Cocoa Beans Detection and Disease Classification Using Sparse YOLO

Razvan Stanciu
dept. EEMCS
University of Twente
r.c.stanciu@student.utwente.nl

David Galati
dept. EEMCS
University of Twente
d.galati@student.utwente.nl

Jeongyeon Huh
dept. EEMCS
University of Twente
j.huh@student.utwente.nl

Artur Dylewski
dept. EEMCS
University of Twente
a.dylewski@utwente.nl

Louis Daniël Lizarazo Fuentes
dept. EEMCS
University of Twente
l.d.lizarazofuentes@student.utwente.nl

I. Abstract

This research investigates the application of sparse training techniques to YOLOv11 for detecting diseases in cocoa beans. By reducing computational demands and optimizing performance in high-noise datasets, we aim to demonstrate how structured sparsification can maintain or even enhance prediction accuracy. Benchmarking against dense YOLOv11 models, this study evaluates the trade-offs in accuracy, model size, and computational efficiency.

II. Introduction

The Netherlands is one of the largest cocoa bean importers trading 20 to 25% globally [1]. The Netherlands is also one of the leading suppliers of cocoa bean products in Europe. After some processing, cocoa beans are exported more as a form of paste, butter, and powder than the cocoa bean imports [2]. the Dutch agriculture import and export industry of cocoa products relies on the stable supply of cocoa beans. Diseases on cocoa beans reduce cocoa bean production which raises concerns about an increase in import costs, fewer cocoa beans to process and make the product, and less product to import. As the majority of cocoa pods are collected and filtered by hand there remains large potential for improvement through automation.

Cocoa bean diseases can either affect the pods directly by rotting them or impact the plantation as a whole with a reduced quality. Among other diseases, Monilla and Phytophthora pose a major threat as they are responsible for 80% of losses in cocoa production [3]. The cocoa bean diseases are either detected with manual visual inspection or with the use of drones. The manual check is done on common symptoms such as discoloration, lesions, and fungal growth. However, it is time-consuming, inconsistent, prone to human errors, and can miss the early signs. Drones provide a scalable alternative for large-scale farms but the quality of images is not guaranteed. Addressing this challenge requires robust, accurate, and efficient disease detection.

Image recognition models are increasingly resource-intensive, posing challenges for their deployment in environments with limited computational power. Sparse training, which reduces the number of active parameters in a neural network, offers a promising solution. This study focuses on applying sparse training techniques to YOLOv11, a state-of-the-art object detection model, to optimize its performance in noisy datasets. Specifically, we explore its application in identifying cocoa bean diseases, building on prior research where dense models achieved 60% accuracy in high-noise scenarios.

III. Research Question

Can sparse training improve YOLOv11 accuracy prediction compared to a dense architecture of the network?

- Which method of sparsification yields the best results in terms of accuracy?
- What differences are observed regarding model speed in terms of inference time?
- What differences are observed regarding model size reduction after applying sparsification?

IV. Related Works

Sparse training is a growing area of interest due to its potential to reduce computational demands without compromising performance. However, limited studies have examined its application in high-noise settings or specifically with YOLOv11. By addressing this gap, this research could provide valuable insights into optimizing object detection models for practical applications.

Previous papers that have focused on YOLO sparse training emphasize the benefits of this architecture in terms of reduction in computational resources [4], [5]. However, none of these aim to test whether the sparse model can improve accuracy for predictions on noisy datasets. This idea was investigated before [6], and we draw our hypothesis from the outcomes of these paper, tested on different models for image and video prediction.

We found a particular application of YOLOv4 on a noisy image dataset with the aim of detecting cocoa beans and predicting their diseases. [7], [8]. However, the author only made use of a dense architecture for his research and managed to obtain 60% accuracy on the test dataset.

A recent study [9] introduced a feature sparsification approach in YOLOv11 through the Sparse Feature (SF) module, which replaces the Spatial Pyramid Pooling Fast (SPPF) module. This method improves small object detection by ranking channel importance and applying pruning to remove less significant features, thereby enhancing feature selection efficiency. While this approach is specifically designed for side-scan sonar (SSS) images, it demonstrates how sparsification techniques can be integrated into object detection models to improve performance without significantly increasing computational complexity.

