than the baseline YOLOv5s model. In terms of computational complexity, the YOLOv5s model based on the ShuffleNetV2 backbone was 64.1% smaller than the YOLOv5s model based on the MobileNetV3 backbone, and 99.5% smaller than the original model. Furthermore, this model had only 2,906,091 trainable parameters which was 56.7% and 82.95% less than the parameter counts

of MobileNetV3_YOLOv5s and the YOLOv5s models, respectively. In terms of inference speed, the FPS reported by the base YOLOv5s, ShuffleNetV2_YOLOv5s, and MobileNetV2_YOLOv5s models in a validation dataset in an NVIDIA Tesla T4 GPU were 56.1, 98, and 53.7 FPS respectively. However, due to the four-way trade-off between model size, model cost, inference speed, and overall accuracy, the light-network model-of-choice could not outperform the other two models in terms of mAP@0.5, which was 0.9% lower than the baseline model and 0.5% lower than the other lightweight model. In terms of F1 score, this model was the second-best model, only behind the original model, as it resulted in 0.4% lower than YOLOv5s but 0.9% higher than the other light-network option. However, it can be argued that this drop in both mAP and F1 scores compared to the original network can be neglected considering the significant reductions in the metrics of size and cost, which created a suitable condition for deployment in the resource-constrained embedded devices. Moreover, the mAP of 97.7% achieved by the ShuffleNetV2_YOLOv5s model is still higher than what an expert grader functioning under both fatigue and non-fatigue condition will achieve. The detection made by the three models on the same representative test image is visualized in Fig 3.10. The false negative case observed only on the image detected with the weights of the MobileNetV2_YOLOv5s model is due to the lowest F1 score achieved by this model. Hence, between the two lightweight network options, the ShuffleNetV2_YOLOv5s model emerged as the better model as it outperformed the MobileNetV3_YOLOv5s model in all the metrics except mAP.

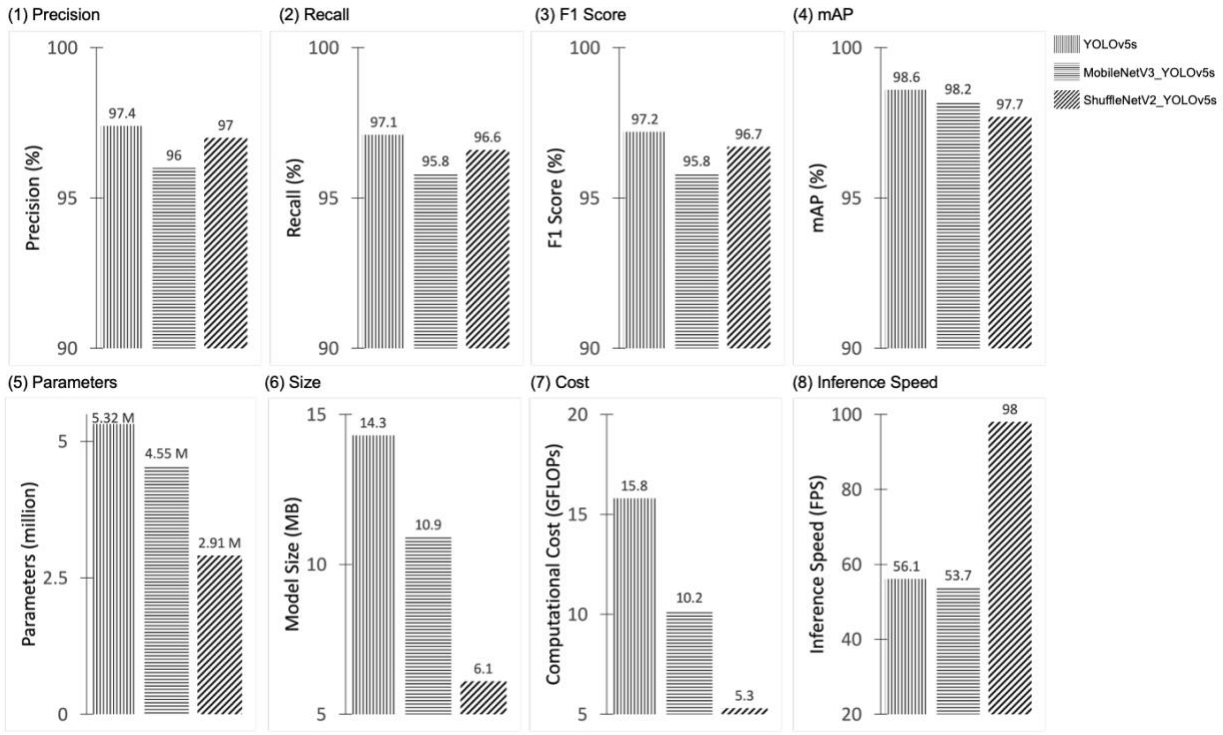


Fig. 3. 9. Comparison of lightweight models (MobileNetV3_YOLOv5s and ShuffleNetV2_YOLOv5s) with the Base YOLOv5s model.

