Real-Time Plant Disease Detection

by

Ahmed AbdElKhalek,

Asmaa Qindeel,

Mahmoud Basha

A thesis submitted to the

School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

December 2022

Abstract

In this thesis, we discuss how to train a plant disease detection model and how the process of data annotation can affect the model's robustness against real-environment images. We do this by presenting the results of training YOLOv5 and YOLOv7 over apple and bell pepper crops. We would like to show that unlike previous work with YOLO's older version -which mostly use the same dataset (the PlantVillage dataset)- our experiment shows that by changing the annotation method of the PlantVillage dataset, we can achieve high mAP accuracy, not only that, we also show that the model is not overfitted. The model we built can perform well on data with complex environment.

potential of YOLOv4 for plant disease detection

Acknowledgments

In the name of Allah, the Most Gracious, the Most Merciful. Alhamdulillah, all praise belongs to Almighty Allah, the Lord of the worlds. And prayers and peace be upon Muhammad His servant and messenger. The writers would like to thank Allah for His blessing in the completion of this thesis and for sending the right people in our way to help us deliver.

Contents

List of Figures

Chapter 1

Introduction

Welcome to the smart world, computers are every where, in every hand, which became a great source of change. Let us put this massive computing force to use.

1.1 Motivation

According to the UN projections the world is heading for its peak population around the ending of this century of a number around 10.4 billion people. This is a valuable information which tells us that now is the time to improve, to build for a better future, and our field of choice is agriculture as it will be a challenging production for that huge number of population. We will use AI to improve the current methods of farming, leading to an easier, more efficient, machine-dependant process, helping farmers everywhere to secure better yields by eliminating the minimal tasks, and automating them to be done by machines. One of these tasks is the plant disease detection, it is a repeated minimal work that takes up a lot of time and attention, hence, automating this task will help secure the crops and reduce labour cost.

1.2 Problem

The technical problem is annotating the dataset. The annotation process takes hours to complete on a single class, and as we will discuss, there will be times where we will have to re-annotate the dataset. But let's take a step back and discuss why this task is important? To understand this we illustrate the steps of the task; First of all, the crops must be monitored daily -this includes huge farms, monitoring all the trees is necessary because diseases are contagious and one tree could be the carrier of the entire yield's black death. Moreover, when changes are detected in the leaves -by an expert of course- appropriate control measures should be taken at the right time to secure the rest of the yield. Each disease is different, and requires different treatments, carelessness in this process will lead to wasting time and money. Remember, we are talking about large-scale farming for huge population here, mistakes which can cost us the yield may lead to price uprising or famine at worst case scenario. So as you can see, this requires a lot of labour work, that can be easily and more efficiently replaced by machines. And this is when AI comes in handy, it can handle **the repetitive task that can bear no error.**

1.3 Objective

Our goal is to build a plant disease detection system using YOLO(you only look once) the famous object detection model. We examine the performance of some versions of YOLO such as version four, five, and seven.

1.4 Contributions

In order of tasks, Asmaa was responsible for the theory and research task, Ahmed handled data preparation and model training, and Mahmoud implemented and delivered the final output as a web app.

1.5 Organization of Thesis

This paper will take you into details of our experiment going through the steps in order as we achieved them; first the data processing and the right annotation we used, then model training and different models performance and comparison in the results section after which we include other's work in the same area, then we represent our output which is the website and discuss how to improve this work in the future.

Chapter 2

Background

Real-time plant disease detection like all other detection projects, consists of two main components, a collection of enough data of the object you want to detect for model training and the model architecture. The data in our case is pictures of the plant and its disease at hand. Such data needs to be labeled correctly, and for leaf images the labels may include the indentation of the lesion, as in the following figure:

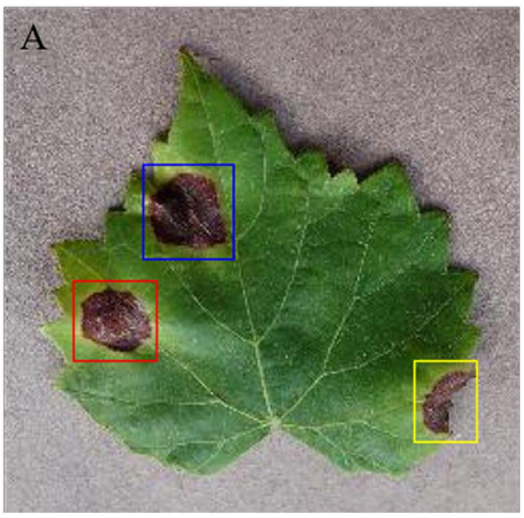


Figure 2.1: Annotation of the Grape Leaf. (A) Annotated image. (B) XML file fragment of Black rot disease.

The main problem in thid component is annotating the dataset. The annotation process takes hours to complete on a single class, and as we will discuss and there may be times where we have to re-annotate the dataset.

Another problem is obtaining the images in the first place. Most leaf lesions are rare to spot, specially in early stages. Moreover, the lab environment photos affect the model differently than the real environment images.

The second component is the model design, how to do the detection, there are many methods like SIFT [6], Haar[11], and Convolutional Features which is the most promising and is used in most of YOLO's versions.