V. YOLO and YOLOv11 Description

YOLO (You Only Look Once) is a family of object detection models designed for real-time applications. It is a single-shot detector that uses a convolutional neural network (CNN) to process an entire image in a single pass making predictions about the presence and location of objects in the image. YOLO performs predictions using a grid-based approach, where each grid cell is responsible for detecting objects.

YOLOv11 is an advanced version with additional improvements in accuracy and speed. The architecture builds upon the advancements of previous versions of YOLO. The three main components of the architecture are the backbone, neck, and head.

A. Backbone

The backbone extracts feature maps from input images. It contains the C3K2 block that manages feature extraction at different levels throughout the backbone, allowing the model to handle information propagation through the network by dividing the feature map and performing a succession of smaller kernel convolutions (3x3). It improves the feature extraction while reducing computational complexity, making the model more efficient.

B. Neck

Neck collects the features from different levels which can improve the multi-scale detection. The SPFF(Spatial Pyramid Pooling Fast) module is still used to pool features from various areas of an image using different sizes and multiple max-pooling operations to improve computational efficiency. YOLOv11 employs the C2PSA (Cross-Stage Partial Self-Attention) block as an attention mechanism, minimizing the repetitive computations across extracted features based on selected key area locations within an image.

C. Head

The head outputs bounding box coordinates, class probabilities, and objectness scores. Unlike the previous versions which had predefined anchor boxes, YOLOv11 evaluates objects with anchor-free detection which helps reduce complexity by simplifying the generation of bounding boxes.

VI. Pruning Methods Used

The PyTorch library was utilized for implementing sparsification as it had the required native pruning functions which could easily be run over PyTorch's Conv2d classes.

1) Structured Pruning: Channel-wise and filter-wise pruning to zero out entire channels or filters, ensuring efficient computation. The pruning criteria is based on L2-norm.

2) Unstructured Pruning: L1-norm based pruning to sparsify weights across all layers without maintaining a structured pattern.

VII. Dataset and Pre-trained Models

A. Dataset

1) Source: The dataset utilized in this project originates from prior research conducted by Serrano-Arenas et al. [8], which aimed to facilitate the detection of diseases in cocoa pods. The images were taken under standardized conditions to ensure consistency. The dataset was annotated using the LabelImg tool, enabling the differentiation between healthy and diseased cocoa pods through bounding box annotations. The annotations were stored in separate text files, each containing the class labels and corresponding bounding box coordinates for the cocoa pods in the images.

2) Distribution: Across all images there were 1591 objects, 77.6% healthy, 14.3% with phytophthora and 9.1% with monilia. The dataset is not balanced and many images contain a majority of healthy cocoa pods. This imbalance however did not pose a significant issue.

3) Data Challenges: The background is very noisy and cocoa pods are easily obstructed by other objects. The dataset included many images of cocoa trees in farm environments, which often times have occlusions. This is illustrated in Figure 3 where the healthy 0.56 cocoa pod partially obstructs the view to the monilia 0.25 pod behind it leading to reduced confidence for both. Furthermore trees, leaves and the image iteself might cut off cocoa pods (also visible at the top in 3). Additionally green and red cocoa pods easily blend into the environment's leaves while brown cocoa pods blend in with dry leaves while branches and shrubs can obstruct the view to lower growing cocoa pods. Dry or rotten twigs and branches can visually look like monilia which adds to the background's complexity.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

Fig. 1: Comparison of YOLOv11 model sizes, including the selected YOLOv11x model.

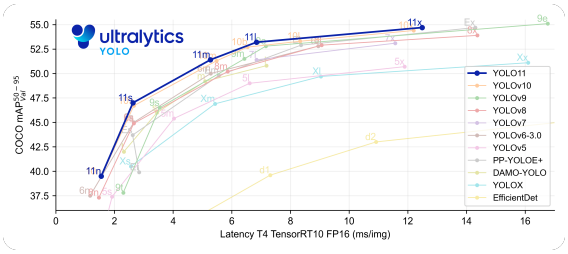


Fig. 2: Peformance comparison between YOLO models.

B. Pre-trained Models

The model employed in this study was initialized with pre-trained weights from the 2017 COCO dataset, a large-scale object detection dataset comprising 80 object categories and over 120,000 images. By leveraging pre-trained weights, the models benefit from prior feature extraction knowledge, thereby improving convergence speed and accuracy when training on the cocoa dataset.

