Field Plant Identification Through Indoor Imagery Using Image Augmentation and Object Detection Algorithms

by

Parsa Sotoodeh

A Thesis submitted to the Faculty of Graduate Studies of
The University of Winnipeg
in partial fulfillment of the requirements of the degree of

Master of Science

Department of Applied Computer Science
University of Winnipeg
Winnipeg, Manitoba, Canada

Abstract

A growing global population and recent changes in climate make it increasingly necessary to incorporate recent advances in machine learning and robotics into agricultural practices. Plant detection is the problem of localizing and classifying all the plants within a given scene. Recent state-of-the-art object detection algorithms show promising results in detecting multiple objects and have great potential for outdoor plant detection. They require, however, a massive annotated plant dataset. It takes a great deal of time and expertise to manually annotate a plant dataset on such a large scale. We propose automating the plant annotation process in this thesis. Synthetic plant image datasets are generated in an outdoor setting by augmenting indoor images of plants that were captured and annotated automatically using a robotic camera system. We examine two different approaches: using image processing techniques to place plants on a soil background and using a generative adversarial model to generate fully synthetic outdoor datasets. We train two different plant detection algorithms on the synthetic datasets and evaluate the results on a manually-annotated outdoor dataset. Our best-performing dataset shows promising results for adoption in larger-scale automatic outdoor plant dataset annotation.

Acknowledgment

My deepest gratitude goes out to my supervisors, Dr. Christopher Henry and Dr. Christopher Bidinosti. They are a great team, but they are also incredible on their own. I have always found Dr. Henry to be a great teacher; he has always been patient and supportive throughout my academic career. It is truly inspiring to see his passion for teaching, his compassion for his students, and his joyful smile. I would also like to thank Dr. Bidinosti for his trust, sincere support, and the unique insight he provided during my studies.

Thanks to all the members of the TerraByte group at the University of Winnipeg, especially Dr. Michel Beck and Alexander Krosney, as I was able to collaborate closely with them and learn a lot from them.

Also, I would like to thank my parents for their unwavering support, especially over the last three years. Hard work, tenacity, love, and resilience have been some of their greatest lessons, and if they hadn't loved me unconditionally, I wouldn't have survived. I also want to thank my brother, Mani; you are my rock, my very foundation in this whole world, and I could never have asked for someone more intelligent and kind and unique than you. I am truly grateful to have you in my life, and your support helped me to have a strong foundation when I needed it the most.

I also want to thank my friend, especially the fantastic friends I made in Winnipeg. They are great company, and I cherish all my memories with them. I also want to thank the Mitacs and the University of Winnipeg staff for their support.

Dedication

Dedicated to Jina (Mahsa) Amini's pure spirit and all the gallant Iranian people who stand up against tyranny.