Real-time plant disease detection is a work-in-progress AI task, mostly because it needs to be divided into smaller milestones, each milestone represents one plant. Researchers usually train one model per plant at a time. Hence this project is not a first of a kind as an idea. There are many published papers that have done similar work and here we will mention a few of them, the closest to our approach..

2.1 Conventional Methods

At the early 2010's, there were many tries for plant disease detection models using different methodologies such as principal component analysis and backpropagation networks[16] or K-means clustering to segment disease spots[14]. Then, after the Alex Net and VGG was published, Convolutional Neural Nets was the go-to in image detection, and started the race of variant models/architectures of CNN.

One example of a deep CNN model is the work done by Mishra et al., [9] to detect Corn diseases in India, which was released in the ICCIDS 2019, and had accuaracy of 88%. Their model also used Plant village data set and was trained then optimized

to be deployed into raspberry pi 3, giving it the Real-Time inference advantage. In other words, individuals can have this system working on a simple standalone device right in their hands.

The same task was built for the grape plant in China by Xiaoyue Xie et al., 2020[17]. The grape disease diagnosis is of high cost and high risk of error. So, they proposed a real-time solution and introduced the Faster DR-IACNN architecture, with precision of 81% mAP. Their architecture proved effective in accurately detecting four common grape leaf diseases.

2.2 YOLO : You Only Look Once

YOLO[13], is a state-of-the-art, real-time object detection model based on CNN. Its first version was a big success in 2016 due to its speed in real-time detection..

"..We frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance."[13].

YOLO's architecture, fig: 2.2, has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers.

Its key strength, alongside speed, is generalizability, meaning it can adapt to new or unexpected inputs, this is why it is used to detect traffic signals, people, parking

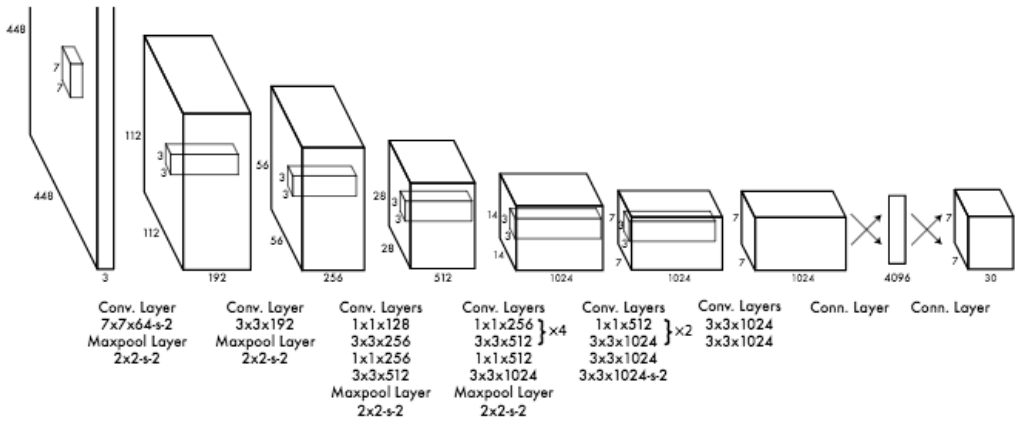


Figure 2.2: YOLOv1 Architecture.

meters, and animals.

After the first version of YOLO was released in 2016, there has been five more versions ever since, indicating how much promising is the algorithm. And regarding our task, YOLO has been utilized in the real-time plant disease detection as we'll see.

Chapter 3

Methodology

3.1 Data Set

3.1.1 Initial Dataset

The dataset that we looked at, and used with YOLOv4 is the PlantVillage dataset. The difficulty we faced while searching for a dataset that perfectly captures the details of plant leaves made choosing the PlantVillage dataset an obvious choice as it provides very detailed pictures of diseased leaves. The dataset consists of 55,180 images. The images are divided into 4 classes for each crop. There is always a class of the healthy crop, and other classes representing different diseases. Images are resized to 256x256 and are also reduced to a horizontal, and vertical resolution of 96 dpi. The setting of the environment for all images was the same in terms of background and lighting. All of the images were snapped having a gray/concrete background. Every image displays a single leaf taken from a particular crop. This paper examines the apple crop and the bell pepper dataset. The apple crop dataset consists of 3 different classes healthy, black rot and scab.

Characteristics of different diseases that affect the apple crop

(a) Black rot can appear as brown to black concentric rings on the leaves.

(b) Scab can appear as circular spots that are light brown.

(a)
Scab

(b)
Black
Rot

Figure 3.1: Images of the apple leaf diseases

3.1.2 Data Augmentation

Bag of Freebies is a set of techniques that increase the accuracy of the object detector without increasing the inference cost by changing the training strategy or the training cost, as mentioned in the "YOLOv4 optimal speed and accuracy of object detection"[2]. The techniques are: Data Augmentation, Semantic Distribution Bias in Datasets, and Objective Function of BBox Regression. The most significant strategy of them is data augmentation. The main benefit of augmentation is to generalize the dataset for the model, so that the model can detect images captured in different external environment and settings.

There are two data augmentation methods that we employ in this project:

(1) Photometric Distortions: is the process of changing the brightens, contrast, hue, saturation, or adding some noise to the image.