For this study, the largest available YOLOv11 model, YOLOv11x is used. YOLOv11x offers a higher number of parameters and layers compared to smaller variants, making it particularly well suited for sparsification. Figures 1 and 2 provide an illustration of the differences between YOLOv11x and other variants of YOLOv11 in various metrics.

Experimental Framework

VIII. Metrics

In this study, we employed specific evaluation metrics to assess the performance of our sparsified YOLOv11 models in detecting cocoa bean diseases.

These metrics were chosen to provide a comprehensive understanding of the models' accuracy, efficiency, and robustness, aligning with our research objectives and addressing key research questions.

A. Mean Average Precision (mAP)

Mean Average Precision (mAP) is a standard metric in object detection that evaluates both the precision and recall of a model [10], [11]. It measures the average precision across all classes and is calculated by integrating the area under the precision-recall curve.

mAP@0.50 (mAP50): This metric calculates the average precision at an Intersection over Union (IoU) threshold of 0.50, meaning a predicted bounding box must overlap at least 50

mAP@0.50:0.95 (mAP50-95): This metric averages the precision over multiple IoU thresholds, ranging from 0.50 to 0.95 in increments of 0.05. It provides a stricter evaluation of detection accuracy by considering how well bounding boxes align with ground truth annotations across various thresholds.

The combination of mAP50 and mAP50-95 allows us to assess both general detection accuracy and precise localization, ensuring a comprehensive evaluation of model performance. This is crucial in determining the impact of sparsification techniques on detection quality.

B. Inference Time

Inference time refers to the duration the model takes to process an input image and generate a prediction. A reduction in inference time indicates improved computational efficiency, which is essential for real-time applications. This metric helps determine whether sparsification enhances model deployability in environments with limited computational resources.

C. Model Size

Model size denotes the total number of parameters within the neural network, directly impacting storage requirements and computational complexity. Evaluating model size helps determine how well sparsification reduces network complexity. A smaller model is advantageous for deployment on edge devices and in scenarios where memory efficiency is critical.

D. Connection to Research Questions

These metrics were selected to comprehensively address our research objectives:
Effectiveness of Sparsification Methods: By analyzing mAP scores, we assess the impact of sparsification on detection accuracy.
Impact on Computational Efficiency: Inference time and model size provide insights into the computational benefits of sparsification.
Performance in Noisy Environments: Robustness evaluation ensures the model remains effective under realistic conditions.

EXPERIMENTAL FRAMEWORK

E. Data preparation

Label data was modified to follow YOLO data requirements. Class and Bounding box coordinates need to be stored in .txt files in the following format: [class x_center y_center width height]. All box-related data was normalized.
For the purposes of this research, the dataset was reorganized. For model training and validation, the images were randomly split into training and validation sets beforehand. The final dataset configuration consisted of 250 images allocated for training and 62 images reserved for validation.

F. Models pruning and training

First, the dense pre-trained YOLOv11x model was trained on the cocoa diseases dataset. After that, the above mentioned pruning techniques have been applied, with different pruning amounts ranging between 10%-50%. Each pruned model has then been re-trained on the cocoa dataset with a lower learning rate and batch to ensure steady convergence. Sparse-aware training methods could not have been achieved due to reasons that will be discussed in section IX.

G. Validation

The trained model has been tested on the validation set for the performance metrics described in VIII. K-fold cross validation has not been applied due to resource intensive training process.

RESULTS

The evaluation of sparsification techniques on YOLOv11 revealed notable differences in performance across structured and unstructured pruning methods. Key findings are as follows:

H. Dense Model Performance

The YOLOv11 model was trained on a dataset of healthy and unhealthy cocoa beans, classifying them into three categories: Healthy, Monilia, and Phytophthora. In addition to classification, the model performs positional detection, evaluated using mean Average Precision (mAP).

The dense YOLOv11 model serves as the baseline for comparison against all pruned versions. It achieves strong results, demonstrating effectiveness in detecting cocoa bean diseases even in noisy datasets.