Contents

Contents iii

List of Tables v

List of Figures vi

1 Introduction 1

1.1 Problem Definition . . . 3

1.2 Contribution . . . 3

2 Methodology 4

2.0.1 Single computational neuron . . . 4

2.0.2 Activation functions . . . 5

2.0.3 Multi-layer perceptron (MLP) . . . 6

2.0.4 Gradient descent . . . 8

2.1 Deep learning, computer vision and object detection . . . 9

2.1.1 Convolutional neural network . . . 9

2.1.2 Object detection: problem definition . . . 12

2.1.3 Metrics commonly used in computer vision and object detection . . . 12

2.1.4 Object detection through YOLO . . . 16

2.1.5 YOLO Version 4 . . . 22

2.1.6 R-CNN . . . 24

2.1.7 Feature Pyramid Networks . . . 26

2.1.8 Generative Adversarial Networks . . . 27

3 Plant detection literature review **30**

4 Outdoor plant identification, datasets and experimental settings **33**

4.1 Indoor dataset 34

4.2 Outdoor dataset 36

4.3 Validation outdoor dataset 37

4.4 Composite dataset 38

4.5 Synthetic 1 dataset 41

4.6 Synthetic 2 dataset 44

5 Improved field plant classification using synthetic plant image generation **49**

5.0.1 Experiments settings 50

5.0.2 YOLOV5 train technical setting 50

5.0.3 Faster R-CNN technical settings 51

5.1 Binary Problem 52

5.2 Multiclass 54

6 Conclusion **57**

6.1 Future work 57

List of Tables

4.1 Natural datasets properties. 37

4.2 Synthetic dataset properties 48

5.1 Number of parameters of different YOLOV5 models. 51

5.2 YOLOV5 trained on synthetic datasets, binary problem. 54

5.3 YOLOV5 trained on synthetic datasets, multiclass, all the classes. 55

5.4 YOLOV5 trained on synthetic datasets, multiclass, canola. 55

5.5 YOLOV5 trained on synthetic datasets, multiclass, soybean. 55

List of Figures

1.1 Flowchart of processes 2

2.1 Architecture of a single neuron in a neural network 5

2.2 MLP 6

2.3 An example of 2D Convolution. 10

2.4 Max-pooling. 11

2.5 Intersection Over Union. 13

2.6 YOLO1 architecture 17

2.7 Overview of YOLO architecture. 18

2.8 Analysis of the number of clusters vs IOU on the COCO and VOC 2007 dataset 19

2.9 Darknet-53 architecture 20

2.10 YOLOV3 Architecture 21

2.11 Overview of Object Detection Architectures Concepts. 22

2.12 Dense blocks and transition layers interconnection 23

2.13 Mosaics of images 23

2.14 R-CNN Architecture 25

2.15 Fast R-CNN Architecture. 26

2.16 Architecture of Feature Pyramid Networks. 27

2.17 General workflow of GAN 28

4.1 EAGL-I system 34

4.2 Indoor dataset . . . 35
4.3 Samples of outdoor data. The figure on top is the soybean in outdoor setting and on the bottom is the canola in outdoor setting. . . . 36
4.4 Single indoor image of canola and its corresponding mask. . . . 39
4.5 Samples of the Composite dataset . . . 39
4.6 Flowchart of composite data generation. . . . 40
4.7 Composite dataset generation . . . 41
4.8 Soybean in indoor setting versus outdoor setting. . . . 41
4.9 Synthetic 1 dataset generation. . . . 43
4.10 Samples of the Synthetic 1 dataset . . . 44
4.11 Synthetic 1 canola with clear signs of distortion around the leaves. . . . 44
4.12 Samples of cropped images that were removed . . . 45
4.13 Single outdoor canola . . . 46
4.14 Samples of Synthetic 2 dataset . . . 46
4.15 Synthetic 2 dataset generation. . . . 47

5.1 Different YOLOV5 models performance and speed trade off . . . 50
5.2 Confusion matrix for the binary problem . . . 53
5.3 Confusion matrix for the binary problem. . . . 56

Chapter 1

Introduction

With the earth's population rising at unprecedented levels [1], the need for more food supply, including agricultural products, is increasing. These changes have created a need for more sustainable methods in agriculture practices. With recent technological advances, specifically crewless vehicles [2] and artificial intelligence [3], new opportunities are available to incorporate these novel tools into a solution for a greater population. Precision agriculture [4] is defined as using recent technological advances to address agriculture's challenges and modernizing farming practices to make them more efficient and sustainable.

Plant monitoring is one of the challenges faced in farming. For example, one obvious use of plant monitoring is weed removal. As crops grow in an agricultural field, inevitably, weeds grow among them. The growth of weeds hampers the growth of crops, as they have to compete for resources such as water, sunlight and other resources. Therefore, eliminating weeds is of utmost importance for a more efficient yield. Using herbicides to remove weeds can cause contamination of vital resources [5] such as water and fertile soil. One essential solution is to use robots and autonomous vehicles equipped with an embedded plant detection system to localize and detect plants. If the robot can localize them, then the task of removal can be achieved, for example, by lasers or mechanical means (rather than harmful herbicides).