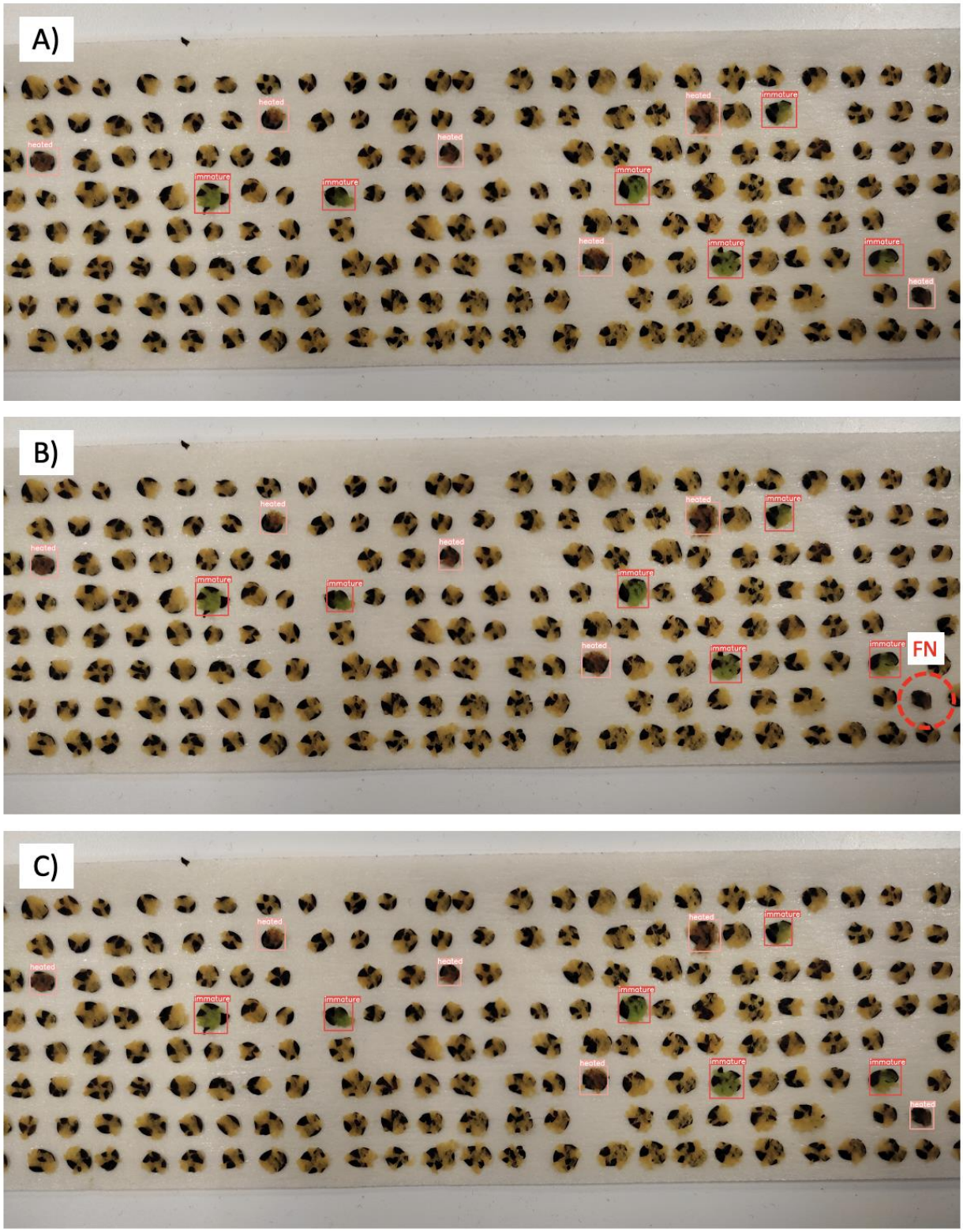

A)
B)
FN
C)

Fig. 3. 10. Detections made by (A) Base YOLOv5s, (B) MobileNetV3_YOLOv5s, and (C) ShuffleNetV2_YOLOv5s on the same representative test image.

3.4.3. Deployment results, hardware limitations, and benefits of inferencing on the edge

The overall system in operation, where no false positive as well as no false negative cases were observed in the real-time test videos captured by the camera onboard the device. This was in contrast to the model's performance on the validation dataset which can be attributed to the higher inference time required in the edge-device compared to the inference speed on the Tesla T4 GPU (98 FPS). Therefore, this higher detection speed in the Maxwell GPU resulted in better accuracy. The baseline YOLOv5s model with the CSPDarknet53 backbone displayed an inference speed of only 1.25 frames per second as it was the heaviest model amongst the three developed models compared to 5 frames per second by the ShuffleNetV2_YOLOv5s model. This four-times increase in FPS validates the network-compaction improvement made on the baseline model. The performance of the ShuffleNetV2_YOLOv5s model when deployed on the Jetson Nano device was also compared with the same models' performance when deployed on a Raspberry Pi 4 device. The inference speed achieved with Jetson Nano was 133.3% higher than the inference speed achieved with the Raspberry Pi 4 device as the former detected the damaged seeds with a speed of 5 FPS compared to only 1 FPS reported by the latter. This observed disparity in inference speed can be attributed to the inherent hardware distinctions between the two platforms. The Jetson Nano device is equipped with a dedicated Maxwell Graphics Processing Unit containing 128 CUDA cores, which is engineered to handle parallel processing tasks, particularly those prevalent in neural network computations. This parallelism is instrumental in accelerating matrix multiplication, convolution, and element-wise operations, which are fundamental to Convolutional networks. Furthermore, the Jetson Nano device augments its computational capabilities with a quad-core ARM Cortex-A57 CPU, which synergistically complements the GPU for AI workloads. The CPU can offload certain computational tasks and coordinate memory management efficiently, contributing to overall performance gains in inference tasks than the Raspberry Pi 4 device which lacks the depth of AI use case-specific hardware optimization, leading to relatively slower

inference times. However, it is crucial to mention that inference speed depends substantially on the type of hardware used, for instance, the best-performing lightweight model reported an FPS of 98 on a Tesla T4 GPU where it was trained, however, due to the relatively low-powered GPU onboard the Jetson Nano, the FPS was 5. The FPS can be increased by using more powerful edge devices such as the NVIDIA Jetson AGX Orin, but it will also increase the cost of the overall system significantly which might be unfeasible for farmers. For instance, the cost of the Jetson AGX Orin device is 16 times greater than the device used for this study. Nevertheless, the lightweight model developed in this study can be readily deployed in these expensive embedded devices for near real-time inferencing at the edge. Regardless of the detection performance of edge devices compared to the cloud GPUs or workstation GPUs, there are quite a number of benefits of inferencing on the edge as compared to sending the data to the cloud for processing. One of the primary benefits is the significantly lower latency achieved with edge computing, making it ideal for real-time applications such as the automated grading of agricultural products. Moreover, edge computing enhances privacy and security by keeping data on the device, reducing the risk of data breaches. It also conserves bandwidth by transmitting only relevant information, making it cost-effective and suitable for scenarios with intermittent internet connectivity.