Model	mAP50	mAP50-95
YOLOv11 (Baseline)	0.7	0.65

TABLE I: YOLOv11 Dense Performance metrics

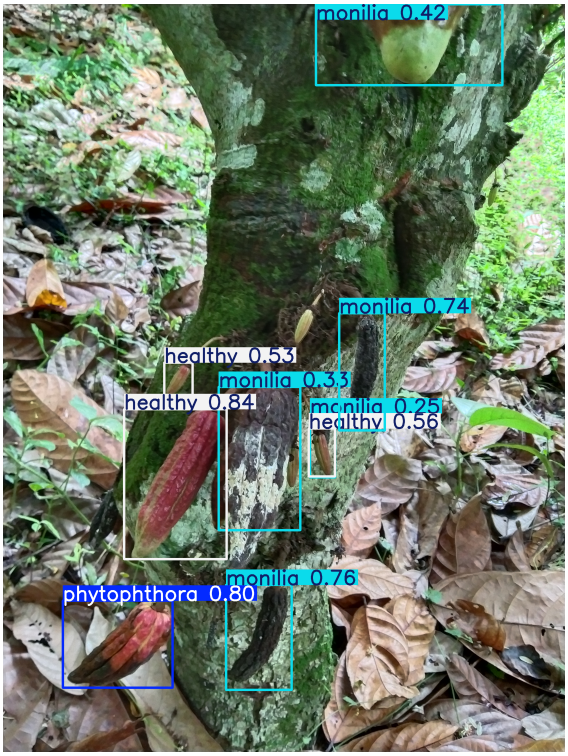


Fig. 3: Sample model Output from the validation set

I. Structured Chanel Pruning:

Channel-wise pruning demonstrated the most promising results:

- At 20% sparsity, it performed comparably to the dense model, with minimal degradation in both mAP50 and mAP50-95.
- Moderate pruning levels (30-50%) resulted in slight accuracy losses, but the models remained competitive, showing the viability of structured pruning for maintaining accuracy while reducing model complexity.

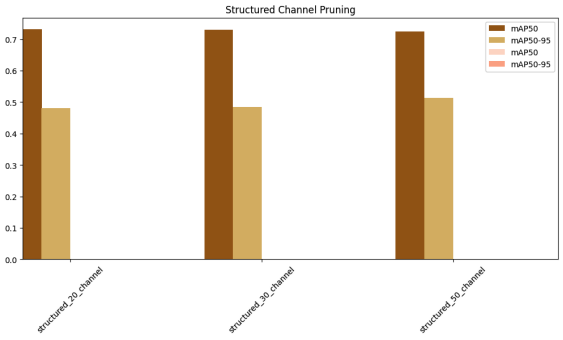


Fig. 4: Structured Channel pruning

Structured Filter (neuron) Pruning:

Filter-wise structured pruning was less effective than channel-wise pruning but still outperformed unstructured methods.

J. Unstructured Pruning:

Unstructured pruning consistently underperformed compared to structured methods, particularly at higher pruning levels (e.g., 40-50%). Performance degradation was most evident in **mAP50-95**, highlighting the challenge of maintaining precision at higher IoU thresholds when sparsity is unstructured. Despite its simplicity, unstructured sparsification failed to deliver robust accuracy while introducing sparsity into the network.

K. Best Performers:

We found the top performers from each method : Structured Channel Pruning with a 20% pruning