(2) Geometric Distortions: is the process of scaling up, or down the image, cropping, flipping, or rotating the image.

Augmentation Framework: We used the website Roboflow. In Roboflow you insert the images and select which augmentation methods you want to use, and Roboflow takes care of the rest.

3.1.3 Augmentation Methods

An issue came up because all the images shooting positions are the same, the model will not learn the variety of positions the leaf can be in, our proposed solution for this problem is to simulate the different orientation of the leaves by flipping and rotating the images(Geometric Distortions). Flipping is performed by randomly flipping the images horizontally or vertically. Rotation is performed by randomly selecting a rotation angle between 0 to 45 degress then rotating the images with the selected angle to obtain new viewpoint.

(a) (b) (c)

Figure 3.2: image with different angles

Second issue is that all the images in PlantVillage dataset are captured with the same

lighting, which does not reflect real life scenarios. In a real-life scenario an image will be taken at daytime, noon, dawn, or at night. Thus, we have to simulate these lighting conditions by applying brightness adjustment, and saturation adjustment to the images randomly. In brightness adjustment we randomly increase or decrease the brightness of the image by 25 percent. In saturation adjustment we randomly increase or decrease the saturation of the image by 25 percent to simulate different random events that occur to camera lenses when capturing photos in different lighting environment.

(a) (b) (c)

Figure 3.3: image with different brightness

Third Issue is that the images in the dataset were taken in a clean, and controlled environment. The leaves were picked and placed clearly in front of the camera. In a real environment the conditions won't be so perfect, there might be some dust or dirt on the leaf or the camera lens. Thus, we have to accommodate these different image capturing settings when training the model. Our approach to accommodate this imperfect conditions is to apply salt and pepper noise to 25 percent of the images.

(a) (b) (c)

Figure 3.4: image with different noise applied to it

Another Problem with the dataset is that every image contains only a single object which is a leaf in the center of the image. This of course is not a real-life scenario; in a real-life scenario an image can have multiple leaves and branches. In a real-life scenario the leaf object in the captured image can be blurred and out of focus. To solve this issue, we apply blur effect on some of the images making some of the leaves appear out of focus.

(a) (b) (c)

Figure 3.5: image with different blur applied to it

3.1.4 Annotation

Annotation Tool: the tool we used to annotate the images is LabelImg. LabelImg is an annotation tool that was created by Tzutalin. It is written in python and uses QT graphical interface.

LabelImg is an easy-to-use tool you only need to input the directory of the folder in which the images reside, drag a bounding box around the segment of the image that affiliates this particular image to a particular class then select the class which the image belongs to.

The output of the annotation process is a text file that has 5 values. The first value is a label encoded number representing the class (i.e. 0 for black rot, 1 for scab). The second and the third values are the x, and y coordinate of the bounding box. The fourth and the fifth values are the width, and height of the bounding box. These values aren't represented as is, these values are divided by the height and the width of the image. This division is done to normalize the values to be in the range of 0 and 1. The normalization is essential for the model training to divert any unwanted biases. Every new text line in the output text file represents a bounding box. First annotation approach in our first try we decided to draw the bounding box on the whole leaf as in [1] paper. The issue of this approach is that there is only one leaf in an image.

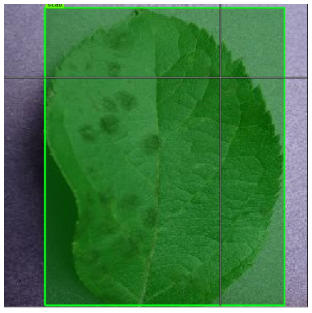


Figure 3.6: Single Annotation

Result of first annotation approach the model learned that in any image there will be only one leaf, so when the model is tested with images that in complex environment - an environment that resemble real life - the model won't display the best performance, as test image would have multiple leaves overlapped on each other, while there is some out of focus in the background of the main leaf of attention. The model draws one bounding box mixing multiple leaves that have different diseases into one box. The model displays a big overfitting problem. The model also can't classify a leaf that has multiple diseases.

Second Annotation approach In our second try we tried to annotate the affected areas of the leaf instead of the whole leaf. This should encourage the model to learn to draw multiple bounding boxes, and that there could be multiple diseases on a leaf.

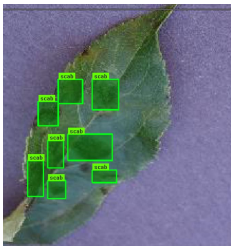


Figure 3.7: Multi-class Annotation

3.2 YOLOv4 Architecture

3.2.1 History of YOLO

There are two state-of-the-art object detection architectures; Two-Stage, and One-Stage architectures. YOLO belongs to the One-Stage architecture, i.e., when the input image is fed to the network it outputs directly the class probabilities and the bounding boxes. In the two-stage architecture, the first part is a region proposal network, where it calculates the probability that an area has an object. Although YOLO traded off accuracy by making the architecture One-Stage, it gained faster inference time. The first ever one-stage detector was YOLO, it was created by Joseph Redmon et al. in 2016[4]. He published his paper detailing how a one-stage detector can outperform two-stage detectors in inference time. He approached the problem of object detection as a regression problem.