3.4.4. Effect of data quality and data annotation on model performance

There is a direct correlation between the quality of the images, the quality of the annotations, the imaging conditions used for model training, and the performance of the model when trained under the same conditions. The results obtained from our study highlighted our observations in accordance with several other studies specifically conducted to check the effect of these factors relating to dataset quality on model performance. For instance, in a study conducted to study the effect of data annotation on model performance by [30], it was found that reannotating 5000 images present in the Google Open Images dataset significantly ameliorated the mean average precision metric. The reannotation process corrected a notable number of incorrect labels and other ambiguities hence refining the overall quality of the dataset. This strategy of improving the training data to improve model performance by fixing the model and improving the data either by reducing the inherent noise present in it or increasing the number of trainable instances is called Data-centric AI and is a rapidly growing route for developing cost-effective AI systems. Similarly,

[31] found that introducing both radial and uniform noises to the annotations made on the COCO2017 dataset negatively affects the mean average precision metric of the RCNN object detection model. Akin to the results observed in the two recent studies, the quality of data annotation on D2 was greatly superior to the quality of the data annotation on D1, and hence the same model's performance on the mAP metric was observed to be 1.4% better. Moreover, the effect of input image size was also studied and the performance of the YOLOv5s model is tabulated in Table 3.1. It can be observed that regardless of the dataset, the lowest input size image (i.e., 320 pixels) showed the most sub-optimal performance and suggests that training the model with inferior quality image data leads to sub-optimal performance in terms of both F1 score and mAP. The trends indicate that an empirical relationship exists between model performance and input image resolution for both datasets, however, no evidence was seen that the change in performance is proportionate to the image size.

Table. 3. 1. The Effect of Model Input Image Resolution on Model Performance for both the datasets.

Dataset	Image Size	P (%)	R (%)	F1 Score (%)	mAP@0.5 (%)
D1	320 x 320	94.8	94.8	94.8	95.8
	512 x 512	95.5	95.2	95.3	96.4
	640 x 640	96.3	95.8	96.1	97.0
D2	320 x 320	95.0	94.8	94.9	96.4
	512 x 512	96.2	96.4	96.3	98.2
	640 x 640	97.4	97.1	97.2	98.6

3.5. Conclusion

Quality of canola can be determined by the number of damaged (immature and heated) seeds. This is due to several factors such as frost during harvest and sub-optimal storage conditions which leads to enzyme inactivation. These damages directly affect the grade and the price in the market as the oil derived from it needs to be processed more, thereby reducing both the yield and

oil quality, and increasing the processing cost. Thus, detecting them is mandatory for setting the correct grade, which to date has been done via manual methods for both the sample preparation and damage seed identification steps. The present study provided a solution to detect damaged canola seeds using an end-to-end computer vision technique. A novel sample preparation technique in the form of a semi-automatic canola crusher machine was introduced, and its comparison with the traditional method shows that the former outperforms the latter both in the parameters of time required and quality of the prepared samples. The solution for automating the second component was framed as a deep learning-based object detection problem using the YOLOv5s model. However, as the final goal was to deploy the model into a resource-constrained environment for real-time inferencing, the original model was compacted to reduce its size and computational complexity. This was achieved by reconstructing the backbone with the ShuffleNetV2 CNN. The resulting model was a smaller and less computationally complex model suitable for embedded devices. This light model detected all the damages in a real-time video feed captured via the camera system onboard the Jetson Nano device with good accuracy and a detection speed of 5 FPS. Therefore, this study provided a groundwork for developing similar end-to-end frameworks for automating the damage detection process in other agricultural products.

References

[1] D. W. Chung, A. Pružinská, S. Hörtensteiner, and D. R. Ort, “The role of pheophorbide a oxygenase expression and activity in the canola green seed problem,” Plant Physiol, 142 (1) (2006), 88–97, doi: 10.1104/pp.106.084483.

[2] Z. Liu and S. Wang, “Broken Corn Detection Based on an Adjusted YOLO with Focal Loss,” IEEE Access, 7, (2019) 68281–68289, doi: 10.1109/ACCESS.2019.2916842.

[3] M. P. Mathew and T. Y. Mahesh, “Leaf-based disease detection in bell pepper plant using YOLO v5,” Signal Image Video Process, 16 (3) (2022), 841–847, doi: 10.1007/s11760-021-02024-y.

[4] W. X. Hu et al., “A method of citrus epidermis defects detection based on an improved YOLOv5,” Biosyst Eng, 227 (2023), 19–35, Mar. 2023, doi: 10.1016/J.BIOSYSTEMSENG.2023.01.018.

[5] J. Li, Y. Qiao, S. Liu, J. Zhang, Z. Yang, and M. Wang, “An improved YOLOv5-based vegetable disease detection method,” Comput Electron Agric, 202, (2022) 107345, doi: 10.1016/J.COMPAG.2022.107345.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” (2016). [Online]. Available: http://pjreddie.com/yolo/

[7] R. P. Thangaraj Sundaramurthy, Y. Balasubramanian, and M. Annamalai, “Real-time detection of Fusarium infection in moving corn grains using YOLOv5 object detection algorithm,” J Food Process Eng, 46 (9), (2023), doi: 10.1111/jfpe.14401.

[8] K. Sun, Y.-J. Zhang, S.-Y. Tong, M.-D. Tang, and C.-B. Wang, “Study on Rice Grain Mildewed Region Recognition Based on Microscopic Computer Vision and YOLO-v5 Model,” Foods, 11 (24) 4031 (2022), doi: 10.3390/foods11244031.

[9] K. Zhang et al., “Compacting Deep Neural Networks for Internet of Things: Methods and Applications,” (2021), doi: 10.1109/JIOT.2021.3063497.

[10] A. Thakuria and C. Erkinbaev, “Improving the Network Architecture of YOLOv7 to Achieve Real-Time Grading of Canola based on Kernel Health,” Smart Agricultural Technology, 5 (2023) 100300 doi: 10.1016/j.atech.2023.100300.