Object detection is the problem of finding and classifying all of the instances of objects found within an image. Recent advances in machine learning, specifically convolutional neural networks, provide robust and powerful object detection models [6] that can also be employed for plant detection.

The main bottleneck in enabling such solutions is the requirement for massive annotated datasets to train the plant detection system. As long as such data are not readily available and difficult to obtain, such techniques may ultimately remain restricted to a limited range of case studies. Review studies [7], as well as works focusing on specific applications such as weed detection [8] highlight the difficulty in finding good-quality ground-truth datasets for plant detection. A wide range of differences in growing

conditions and physical dissimilarities between plants, even those belonging to the same genotype, further complicate the generation and collection of clearly labelled plant data. As the model will have to operate within an outdoor agricultural field, weather conditions such as rain, haze, wind, extreme temperatures, sun visibility, clouds and insects feeding on the plants introduce changes to the data that make it even more challenging to generate. Moreover, even after obtaining the data, we need experts educated in recognition of different crops to annotate the data and specify the exact position of each plant within each image. Involving human specialists to annotate massive datasets is labour intensive and an expensive task. All of these reasons highlight the dire need for scalable and cost-effective methods for generating massive labelled datasets of plants.

The purpose of this thesis is to examine a novel pipeline (see Figure 1.1) for generating synthetic outdoor data using indoor data captured in controlled environments and put its effectiveness to the test by evaluating its ability to be used for training object detection models.

Figure 1.1: Flowchart of processes used in this thesis. The process begins by generating synthetic outdoor data from real indoor image data. This synthetic data is used to train an outdoor object detection method. Finally, we evaluate the trained model on real outdoor data to test the effectiveness of synthetic data.

We use the indoor data generated by EAGL-I [9] (see e.g. Figure 4.1) in a greenhouse setting as one of the inputs to our pipeline. We train object detection models with two available synthetic datasets, named Synthetic 1 and Synthetic 2. A Composite dataset is also created with placing plants on soil background. This is called the Composite dataset throughout the thesis. Synthetic 1 dataset is created with adversarial generative models. Finally, we create another synthetic dataset, Synthetic 2, by fine-tuning the Synthetic 1 dataset and addressing its shortcomings. We also annotate a small subset of the outdoor dataset. We call this the Validation outdoor dataset. We use this dataset to evaluate the effectiveness of the synthetic dataset in training the object detection model.

1.1 Problem Definition

This thesis aims to solve the problem of finding all instances of plants in an outdoor setting given an image or frame from a video containing multiple instances of plants. We are specifically interested in seeing the coordinates of the plant within the image. In addition, we want to classify the type of plant. A massive dataset of plants in an outdoor setting is required to use object detection models.

The process of creating an annotated outdoor dataset of plants is time-consuming and expensive. In order to harness the power of machine learning for agricultural applications and utilize it in vast commercial capacities, we need robust, cost-effective, and minimally human-involved methods that generate annotated datasets. Additionally, methods must be scalable and adaptable to different conditions, such as those encountered in other geographical locations and within their particular environments.

Accordingly, we want to generate datasets that will be used in the training of object detection models to detect plants outdoors.

1.2 Contribution

This thesis provides the following major contributions:

1. Generating synthetic datasets with generative adversarial models.
2. Utilizing three synthetic datasets to train two state-of-the-art object detection models and evaluating their performance on outdoor data without explicitly utilizing outdoor data during training.
3. Annotating a small outdoor dataset used for evaluation.

Chapter 2

Methodology

In this chapter, the theoretical foundations for field plant identification are introduced. Machine learning is the process of using statistical methods to enable computers to learn without being explicitly programmed [10]. Neural network-based models are a type of machine learning algorithms. An overview of neural networks is provided, followed by the introduction of a specific type of neural network, convolutional neural networks [11]. We then conceptualize how employing several layers of neural networks yields more powerful machine learning models [12]. Finally, we describe in detail how our field plant identification classifier is an extension of these concepts.