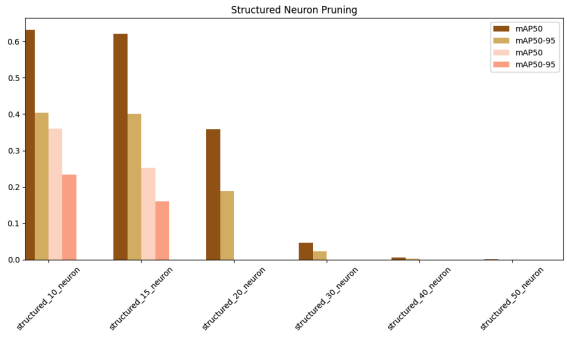


Fig. 5: Structured Neuron pruning

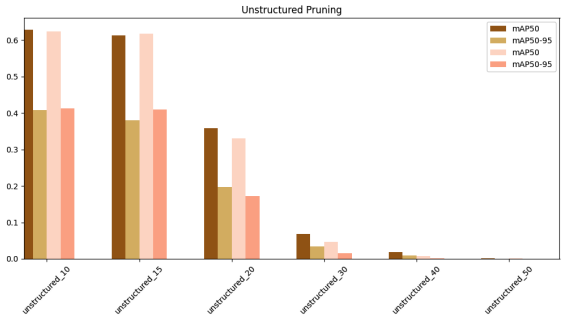


Fig. 6: Unstructured

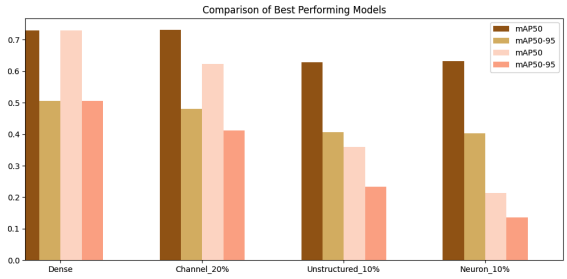


Fig. 7: Top 3 models versus dense model

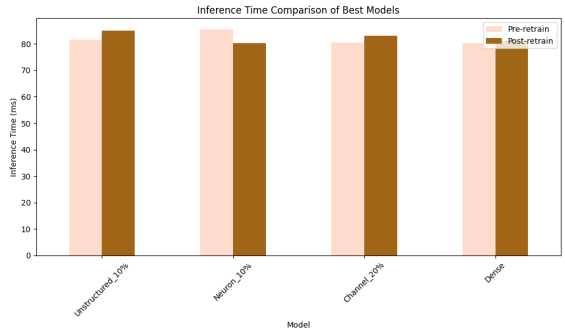


Fig. 8: Inference comparison (pruned models are retrained)

rate, Unstructured Pruning with a 10% pruning rate, and Structured Neuron Pruning with a 10% pruning rate. Those were compared against the Dense model as seen in Figure 7. Additionally, we measured their inference times, as shown in Figure 8. The best-performing models are highlighted in green, indicating that there is no significant loss in performance. Structured pruning methods unsurprisingly have lower general inference times than models which used unstructured pruning. Structured pruning removes entire channels or filters which would reduce the number of operations used for inference. However, the PyTorch framework is optimized for standard model sizes which in this case could results in less efficient batching and using less optimized computation kernels for all pruning methods used.

Furthermore the classification performance was validated by the confusion matrices which can bee seen in Figures 9b,9a,9c,9d. The matrices indicate that all models generally correctly classify more than 60% of the diseases but miss lot of the healthy cocoa pods.

The dense model exhibits a more gradual decline in F1 as confidence increases, suggesting better stability and generalization. On the other hand the pruned models show a steeper drop-off beyond a confidence threshold of 0.8, indicating a higher sensitivity to confidence calibration. The dense model reaches its peak F1 at a confidence of approximately 0.55, whereas the pruned models require a higher confidence threshold (0.59–0.74). Pruning may introduce a bias toward higher-confidence predictions based on these observations. See Figures 10a, 10b, 10c, 10d.

IX. Discussion/Future Work

A. Limitations of Current Sparsification Methods

Existing sparsification techniques in libraries such as PyTorch.prune do not support complete pruning through the permanent removal of less useful weights. Truly sparse implementations could

be achieved, however, in the following ways:

- restructuring the whole model architecture and obtain smaller tensors by removing the zeroed out weights;
- implement sparse matrix operations;

We have investigated both methods, but were unsuccessful in applying any of them. They both had a high level of complexity and required deep understanding of the underlying modular structure of YOLO with its implementation in PyTorch. Since we used a pre-trained model and made use of a user-friendly API to experiment with YOLO on our dataset, none of us were capable of achieving a truly sparse configuration of the model.

There was, however, a third possibility to emulate a sparse implementation and make inferences in size and speed improvements. This option involved using techniques that would maintain the pruned (zeroed-out) weights fixed during the fine-tuning process. As mentioned in our experimental framework, the re-training (fine-tuning) process after pruning the weights will set those weights back to non-zero values during the backpropagation process. We essentially end up with the same dense networks, but with different, perhaps slightly optimized weight values. There are three options which we attempted to deploy at this step in order to maintain sparsity of our network. The methods are described table II, detailing their effects on the network:

The first method resulted in very poor performance of the new network. This method was applying the same mask (before fine-tuning) on an essentially new model (after fine-tuning). The initially pruned weights that got updated in the process might have gained now significantly higher importance than before. Pruning them again meant losing important information of the network.