[11] S. Zhang et al., “Edge Device Detection of Tea Leaves with One Bud and Two Leaves Based on ShuffleNetv2-YOLOv5-Lite-E,” Agronomy, 13 92) (2023), doi: 10.3390/agronomy13020577.

[12] W. Xu et al., “A lightweight SSV2-YOLO based model for detection of sugarcane aphids in unstructured natural environments,” Comput Electron Agric, 211 (2023), doi: 10.1016/j.compag.2023.107961.

[13] F. Qi, Y. Wang, Z. Tang, and S. Chen, “Real-time and effective detection of agricultural pest using an improved YOLOv5 network,” J Real Time Image Process, 20 (2) (2023), doi: 10.1007/s11554-023-01264-0.

[14] L. Jia et al., “MobileNet-CA-YOLO: An Improved YOLOv7 Based on the MobileNetV3 and Attention Mechanism for Rice Pests and Diseases Detection,” Agriculture, 13 (7) (2023) 1285, doi: 10.3390/agriculture13071285.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” (2015). [Online]. Available: http://arxiv.

[16] R. Girshick, “Fast R-CNN,” (2015), [Online]. Available: http://arxiv.org/abs/1504.08083

[17] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” (2015), [Online]. Available: http://arxiv.org/abs/1506.01497

[18] T.-Y. Lin et al., “Microsoft COCO: Common Objects in Context,” (2014), [Online]. Available: http://arxiv.org/abs/1405.0312

[19] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” (2016), [Online]. Available: http://arxiv.org/abs/1612.08242

[20] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” (2018), [Online]. Available: http://arxiv.org/abs/1804.02767

[21] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” (2020), [Online]. Available: http://arxiv.org/abs/2004.10934

[22] G. Jocher et al., “ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference,” (2022), doi: 10.5281/ZENODO.6222936.

[23] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, “CSPNet: A New Backbone that can Enhance Learning Capability of CNN,” (2019), [Online]. Available: http://arxiv.org/abs/1911.11929

[24] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” (2014), doi: 10.1007/978-3-319-10578-9_23.

[25] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Scaled-YOLOv4: Scaling Cross Stage Partial Network,” (2021), [Online]. Available. http://arxiv:2011.08036

[26] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path Aggregation Network for Instance Segmentation,” (2018), [Online]. Available: http://arxiv.org/abs/1803.01534

[27] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” (2017), [Online]. Available: http://arxiv.org/abs/1707.01083

[28] A. G. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” (2017), [Online]. Available: http://arxiv.org/abs/1704.04861

[29] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design,” (2018), [Online]. Available: http://arxiv.org/abs/1807.11164

[30] J. Ma, Y. Ushiku, and M. Sagara “The Effect of Improving Annotation Quality on Object Detection Datasets: A Preliminary Study,” In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, New Orleans, LA, USA, 19–20 (2022), 4850–4859.

[31] C. Agnew, C. Eising, P. Denny, A. Scanlan, P. Van De Ven, and E. M. Grua, “Quantifying the Effects of Ground Truth Annotation Quality on Object Detection and Instance Segmentation Performance,” IEEE Access, 11 (2023) 25174–25188, 2023, doi: 10.1109/ACCESS.2023.3256723.

CHAPTER 4. Thesis summary and recommendations for future research

This thesis provided the groundwork to develop the necessary software required to develop an automatic canola damage identification and grading system. The proposed system is comprised of three major components: A faster canola crushing unit, a microprocessor to take in the video input and process the data, and an Artificial Intelligence model to analyze the data and output the decision in real-time. The primary objective is to improve the throughput of the grading step as the success of the successive steps in the canola processing supply-chain is dependent on this stage. Adoption of the fast-growing AI techniques will remodel the agricultural sector and allow to make reliable and informed decisions. The main contribution of this thesis to this project is creating the roadmap to develop state-of-the-art computer vision algorithms that are specific to detecting damages to canola kernels based on seed condition but are also flexible enough to be used for other seeds such as rice or wheat kernels, by updating the training data. Different network architectures were developed with the objective of optimizing the networks to negate the four-way trade-offs inherent to object detection network. It was have shown that using a semi-automatic canola crusher in place of a traditional handheld roller makes the overall system faster and the quality of the dataset produced by it, which is used for training the developed model is significantly better. Furthermore, it has been shown that GPU based microprocessor (e.g., NVIDIA Jetson Nano) is a better choice than CPU based microprocessor (e.g., Raspberry Pi 4) as the graphical processing units along with the CUDA software in the Jetson Nano is better at parallel computing. However, our report also suggests that the inference speed metric is directly related to the hardware of the edge-AI device. However, as more powerful embedded devices are exponentially expensive than entry-level devices, there is a need to find the sweet spot between speed and cost to achieve mass adoption of the system. Coming to the developed networks, which is the main highlight of the thesis, two object detection networks were developed, YOLOv7_ours and ShuffleNet_YOLOv5s. For the development of the two networks, two unique approaches were taken. In the first network, the metric of accuracy was prioritized, i.e., the main developmental goal was to improve the mean average precision metric while decreasing the metrics of size and computational cost. The addition of two Convolutional Block Attention Mechanism (CBAM) modules increased the accuracy compared to the baseline model while changing the convolutional operations with a more cost-

efficient ghost convolutional operation in all the ELAN modules in the SPPCSPC module decreased the overall size and cost of the model compared to the parent model. Two additional functionalities of tracking and counting were added to the network to automatically count all the detected damages. The use of the ByteTrack multi-object tracking algorithm decreased duplicate counts as all the identified objects were assigned a unique ID number. In the second network, which is based on the most extensively used YOLOv5 network, the main focus was on decreasing the metrics of size and cost and increasing the metric of inference speed. Due to the trade-off factor, the accuracy of this model was lower than the first network but much smaller, less computationally expensive, and much faster. Chapter 3 also reported that there is a direct correlation between dataset quality and model performance and reported that errors in the ground-truth preparation step negatively affect the results.

To overcome the limitations and to make continuous improvements to the networks, here are some recommendations for future studies.