2.0.1 Single computational neuron

Neural networks consist of computational units called neurons. Inspired by the neurons in the brain, every single neuron can have n inputs with each input x_i associated with a respective weight w_i. Each neuron performs a weighted sum followed by the addition of a bias b. This results in a scalar that is the linear product of the weight and input vectors as well as the bias term. An activation function φ is a mathematical function applied after the summation step to get the output. This mathematical function is non-linear with respect to its input. This gives the network the ability beyond scaling (multiplication) and shifting (summation) to capture more complex patterns. Let X be the n-dimensional input vector and W the $n \times m$ weight matrix where m is the number of neurons. Output, $O(X)$, can be expressed mathematically

$$O(X) = \varphi(\big(W^T X + b\big)). \tag{2.1}$$

Figure 2.1 shows the architecture of a single neuron [13]. We next look at some common activation function choices.

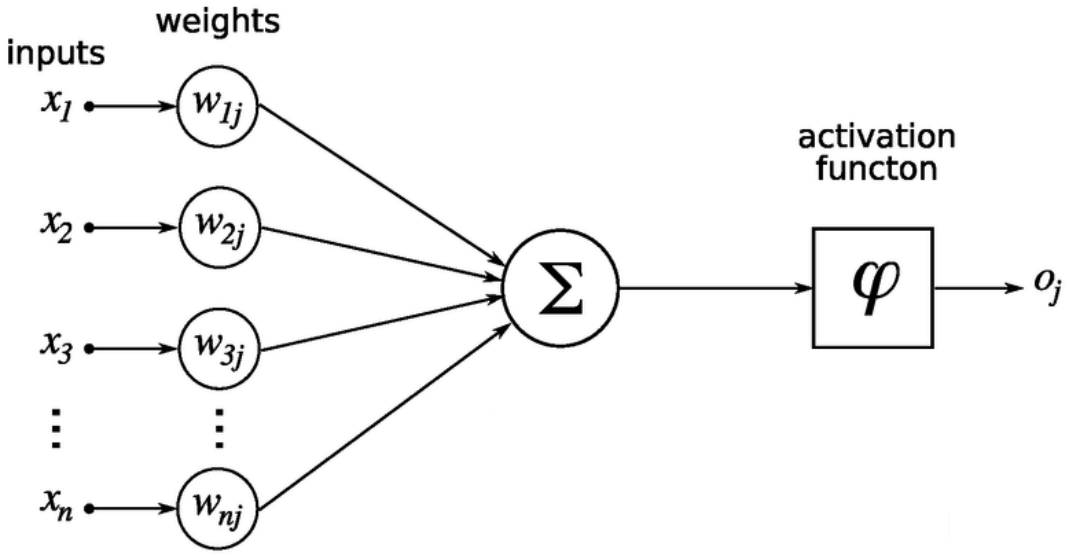


Figure 2.1: Architecture of a single neuron in a neural network, where j denotes the j-th neuron [13].

2.0.2 Activation functions

In this section, we introduce three commonly used nonlinear functions for neural networks [12] applied to obtain output vector, see Eq. 2.1.

Sigmoid: Mathematically, the sigmoid function is defined as

$$y = \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.2}$$

It takes real values as its input and outputs a number between 0 to 1. Therefore this function is used when we want to predict the probability of a sample belonging to a class.

Hyperbolic Tangent: The hyperbolic tangent function is defined as

$$y = 2\sigma(2x) - 1. \tag{2.3}$$

It takes real values as its input and outputs a number between -1 and 1.

RELU: The mathematical form of the RELU function is

$$y = max(0, x). \tag{2.4}$$

This function maps negative inputs to zero, therefore this decreases the ability of function to model negative outputs.