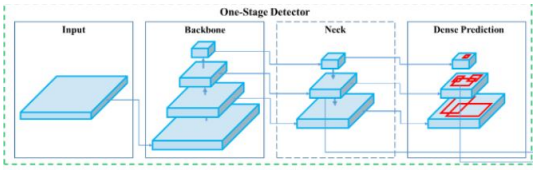


Figure 3.8: One-Stage object detector architecture

3.2.2 History of YOLOv4

In 2018, Redmon and Farhadi published a paper called "YOLOv3: an incremental improvement"[5], where they proposed a new architecture, which uses DarkNet53 network as the backbone. The DarkNet53 is much faster and bigger than previous network in YOLOv1. DarkNet framework is a high performance, open source framework for implementing neural network written in C and Cuda. In April of 2020,

Bochkovsky et al. published YOLOv4[2] on his githup. Instead of using DarkNet53 he used CSPDarknet53.

3.2.3 Architecture

YOLOv4 architecture is split into three parts: Backbone, Neck, and Head.
Backbone uses the CSPDarknet50 network according to the "YOLOv4: Optimal Speed and Accuracy of Object Detection"[2]. CSP stands for Cross Stage Partial connection. Because of the amount of the layers, the last layer has less and less information and context than the first layers. So to solve this problem the network uses skip connections to back-propagate to the first layers without loosing much of the context. To achieve the skip connection, DenseNet Is used, but this will produce a lot of parameters, which make it harder to train and inference. To combat this problem CSPDarknet53 replaces The DenseNet so that it copies and send one copy of the feature map through the dense block and another feature map straight on to the next stage, which solves the computational bottlenecks.

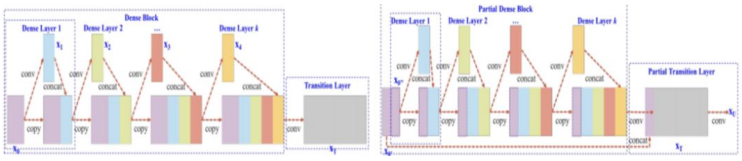


Figure 3.9: change in the dense architecture of DenseBlock

The neck is the features aggregation part. It does this by collecting the feature maps from the backbone part, then mixes them and combines them to be ready for the next stage.
The Neck consists of two parts: SPP(Spatial Pyramid Pooling) block, and PANet(Path

Aggregation Network). SPP block is used to increase the respective field and separate most important features form the backbone. In YOLOv4 it divides the features along the depth dimension, then applying SPP on each part, and then aggregate it to produce the output feature map.

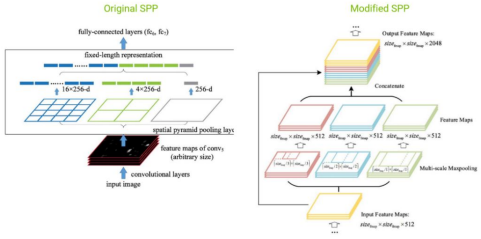


Figure 3.10: Modified SPP Architecture

PANet: the network is used to improve the process of the instance segmentation by keeping the spatial information, which improves the performance of localization.

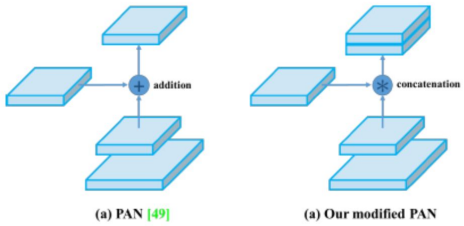


Figure 3.11: PANet Architecture architecture

Head: is the most important part in classification. It is where the bounding box is drawn and classified. Grid of SxS is placed on the input image. Each cell will produce an n-dimensional vector called 'y'. Its parameters are pc, bx, by, bh and bw. Pc is the confidence level that there is an object in the cell that belongs to certain class. Bx and By are the offset from the top left corner of image. Bh is the height of

the bounding box and Bw is the width of the bounding box.

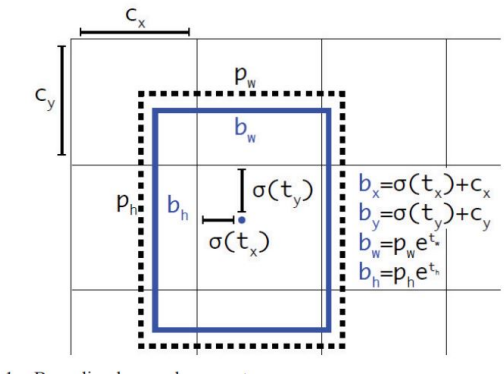


Figure 3.12: Head classification process

Chapter 4

Results

4.0.1 Evaluation Metrics

The metrics used are F1 score, mAP [8], precision and recall. F1 score is the harmonic mean of precision and recall [3]. The highest possible value of F1 score is 1, which indicates perfect precision and recall, where the lowest possible score is 0, which indicates either the precision or recall is zero.

$$F1score = 2 * \frac{(precision * recall)}{precision + recall}$$

Figure 4.1: F1-score equation

mAP is the average mean of precision of all of the classes as show in equation.