1. To overcome the challenge of limited training data, Generated Adversarial Networks (GANs) or Denoising Diffusion Probabilistic Models (DDPMs) can be used to augment the size of the dataset. As the performance of the model is directly proportional to the amount of data it is trained on, it is hypothesized that this will lead to the creation of more accurate networks.
2. Research on developing object detection architectures utilizing Transformer models (such as the Vision Transformer (ViT) and Shifted Windows Transformers (SWIN-T)), which is a new paradigm that doesn't involve the convolutional operation. There is a rising number of studies which show that transformer-based models surpass CNN-based models in certain conditions.
3. To further improve the accuracy of the network, the effect of other attention mechanisms such as Squeeze and Excitation Networks (SENet) on model performance can be explored. Attention Mechanisms are useful when it is required to selectively focus on the most important features and suppress the non-important ones.
4. Finally, there exists substantial potential to advance the current prototype to a market-ready stage and initiate commercialization. By refining the prototype to include an automated canola crushing unit and integrating the developed object detection software, which processes canola kernels as inputs and autonomously delivers the grading results, we may

render the existing traditional approach obsolete. The versatility of this system extends beyond canola, making it transferable for other types of grains as well.

APPENDIX 1. Python code to create YOLOv7_Ours

A1.1. Network Architecture

(Note: Please refer to Fig. 2.4. for the graphical representation of the structure)

```
# parameters
nc: 2  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple

# anchors
anchors:
  - [12,16, 19,36, 40,28]  # P3/8
  - [36,75, 76,55, 72,146]  # P4/16
  - [142,110, 192,243, 459,401]  # P5/32

# yolov7 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [32, 3, 1]],  # 0
   [-1, 1, Conv, [64, 3, 2]],  # 1-P1/2
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [128, 3, 2]],  # 3-P2/4

   [-1, 1, GhostConv, [64, 1, 1]],
   [-2, 1, GhostConv, [64, 1, 1]],
   [-1, 1, GhostConv, [64, 3, 1]],
   [-1, 1, GhostConv, [64, 3, 1]],
   [-1, 1, GhostConv, [64, 3, 1]],
   [-1, 1, GhostConv, [64, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, GhostConv, [256, 1, 1]],  # 11

   [-1, 1, MP, []],
   [-1, 1, Conv, [128, 1, 1]],
   [-3, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [128, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 16-P3/8

   [-1, 1, GhostConv, [128, 1, 1]],
   [-2, 1, GhostConv, [128, 1, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, GhostConv, [512, 1, 1]],  # 24

   [-1, 1, MP, []],
```

```
   [-1, 1, Conv, [256, 1, 1]],
   [-3, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 29-P4/16

   [-1, 1, GhostConv, [256, 1, 1]],
   [-2, 1, GhostConv, [256, 1, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, GhostConv, [1024, 1, 1]],  # 37

   [-1, 1, MP, []],
   [-1, 1, Conv, [512, 1, 1]],
   [-3, 1, Conv, [512, 1, 1]],
   [-1, 1, Conv, [512, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 42-P5/32

   [-1, 1, GhostConv, [256, 1, 1]],
   [-2, 1, GhostConv, [256, 1, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [-1, 1, GhostConv, [256, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, GhostConv, [1024, 1, 1]],  # 50
  ]

# yolov7 head
head:
  [[-1, 1, GhostSPPCSPC, [512]], # 51

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [37, 1, Conv, [256, 1, 1]], # route backbone P4
   [-1, 1, CBAM, [256]],
   [[-1, -3], 1, Concat, [1]],

   [-1, 1, GhostConv, [256, 1, 1]],
   [-2, 1, GhostConv, [256, 1, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [-1, 1, GhostConv, [128, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, GhostConv, [256, 1, 1]], # 64

   [-1, 1, Conv, [128, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [24, 1, Conv, [128, 1, 1]], # route backbone P3
   [-1, 1, CBAM, [128]],
   [[-1, -3], 1, Concat, [1]],

   [-1, 1, GhostConv, [128, 1, 1]],
```

```
  [-2, 1, GhostConv, [128, 1, 1]],
  [-1, 1, GhostConv, [64, 3, 1]],
  [-1, 1, GhostConv, [64, 3, 1]],
  [-1, 1, GhostConv, [64, 3, 1]],
  [-1, 1, GhostConv, [64, 3, 1]],
  [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
  [-1, 1, GhostConv, [128, 1, 1]], # 77

  [-1, 1, MP, []],
  [-1, 1, Conv, [128, 1, 1]],
  [-3, 1, Conv, [128, 1, 1]],
  [-1, 1, Conv, [128, 3, 2]],
  [[-1, -3, 64], 1, Concat, [1]],

  [-1, 1, GhostConv, [256, 1, 1]],
  [-2, 1, GhostConv, [256, 1, 1]],
  [-1, 1, GhostConv, [128, 3, 1]],
  [-1, 1, GhostConv, [128, 3, 1]],
  [-1, 1, GhostConv, [128, 3, 1]],
  [-1, 1, GhostConv, [128, 3, 1]],
  [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
  [-1, 1, GhostConv, [256, 1, 1]], # 90

  [-1, 1, MP, []],
  [-1, 1, Conv, [256, 1, 1]],
  [-3, 1, Conv, [256, 1, 1]],
  [-1, 1, Conv, [256, 3, 2]],
  [[-1, -3, 51], 1, Concat, [1]],

  [-1, 1, GhostConv, [512, 1, 1]],
  [-2, 1, GhostConv, [512, 1, 1]],
  [-1, 1, GhostConv, [256, 3, 1]],
  [-1, 1, GhostConv, [256, 3, 1]],
  [-1, 1, GhostConv, [256, 3, 1]],
  [-1, 1, GhostConv, [256, 3, 1]],
  [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
  [-1, 1, GhostConv, [512, 1, 1]], # 103

  [77, 1, RepConv, [256, 3, 1]],
  [90, 1, RepConv, [512, 3, 1]],
  [103, 1, RepConv, [1024, 3, 1]],

  [[104,105,106], 1, IDetect, [nc, anchors]],   # Detect(P3, P4, P5)
 ]
```