2.0.3 Multi-layer perceptron (MLP)

A neural network layer is a group of single neurons stacked together. A multi-layer perceptron (MLP) is a group of layers stacked together. Early work started in the 1940s with biologically inspired models such as perceptron and multi-layer perceptron [14]. It learns a function $f(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^o$ after training on a dataset, here n denotes the number of dimensions for input and o is the number of dimensions for output and $\mathbb{R}$ is the set of real numbers. Supplied the set of inputs (features) $X = x_1, x_2, \ldots, x_n$ and a target y, it can learn a non-linear function approximation for classification. There may be one or more layers between the input and the output layer, called hidden layers. The general schema of a one hidden layer MLP with scalar output is shown in Figure 2.2.

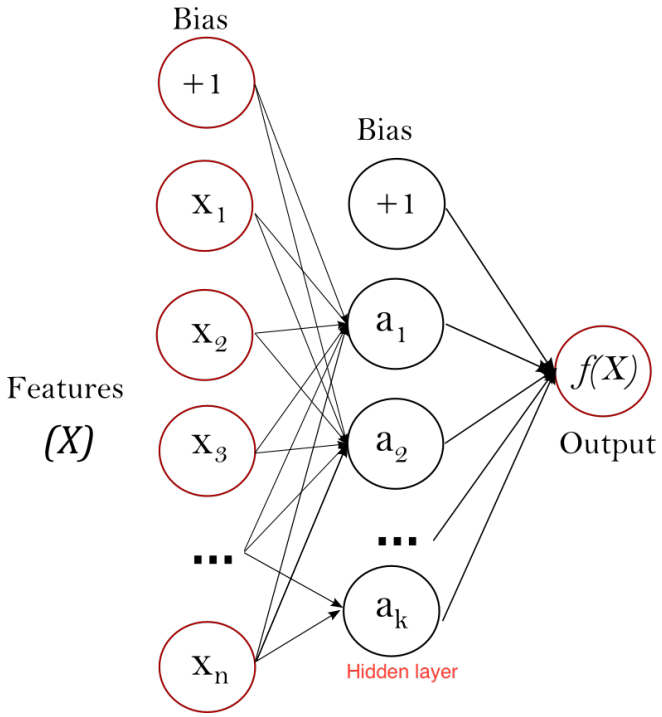


Figure 2.2: One hidden layer MLP, consisting of the input (feature) layer, followed by another layer (hidden) and output layer with the function f applied to the output [15].

Assuming a set of training examples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ where $x_i \in \mathbb{R}^n$ and $y_i \in \{0, 1\}$, a one hidden layer one hidden neuron MLP learns the function $f(x) = W_2^T g\left(W_1^T x + b_1\right) + b_2$ where $W_1 \in \mathbb{R}^m$ and $W_2, b_1, b_2 \in \mathbb{R}$ are parameters that will be learned by the model. W_1, W_2 show the weights of the input layer and hidden layer accordingly; and b_1, b_2 denote the bias added to the hidden layer and the output layer, respectively. $g(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ is the activation function.

For binary classification, $f(x)$ is transformed through the sigmoid function to get output values bound between zero and one. A threshold θ would assign samples of outputs equal or larger than θ to the positive class, and otherwise to the negative class.

If the problem is defined by more than two classes, $f(x)$ would become a vector with length equal to number of classes. Furthermore, in place of the logistic function, the softmax function would be used as in

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}. \tag{2.5}$$

In Equation 2.5, z_i denotes the ith element of the input to softmax, corresponding to class i, and k is the number of classes. The resulting vector contains the probabilities of sample x belonging to each class. The output is then set as the class with the highest probability.