$$mAP = \sum_{q=1}^{Q} \frac{AveP(q)}{Q}$$

Figure 4.2: mAP equation

The precision and recall can be calculated by calculating the IOU(Intersection over union), where it is the ratio between area of the overlap and the area of union of the ground truth label and the prediction label. Threshold is used to decide if the prediction is true positive or false positive.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \quad Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

(a) (b)

Figure 4.3: Recall and Precision

In our first annotation attempt where we annotated the whole image, the model overfitted, it gave 99% accuracy. So, in the second annotation we decided to annotate the affected area only. The model performed great on the black rot class with 82.13% of the test images classified correctly, while the model didn't perform well on the scab class with only 38.84% test accuracy. The model couldn't learn to classify the scab class. The model mAP result was 58.49

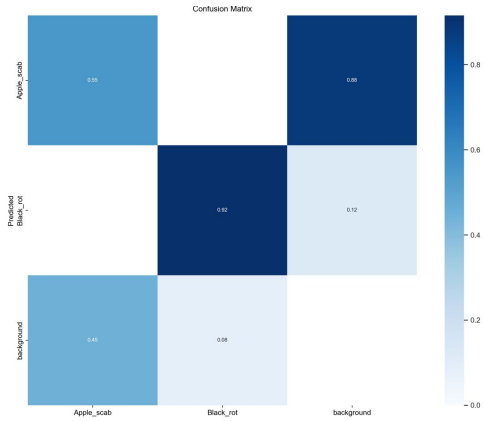


Figure 4.4: confusion matrix of the model trained with the first annotation method

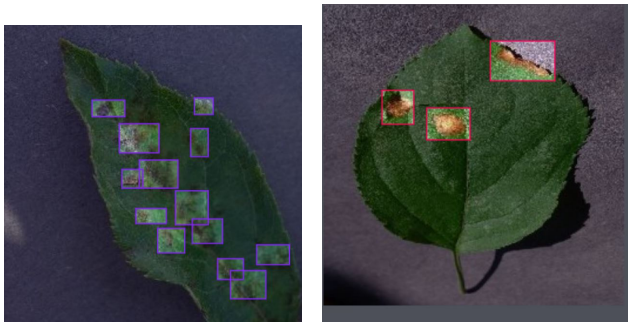
4.0.2 Experiments and results

In our third annotation attempt we decided to find what annotation practices we used in the black rot but didn't use on the scab class.

In the black rot class the brown circles don't appear as much in the leaf as much as the scab disease appears on the leaf. The black rot doesn't spread as much as the scab, so when we draw the bounding box, we draw a couple of boxes only, while in the scab disease the disease spreads all over the leaf, and sometimes it spreads through the veins of the leaf, which made us draw a lot of bounding boxes which caused the model to be less sure about the disease locality, meaning smaller probability of location.

So, we decided to group a lot of the effected area in fewer boxes for the scab disease. For instance, making 80% of the box contain the effected area will create bigger annotation boxes.

The second annotation practice we missed is that we drew small bounding boxes

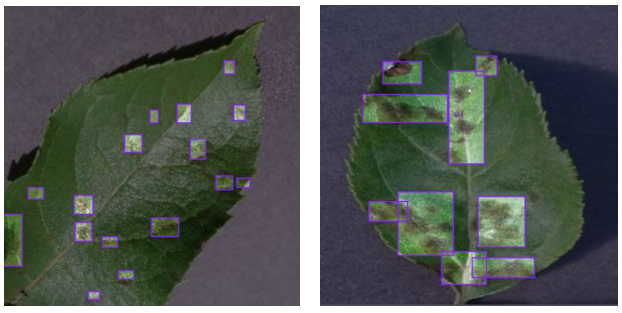


(a) (b)

Figure 4.5: difference between the amount of bounding boxes we have to draw between the two classes

in the scab class. In good annotating practices the bounding box shouldn't be less than 1.5% of the image dimension, but in the scab class there is a lot of disease spots that is smaller than the annotation threshold. We decided to ignore the small spots if we can't group them together.

The third annotation practice is not annotating large diagonal parts. In the scab class the edges of the leaf might be affected, so we can't annotate all of it since most of the box will include the background not the affected area.

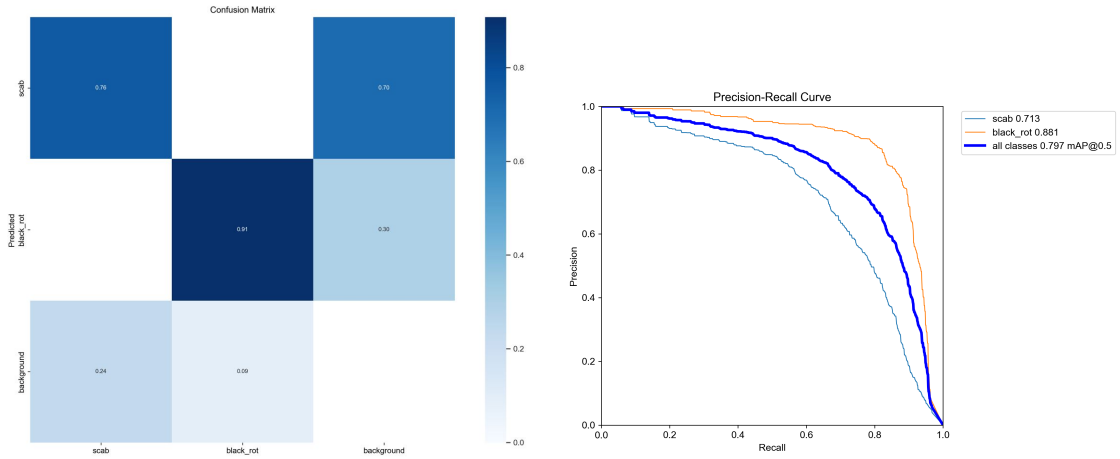


(a) (b)