A1.2. Python code to create a Convolutional Block Attention Mechanism (CBAM) module

```
class ChannelAttention(nn.Module):

  def __init__(self, in_planes, ratio=16):
    super().__init__()
```

```
    self.avg_pool = nn.AdaptiveAvgPool2d(1)
    self.max_pool = nn.AdaptiveMaxPool2d(1)
    self.f1 = nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False)
    self.relu = nn.ReLU()
    self.f2 = nn.Conv2d(in_planes // ratio, in_planes, 1, bias=False)
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    avg_out = self.f2(self.relu(self.f1(self.avg_pool(x))))
    max_out = self.f2(self.relu(self.f1(self.max_pool(x))))
    out = self.sigmoid(avg_out + max_out)
    return out

class SpatialAttention(nn.Module):

  def __init__(self, kernel_size=7):
    super().__init__()
    assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
    padding = 3 if kernel_size == 7 else 1
    self.conv = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    avg_out = torch.mean(x, dim=1, keepdim=True)
    max_out, _ = torch.max(x, dim=1, keepdim=True)
    x = torch.cat([avg_out, max_out], dim=1)
    x = self.conv(x)
    return self.sigmoid(x)

class CBAM(nn.Module):
  def __init__(self, c1, c2, ratio=16, kernel_size=7):
    super().__init__()
    self.channel_attention = ChannelAttention(c1, ratio)
    self.spatial_attention = SpatialAttention(kernel_size)

  def forward(self, x):
    out = self.channel_attention(x) * x
    out = self.spatial_attention(out) * out
    return out
```

A1.3. Python code to create a Ghost Convolutional network

```
class GhostConv(nn.Module):
    def __init__(self, c1, c2, k=1, s=1, g=1, act=True):  # ch_in, ch_out, kernel, stride, groups
        super(GhostConv, self).__init__()
        c_ = c2 // 2  # hidden channels
        self.cv1 = Conv(c1, c_, k, s, None, g, act)
        self.cv2 = Conv(c_, c_, 5, 1, None, c_, act)

    def forward(self, x):
        y = self.cv1(x)
        return torch.cat([y, self.cv2(y)], 1)
```

A1.4. Python code to create a standard convolutional (Conv) layer

```
class Conv(nn.Module):
    # Standard convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Conv, self).__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p), groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = nn.SiLU() if act is True else (act if isinstance(act, nn.Module) else nn.Identity())

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))

    def fuseforward(self, x):
        return self.act(self.conv(x))
```

A1.5. Python code to create the GhostSPPCSPC module

```
class SPPCSPC(nn.Module):
    def __init__(self, c1, c2, n=1, shortcut=False, g=1, e=0.5, k=(5, 9, 13)):
        super(SPPCSPC, self).__init__()
        c_ = int(2 * c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(c_, c_, 3, 1)
        self.cv4 = Conv(c_, c_, 1, 1)
        self.m = nn.ModuleList([nn.MaxPool2d(kernel_size=x, stride=1, padding=x // 2) for x in k])
        self.cv5 = Conv(4 * c_, c_, 1, 1)
        self.cv6 = Conv(c_, c_, 3, 1)
        self.cv7 = Conv(2 * c_, c2, 1, 1)

    def forward(self, x):
        x1 = self.cv4(self.cv3(self.cv1(x)))
        y1 = self.cv6(self.cv5(torch.cat([x1] + [m(x1) for m in self.m], 1)))
        y2 = self.cv2(x)
        return self.cv7(torch.cat((y1, y2), dim=1))

class GhostSPPCSPC(SPPCSPC):
    def __init__(self, c1, c2, n=1, shortcut=False, g=1, e=0.5, k=(5, 9, 13)):
        super().__init__(c1, c2, n, shortcut, g, e, k)
        c_ = int(2 * c2 * e)  # hidden channels
        self.cv1 = GhostConv(c1, c_, 1, 1)
        self.cv2 = GhostConv(c1, c_, 1, 1)
        self.cv3 = GhostConv(c_, c_, 3, 1)
        self.cv4 = GhostConv(c_, c_, 1, 1)
        self.cv5 = GhostConv(4 * c_, c_, 1, 1)
        self.cv6 = GhostConv(c_, c_, 3, 1)
        self.cv7 = GhostConv(2 * c_, c2, 1, 1)
```

A1.6. Python code to create a maxpool (MP) layer

```
class MP(nn.Module):
    def __init__(self, k=2):
        super(MP, self).__init__()
        self.m = nn.MaxPool2d(kernel_size=k, stride=k)

    def forward(self, x):
        return self.m(x)
```

APPENDIX 2. Python code to create ShuffleNetV2_YOLOv5s

A2.1. Network Architecture

(Note: Please refer to Fig. 3.4. for the graphical representation of the structure)

```
# Parameters
nc: 2  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [24]],  # 0-P1/2

   [-1, 4, shufflenetV2, [48]], # 1-p2/4

   [-1, 8, shufflenetV2, [96]], # 2

   [-1, 4, shufflenetV2, [192]], # 3

   [-1, 1, SPPF, [1024, 5]],
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]], # 5
   [-1, 1, nn.Upsample, [None, 2, 'nearest']], # 6
   [[-1, 2], 1, Concat, [1]],  # cat backbone P4 7
   [-1, 3, C3, [512, False]],  # 8

   [-1, 1, Conv, [256, 1, 1]], #9
   [-1, 1, nn.Upsample, [None, 2, 'nearest']], #10
   [[-1, 1], 1, Concat, [1]],  # cat backbone P3 11
   [-1, 3, C3, [256, False]],  # 12 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],  #13
   [[-1, 9], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 15 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]], #16
   [[-1, 5], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 18 (P5/32-large)

   [[12, 15, 18], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

A2.2. Python code to create a ShuffleNetV2 module