The other two methods were the only ones that could have worked. Both of them required to access and modify specific steps in the training loop of the YOLO algorithm. To do that, we tried to implement a custom training loop by accessing the underlying PyTorch implementation of YOLOv11. We were, however, unsuccessful in managing to make it work. Several issues related to the inner-workings of the YOLO API data format requirements made the task impossible to achieve, combined with our poor understanding of PyTorch. This method would have allowed us to achieve sparsity aware training, obtaining a fine-tuned model whose pruned weights would be preserved. This, in turn would have uncovered valuable insights for our research project.

Due to these limitations, both the inference speed and model size do not exhibit significant improvements, as the underlying computational structure remains largely unchanged.

B. Complete Pruning

This project has been somewhat affected by the methodology and current technological limitations discussed below. However, retrying this study with improved methods—such as fully pruning or locking down the sparsified weights—could enhance its merit and usefulness. This approach may also lead to a reduction in neural network size and potentially impact the computational requirements of the presented models.

C. Lottery ticket hypothesis and iterative pruning

Our study is based on the lottery ticket hypothesis, as stated in [12]. The paper suggests furthermore that iterative pruning techniques which gradually reduce weights across multiple rounds lead to better results than one-shot pruning. This would be an interesting future experimentation area for aiming to improve the accuracy on a sparsified YOLO model. However, this idea could not have been implemented due to issues mentioned above (IX-A).

X. Conclusions

This study demonstrated that structured pruning, particularly **channel-wise pruning**, is the most effective sparsification method for YOLOv11 in high-noise datasets. The results highlight key insights:

Structured Pruning Efficiency: Channel-wise pruning maintained accuracy comparable to the dense model at lower sparsity levels (e.g., 20%). This method is preferable because it creates regular patterns throughout the model which comes in handy for GPU architectures.

Unstructured Pruning Limitations: Unstructured pruning resulted in substantial accuracy loss,

Step	Method	Effect
After retraining	Reapply mask to weights	Ensures exact sparsity preservation of the same weights
During loss computation	L1-regularization	Encourages sparsity dynamically by penalizing large weights
During optimizer step	Zero out gradients of pruned weights	Prevents updates to pruned weights

TABLE II: Options for sparse-aware training

particularly in stringent evaluation metrics like mAP50-95. This could be due to the uneven distribution of sparsification throughout the network.

Balance Between Accuracy and Efficiency: Structured pruning offers a practical approach to optimize YOLOv11 for resource-constrained environments, enabling efficient deployment without sacrificing performance.

These results establish a foundation for advancing the deployment of sparsified YOLO models in applications that require robust performance in noisy and resource-limited settings.

XI. Reflection

This study explored the impact of sparsification techniques on the performance of YOLOv11 for cocoa bean disease detection. The findings provided valuable insights into the trade-offs between model accuracy, computational efficiency, and real-world applicability.

A. Key Insights

The results demonstrated that structured pruning, particularly channel-wise pruning, maintained accuracy close to that of the dense model while significantly reducing model complexity. This indicates that sparsification can be a viable method for optimizing object detection models in constrained environments. However, unstructured pruning resulted in notable accuracy degradation, suggesting that removing individual weights without preserving structured representations leads to suboptimal performance.

B. Challenges and Limitations

One of the primary challenges encountered in this study was the lack of full support for sparse convolutional operations in PyTorch. Although pruning successfully reduced the number of active parameters, the underlying computation graph remained largely unchanged, preventing substantial speedup gains. Future research should explore frameworks optimized for sparse computations to fully leverage the benefits of sparsification.

Another limitation was the dataset itself. The presence of noise, occlusions, and class imbalances introduced difficulties in achieving high detection performance. While the results demonstrated improvements over prior dense models, further dataset expansion and augmentation techniques could enhance generalization.

Additionally, post-pruning retraining was required to recover accuracy losses from sparsification. This additional step increased computational costs and complexity, suggesting that more refined sparsification techniques—such as dynamic sparse training—could be beneficial.

Overall, this study highlights the potential of structured sparsification for object detection tasks but also underscores the need for continued research in optimizing sparsity-aware architectures for practical deployment.