Figure 4.6: Bad annotating practices

Third Annotating attempt: in the third attempt the YOLOv5 was able to learn both

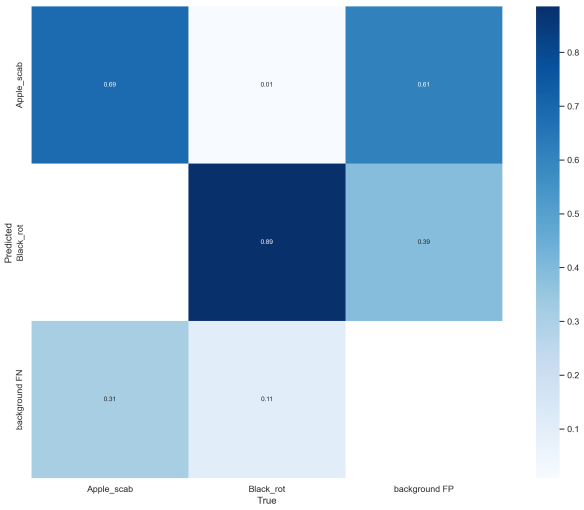
classes: black rot and scab. The mAP result of the result is 79.7%. The precision is 77.5% with recall of 76%.



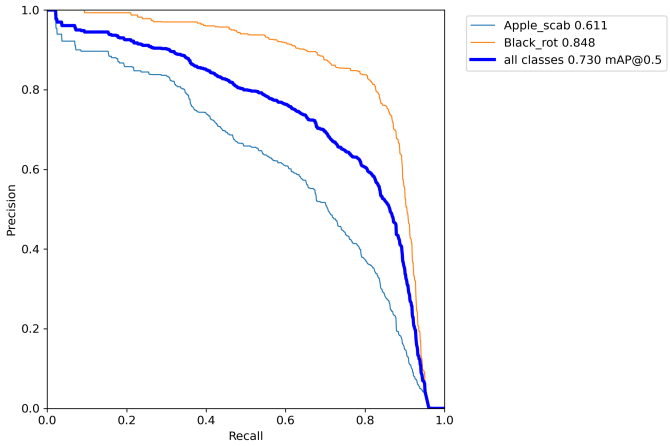
(a) (b)

Figure 4.7: (a) confusion matrix of the good annotation (b) mAP curve

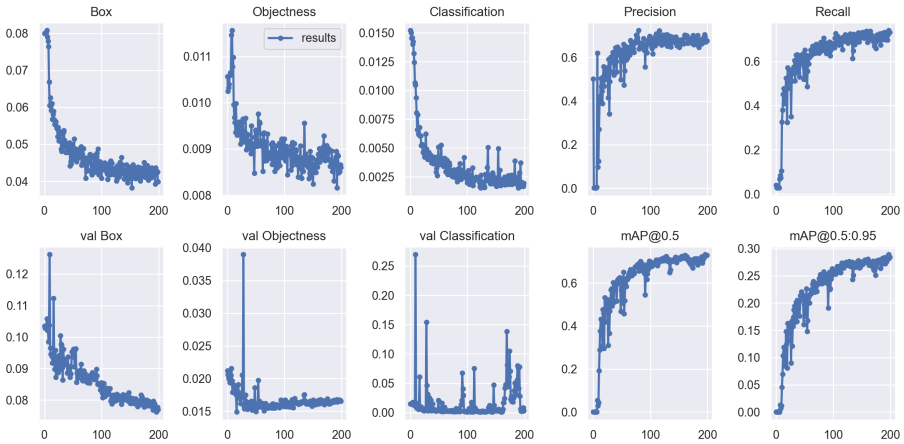
To see if we can make our results better we decided to train YOLOv7 model, to see if we can increase the mAP result, we tried the model with the exact same parameters as the YOLOv5. The result weren't as great as the YOLOv5, in some cases like in the scab the classification accuracy wasn't good enough, it performed marginally worse.



(a)



(b)



(c)

Figure 4.8: (a) confusion matrix (b) mAP curve (c) conclusive overview of the YOLOv7 performance

Chapter 5

Discussion

5.1 Related Work: Plant Disease Detection with YOLO

Many researchers based their leaf disease detection on YOLO's different versions as it is generalizable, fast and real-time. Even more, some versions can run on conventional GPU with 8-16 GB-VRAM.

In 2019, a project for apple lesions detection was made with YOLOv3-Dense, which is an alternation of the original YOLOv3, by adding more dense layers to the convolutional layers[15]. The researchers also used CycleGAN as a method of data augmentation, since the apple lesions are rare and the data set was insufficient. Their model achieved 95.57% accuracy, which is higher than VGG and GoogleNet, and cut down the average detection time greatly, enabling it to be almost real-time[15].