```
class shufflenetV2 (nn.Module):
    def __init__(self, c1, c2, n = 1):
        super().__init__()
        self.feature = []
        # print(len(self.feature), '*****')
        for i in range(n):
            if i == 0:
                self.feature.append(InvertedResidual(c1, c2, 2, 2))
            else:
                # print(c1, c2)
                self.feature.append(InvertedResidual(c2, c2, 1, 1))
        self.features = nn.Sequential(*self.feature)
    def forward(self, x):
        x = self.features(x)
        return x

class InvertedResidual(nn.Module):
    def __init__(self, inp, oup, stride, benchmodel):
        super().__init__()
        self.benchmodel = benchmodel
        self.stride = stride
        assert stride in [1, 2]

        oup_inc = oup//2

        if self.benchmodel == 1:
            #assert inp == oup_inc
            self.banch2 = nn.Sequential(
                # pw
                nn.Conv2d(oup_inc, oup_inc, 1, 1, 0, bias=False),
                nn.BatchNorm2d(oup_inc),
                nn.ReLU(inplace=True),
                # dw
                nn.Conv2d(oup_inc, oup_inc, 3, stride, 1, groups=oup_inc, bias=False),
                nn.BatchNorm2d(oup_inc),
                # pw-linear
                nn.Conv2d(oup_inc, oup_inc, 1, 1, 0, bias=False),
                nn.BatchNorm2d(oup_inc),
                nn.ReLU(inplace=True),
            )
        else:
            self.banch1 = nn.Sequential(
                # dw
                nn.Conv2d(inp, inp, 3, stride, 1, groups=inp, bias=False),
                nn.BatchNorm2d(inp),
                # pw-linear
                nn.Conv2d(inp, oup_inc, 1, 1, 0, bias=False),
                nn.BatchNorm2d(oup_inc),
                nn.ReLU(inplace=True),
            )

            self.banch2 = nn.Sequential(
```

```
            # pw
            nn.Conv2d(inp, oup_inc, 1, 1, 0, bias=False),
            nn.BatchNorm2d(oup_inc),
            nn.ReLU(inplace=True),
            # dw
            nn.Conv2d(oup_inc, oup_inc, 3, stride, 1, groups=oup_inc, bias=False),
            nn.BatchNorm2d(oup_inc),
            # pw-linear
            nn.Conv2d(oup_inc, oup_inc, 1, 1, 0, bias=False),
            nn.BatchNorm2d(oup_inc),
            nn.ReLU(inplace=True),
        )
    @staticmethod
    def _concat(x, out):
        # concatenate along channel axis
        return torch.cat((x, out), 1)

    def forward(self, x):
        if 1==self.benchmodel:
            x1 = x[:, :(x.shape[1]//2), :, :]
            x2 = x[:, (x.shape[1]//2):, :, :]
            out = self._concat(x1, self.banch2(x2))
        elif 2==self.benchmodel:
            out = self._concat(self.banch1(x), self.banch2(x))

        return channel_shuffle(out, 2)

def channel_shuffle(x, groups):
    batchsize, num_channels, height, width = x.data.size()

    channel_per_group = num_channels // groups

    x = x.view(batchsize, groups, channel_per_group, height, width)
    x = torch.transpose(x, 1, 2).contiguous()

    x = x.view(batchsize, -1, height, width)

    return x
```

A2.3. Python code to create a Convolutional layer

```
class Conv(nn.Module):
    # Standard convolution with args(ch_in, ch_out, kernel, stride, padding, groups, dilation, activation)
    default_act = nn.SiLU()  # default activation

    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, d=1, act=True):
        super().__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p, d), groups=g, dilation=d, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = self.default_act if act is True else act if isinstance(act, nn.Module) else nn.Identity()

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))
```

```
    def forward_fuse(self, x):
        return self.act(self.conv(x))
```

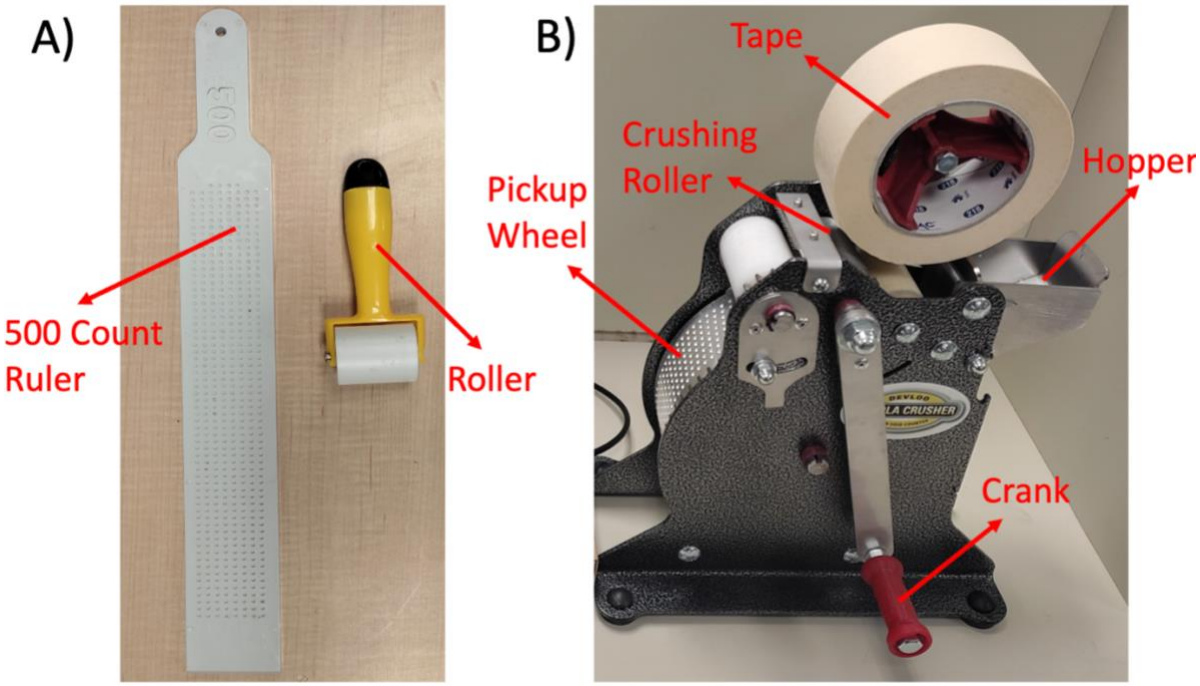
A2.4. Python code to create a Spatial Pyramid Pooling - Fast module