References

[1] [Online]. Available: https://www.idhsustainabletrade.com/uploaded/2022/12/DISCO-Report_161222_final.pdf

[2] "The Dutch market potential for cocoa | CBI." [Online]. Available: https://www.cbi.eu/market-information/cocoa-cocoa-products/netherlands/market-potential

[3] D. B. Vera, B. Oviedo, W. C. Casanova, and C. Zambrano-Vega, "Deep Learning-Based Computational Model for Disease Identification in Cocoa Pods (Theobroma cacao L.)," Jan. 2024, arXiv:2401.01247 [cs]. [Online]. Available: http://arxiv.org/abs/2401.01247

[4] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020, arXiv:2004.10934 [cs]. [Online]. Available: http://arxiv.org/abs/2004.10934

[5] J. Zhang, P. Wang, Z. Zhao, and F. Su, "Pruned-YOLO: Learning Efficient Object Detector Using Model Pruning," in *Artificial Neural Networks and Machine Learning – ICANN 2021*, I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds. Cham: Springer International Publishing, 2021, pp. 34–45.

[6] B. Wu, Q. Xiao, S. Wang, N. Strisciuglio, M. Pechenizkiy, M. v. Keulen, D. C. Mocanu, and E. Mocanu, "Dynamic Sparse Training versus Dense Training: The Unexpected Winner in Image Corruption Robustness," Oct. 2024, arXiv:2410.03030 [cs]. [Online]. Available: http://arxiv.org/abs/2410.03030

[7] https://datasetninja.com/cocoa diseases, "Cocoa Diseases." [Online]. Available: https://datasetninja.com/cocoa-diseases
[8] J. S. Serrano Arenas and C. A. Torres Villamizar, "Prototipo de aplicación móvil para la identificación de mazorcas de cacao enfermas haciendo uso de visión por computadora y aprendizaje de máquina," 2020, accepted: 2021-05-25T15:21:48Z Publisher: Universidad Autónoma de Bucaramanga UNAB. [Online]. Available: https://repository.unab.edu.co/handle/20.500.12749/13367
[9] "Side-Scan Sonar Small Objects Detection Based on Improved YOLOv11." [Online]. Available: https://www.mdpi.com/2077-1312/13/1/162
[10] J. Hui, "mAP (mean Average Precision) for Object Detection," Apr. 2019. [Online]. Available: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173
[11] Ultralytics, "YOLO Performance Metrics." [Online]. Available: https://docs.ultralytics.com/guides/yolo-performance-metrics
[12] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," Mar. 2019, arXiv:1803.03635 [cs]. [Online]. Available: http://arxiv.org/abs/1803.03635
[13] ADylewski, "ADylewski/ML2," Jan. 2025, original-date: 2024-11-28T14:31:57Z. [Online]. Available: https://github.com/ADylewski/ML2

XII. Appendix

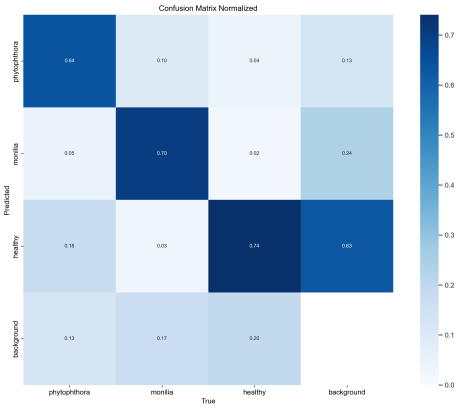
A. Code

The code used for this assignment can be found in our GitHub Repository [13]

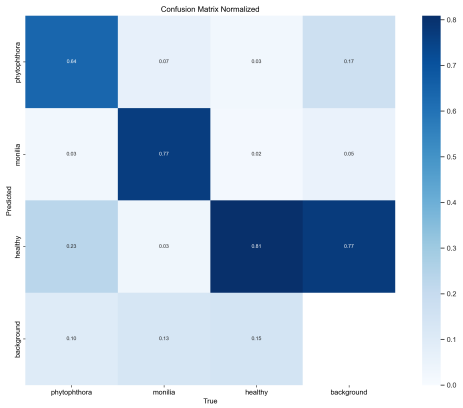
B. AI Tools

During the preparation of this work, the group used ChatGPT to aid with writing the code, clarifying ideas, and performing grammar checks. After using this tool/service, we thoroughly reviewed and edited the content as needed, taking full responsibility for the final outcome.