YOLOv3 was used similarily in onion plant disease in "An Aerial Weed Detection System for Green Onion Crops Using YOLOv3"[12], with precision of 93.81% and an F1 score of 0.94.

YOLOv5, released 2020, was utilized in application for the detection of diseases in the bell pepper plant[7] by Mathew and Mahesh. They proved the lightness of v5 over

v4, by reaching the same accuracy with almost 1:10 of the training time of YOLOv4 and a much smaller weight file size.

in the paper "Discovering Bias in the PlantVillage dataset"[10] the researchers surveyed all of the papers that uses the same dataset, they discovered that most of the result are higher than 98%, which means that most of the models in these papers are overfitted. The highest mAP score on the COCO dataset barely gets higher than 60%. So a lot of the papers' results can't be trusted since there is obvious bias exploited by the model. In the same paper[10] the researchers show that even by extending the dataset to new dataset, the issue can't be alleviated.

5.1.1 Demo Website

A website was developed to showcase the real time detection for apple plant disease. Our implementation is built upon the implementation found at this github repo.

The Web-App consists of two main parts; the back-end and the front-end. In the front-end React library is used to control and capture the images from the live feed camera. The captured image is sent to the back-end which is developed using FastAPI. The back-end process the captured image and send the predictions back to the front-end to be displayed on the website.

Both the front-end and the back-end are dockerized for seamless deployment on web hosting services. In the back-end we use torch hub. to load the base YOLOv5 model then we apply our custom weights to the model.

In figure5.1 we see a screen-shot of the website in action. This test run was made possible by blocking the live feed of the website camera with a third party tool and uploading an image of infected apple leaf directly to the website.

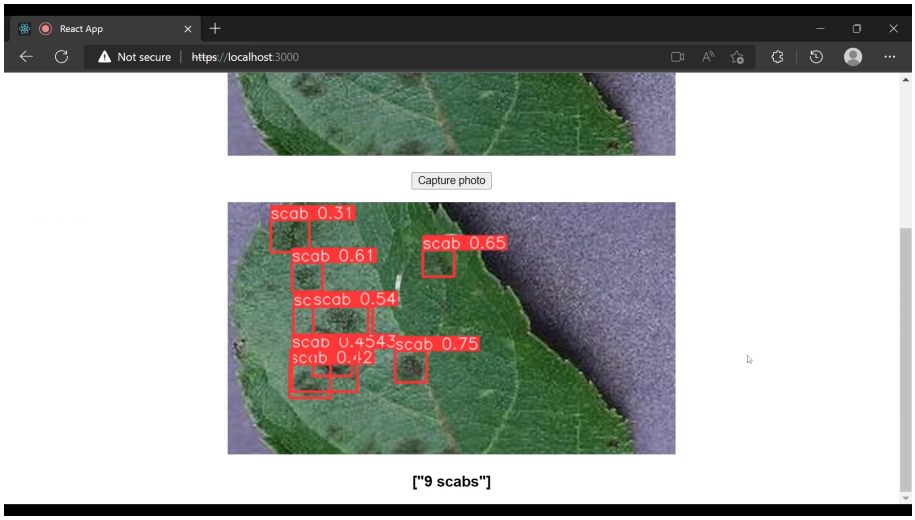


Figure 5.1: screenshot of the website in action

Now to explain the components of the website. First there is the live feed of the camera which was overridden by an uploaded image then there is the "capture photo" button which is used to capture a single picture from the live feed in order to send it to the back-end for prediction and finally the exist the response of the back-end which is the captured photo with the infected areas annotated.

Whenever the website to be deployed valid certificates for the front-end and back-end applications must be acquired in order to gain access to the camera.

5.2 Challenges Faced

1. The annotation process took so long, each plant class has on average thousands of images, and each image has multiple affected area that has to be annotated. One disease took on average 6 hours to fully annotate. This much time spent just drawing boxes.

2. The training process also took too long. On YOLOv4 model took 3 to 6 hours to train, although in later models like YOLOv5, and YOLOv7 the training model is much faster. YOLOv5, and YOLOv7 took no more than 3 hours to train, but even with faster training time, the time we would take to re-annotate and then train again was too long. The training time didn't reduce the time wasted because it required a lot of annotation.

3. Unfair dataset. A lot of time was consumed searching for the right annotation method and taking a lot of time to analyze how to fix the annotation, this is because the dataset was collected in a strict lab environment, which wasn't very inferring to the model. Most of the trials was a failure because the model overfitted or was biased and didn't learn right from the data. The bias in the PlantVillage dataset can be more discussed in this paper"Uncovering bias in the PlantVillage dataset"[10]

4. Extra resources. At he beginning we didn't expect that we would need more resources, but after working on both Colab, and our local pc, we discovered that we needed more resources to speed up the training process. Extra resources would have made a lot of difference in our research. It would have allowed us to experiment way more.

5.3 Lessons Learned

1. Time management, and assigning tasks. In our research our biggest problem was the time we consumed annotating, and re-annotating the images. The whole process was better when we divided the dataset classes on us, and divided the times when someone trains, because of the limited resources we had. Colab only gives the user 6

hours of GPU runtime every 24 hours, so we had to split our tasks, so we maximize the time we had.