```
class SPPF(nn.Module):
    def __init__(self, c1, c2, k=5):  # equivalent to SPP(k=(5, 9, 13))
        super().__init__()
        c_ = c1 // 2  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c_ * 4, c2, 1, 1)
        self.m = nn.MaxPool2d(kernel_size=k, stride=1, padding=k // 2)

    def forward(self, x):
        x = self.cv1(x)
        with warnings.catch_warnings():
            warnings.simplefilter('ignore')  # suppress torch 1.9.0 max_pool2d() warning
            y1 = self.m(x)
            y2 = self.m(y1)
            return self.cv2(torch.cat((x, y1, y2, self.m(y2)), 1))
```

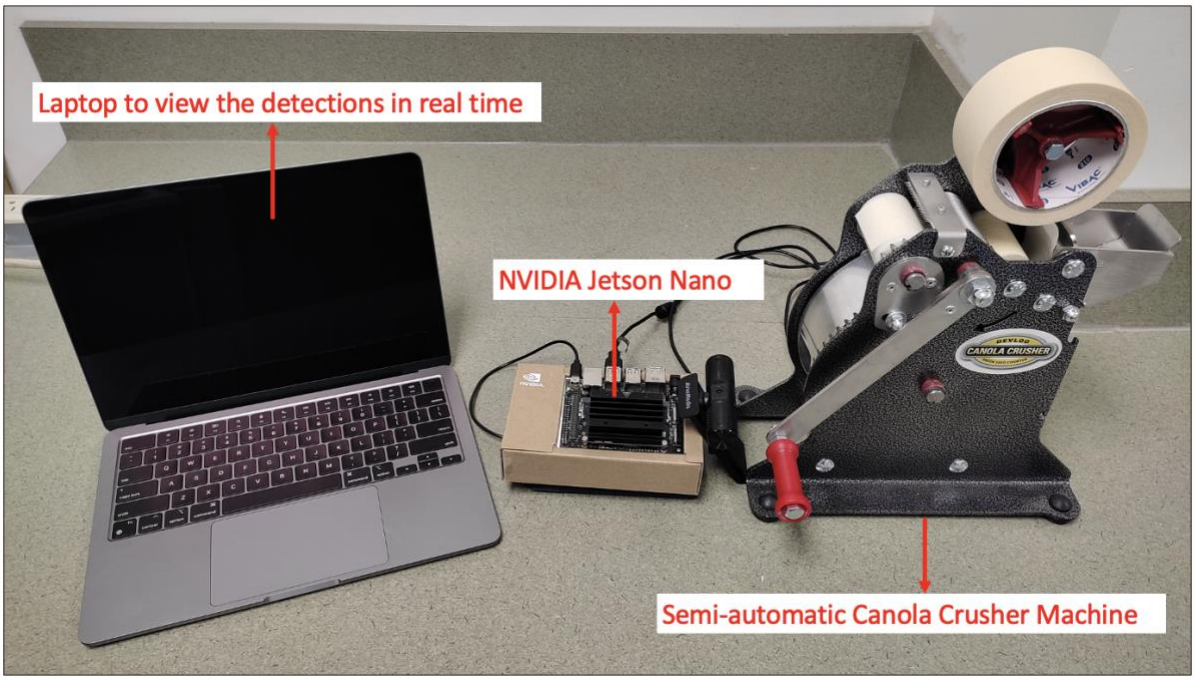
A2.5. Python code to create a Cross Stage Partial with 3 Convolutions (C3) module

```
class C3(nn.Module):
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(2 * c_, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.Sequential(*(Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)))

    def forward(self, x):
        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), 1))
```

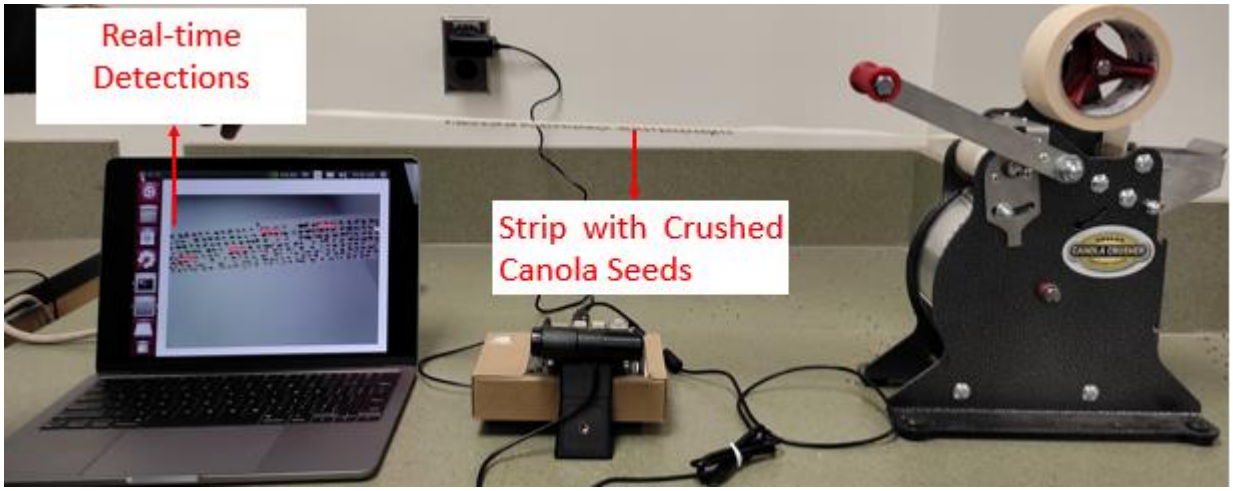
SUPPLEMENTARY FIGURES



SF 1. The two methods used for sample preparation. A) The traditional hand-held roller recommended by the Canadian Grain Commission to crush the canola seeds and detect the damages, B) The semi-automatic canola crusher machine (Devloo Canola Crusher) to crush the canola seeds faster and more uniformly.



SF 2. The overall system assembled to detect damages to canola kernels.



SF 3. The end-to-end computer vision system in operation.