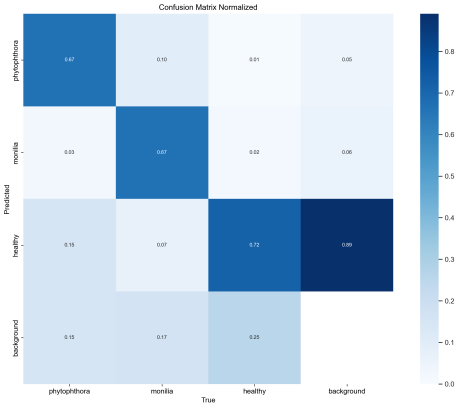
C. Best models confusion matrices and F1 confidence curves



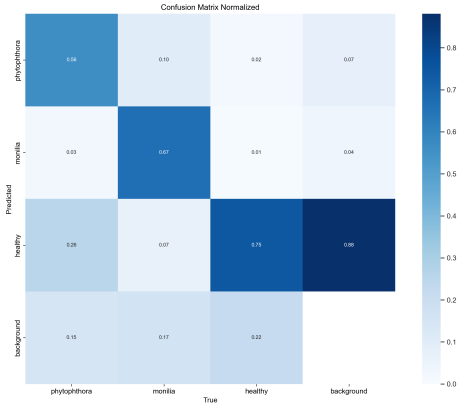
(a) Dense model confusion matrix.



(b) Retrained channel-wise structured pruning 20% sparsity confusion matrix.

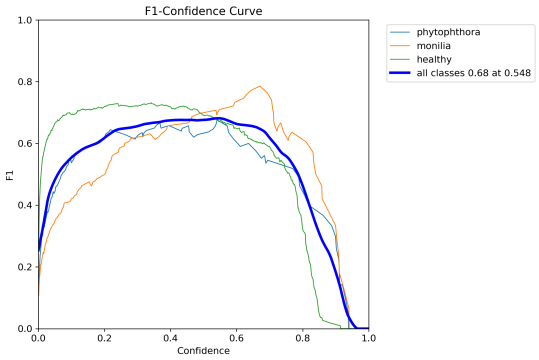


(c) Retrained unstructured pruning 10% sparsity confusion matrix.

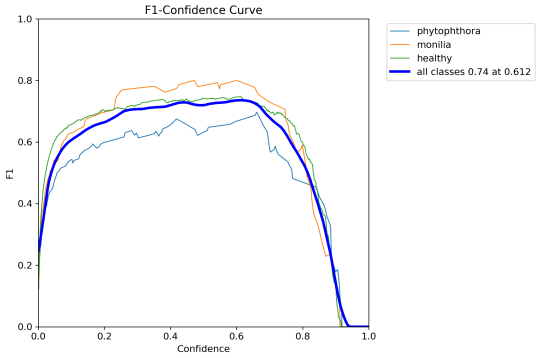


(d) Retrained neuron-wise structured pruning 10% sparsity confusion matrix.

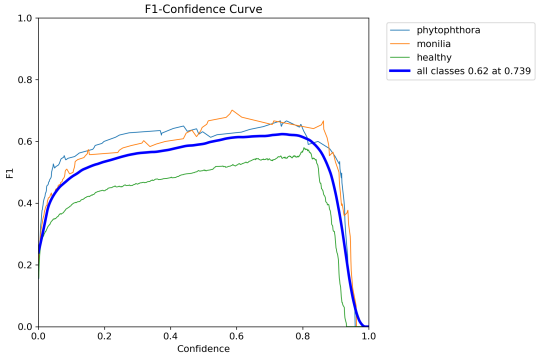
Fig. 9: Confusion matrices of the best performing models vs dense.



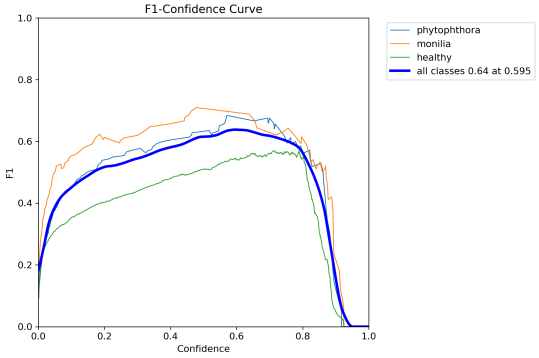
(a) Dense model F1 curve.



(b) Retrained channel-wise structured pruning 20% sparsity F1 curve.



(c) Retrained unstructured pruning 10% sparsity F1 curve.



(d) Retrained neuron-wise structured pruning 10% sparsity F1 curve.

Fig. 10: F1 confidence curves of the best performing models vs dense.