2. Good annotation practices. When we started the research we all thought the annotation process is just drawing bounding boxes on the target object, but as we progressed through the research we discovered that there is specific practices that we have to follow. There is also the fact that every disease has to be annotated differently from other diseases. Black rot disease has small amount of red to black circles on each leaf, while in the scab disease the effected areas are all over the leaf. These two diseases appear way differently, and should be annotated differently.

3. Working with what we got. As we mentioned before, we didn't have enough resources to run the numbers of experiments we wanted. We had to divide our time, and resources to get the model to perform to the accuracy that we wanted.

5.4 Summary

In the beginning of our project, we wanted to explore the capability of YOLO models to classify plant diseases. The research was aimed to compare the 3 most advanced YOLO (v4, v5, v7) models, to see the performance difference between them. Then a problem appeared, the models couldn't learn. The models were overfitting, all of them gave 99.9% mAP accuracy, which is not plausible. So we had to explore our annotation methods to get the model to learn. Although different annotation methods made the model learn, the model couldn't identify all of the classes. The model only correctly identified one of the class 30 percent, of the time. After taking a look at how the model mis-classify the diseases, we looked at the best practices of

annotation, then we applied them on the dataset. The new annotation was really effective. The model gave a 82.13% mAP on the apple dataset, and 73% mAP on the bell pepper dataset. When we searched for papers with the same dataset, and the same models, we discovered that a lot of papers don't declare the annotation method they used, so we can't compare the results. The papers which used the same annotation method didn't publish the results, or their results don't seem plausible. In this paper we concluded how the PlantVillage is biased, and the only way to deal with it is to follow the best practices for annotation, and to keep training, and analyze how to tweak the annotation for each class.

5.5 Future Work

For future work in this domain a number of ideas can be further explored. These ideas include covering more crops in order to have a wider view on how this problem can be scaled and how the many different crops and their respective diseases will react to such inspection. Other data-sets that are more inline with real life scenarios can be examined and explored rather than data-sets created in strict lab environments which benefit most with the theory of the problem and not that much with the actual real life problem. Computer vision models different from YOLO can be explored and coupled with different annotation methods. A Web-App can be developed to be used as large scale detection system in mega farms where the website is monitoring different parts of crops continuously and it gives and alert whenever a diseases is detected for early treatment of the problem before it spreads into multiple plants.

Bibliography

[1] R. J. Achyut Morbekar, Ashi Parihar. Crop disease detection using yolo. *2020 International Conference for Emerging Technology (INCET)*, 2020.

[2] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. URL `https://doi.org/10.48550/arXiv.2004.10934`.

[3] G. M. Davis, J. The relationship between precision-recall and roc curves. *23rd International Conference on Machine Learning*, June 2006.

[4] R. G. A. F. Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection. *arXiv*, 2015.

[5] R. G. A. F. Joseph Redmon, Santosh Divvala. Yolov3: An incremental improvement. *arXiv*, 2018.

[6] D. G. Lowe. Object recognition from local scale-invariant features. 2:1150–1157, 1999.

[7] M. P. Mathew and T. Y. Mahesh. Leaf-based disease detection in bell pepper plant using yolo v5. *Signal, Image and Video Processing*, 2022.

[8] W. T. G. Maxwell, A.E. Accuracy assessment in convolutional neural network-based deep learning remote sensing studies—part 1: Literature review. *Remote Sens*, 2021.

[9] S. Mishra, R. Sachan, and D. Rajpal. Deep convolutional neural network based detection system for real-time corn plant disease recognition. *Procedia Computer Science*, 2020. URL `https://www.sciencedirect.com/science/article/pii/S187705092030702X`.

[10] M. A. Noyan. Uncovering bias in the plantvillage dataset. *arXiv preprint arXiv:2206.04374*, 2022.

[11] C. P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. pages 555–562, 1998.

[12] A. I. B. PARICO and T. AHAMED. An aerial weed detection system for green onion crops using the you only look once (yolov3) deep learning algorithm. *Engineering in Agriculture, Environment and Food*, 2020.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *Journal of Sensors*, 2016. URL `https://doi.org/10.48550/arXiv.1506.02640`.

[14] S. S. Sannakki, V. S. Rajpurohit, V. Nargund, and P. Kulkarni. Diagnosis and classification of grape leaf diseases using neural networks. *IEEE*, pages 1–5, 2013.

[15] Y. Tian, G. Yang, Z. Wang, E. Li, and Z. Liang. Detection of apple lesions in orchards based on deep learning methods of cyclegan and yolov3-dense. *Journal of Sensors*, 2019.

[16] H. Wang, G. Li, Z. Ma, and X. Li. Image recognition of plant diseases based on principal component analysis and neural networks. *IEEE*, pages 246–251, 2012.

[17] X. Xie, Y. Ma, B. Liu, J. He, S. Li, and H. Wang. A deep-learning-based real-time detector for grape leaf diseases using improved convolutional neural networks. *Front. Plant Sci. 11, 751.*, 2020. URL `https://doi.org/10.3389/fpls.2020.00751`.