Loss function

In order to quantify the network's ability to approximate the mapping of training data input to output, we define a loss function. For each sample, we want to measure the difference between the predicted output x_i (observed probability vector) and the actual value (ground truth), y_i. The function that measures this difference for each sample is called the loss function. If the predicted value is near the value of the ground truth, we want the loss to be small, whereas, when the predicted output is far from the ground truth, the loss should be big. The loss function sums or averages all the losses of the collection of samples. If the function is differentiable, we can use its partial derivatives with respect to its inputs i.e gradients, as a measure to adjust the parameters of the network such as weights and biases.

Two of the most commonly used loss functions are, mean square error (MSE) and cross-entropy (CE) [12].

For D inputs, the MSE sums the square of difference between the output of network x_i and the actual value (ground truth) y_i from the training data. The mathematical form of the MSE is

$$MSE(x_i, y_i) = \sum_{i=1}^{D} (x_i - y_i)^2. \tag{2.6}$$

The mathematical form of cross-enthropy is

$$CE = -\sum_{i=1}^{D} y_i \log(x_i). \tag{2.7}$$

2.0.4 Gradient descent

As discussed in the last section, we want to adjust the parameters of the MLP. We defined a loss function that measures the difference between the prediction and ground truth. We want to optimize this function. Gradient descent [16] is an iterative optimization algorithm that finds the local minimum of the function at each step by taking steps in the opposite direction of the derivative of the function at the given step. Assume the cost function J, which depend on a vector of variables $W = (w_0, w_1, .., w_n)$. Gradient descent starts at the initial point $W^0 = (w_0^0, w_1^0, .., w_n^0)$, and at step i+1 changes these value of variable j according to the formula in Eq. 2.8, where α is the rate of the step.

$$w_j^{i+1} = w_j^i - \alpha \frac{\partial J}{\partial w_j}. \tag{2.8}$$

Backpropagation

Let N be an MLP, consisting of k layers $\vec{L} = (l_0, l_1, ..., l_k)$, with each layer having k input $X = (x_0, x_1, ..., x_k)$, k+1 weights $W = (w_0, w_1, ..., w_k)$, and the activation of $\vec{A} = (a_0, a_1, ..., a_k)$.

For each layer l_i we compute its output as

$$a_i = f(w_i x_i + b_i), \tag{2.9}$$

where f is the activation function and b_i is the bias term. In this MLP, the input of the layer l_i is output of l_{i-1}. After computing the output of the last layer, we compute the loss function C and we get the loss of the layer k

$$\sigma_k = \frac{\partial C}{\partial a_k} f^{'}(w_i x_i + b_i), \tag{2.10}$$

where σ_k is the loss of the layer k and $f^{'}$ is the derivative of activation function with respect to its input $(w_i x_i + b_i)$.

The error for the backward layer l_i is calculated as

$$\sigma_i = (w_{i+1}^T \sigma_{i+1}) \cdot f^{'}(w_i x_i + b_i), \tag{2.11}$$

In one forward pass the MLP computes the output of each layer consecutively. In order to optimize the loss function C, it uses the error computed in each layer, going backward and updates the values of the paramers as in Eq. 2.8.

2.1 Deep learning, computer vision and object detection

In a neural network, there can be varying numbers of layers between the input and output layer. As the number of layers increases, the distance between input and output increases or gets deeper. This class of neural networks is called deep neural networks. Deep learning is one of the areas of machine learning, consisting of neural networks with a high number of layers [12]. A notable model in visual problems, convolutional neural networks (CNNs), were introduced to classify handwritten digits [11]. The modern era of deep learning picked up the pace in 2012 with the advent of more complex architectures [17]. In recent years, deep learning has been very successful in computer vision, especially on tasks such as image classification, object detection and image segmentation [18]. This progress has been made mainly due to improvement in computational power and availability of annotated datasets [19]. We next look at some fundamental concepts commonly used in computer vision models.

2.1.1 Convolutional neural network

A convolutional neural network (CNN) is a modified MLP that is commonly used to process images. CNN outperforms MLPs in this area, despite the fact that MLPs can also classify images. Whereas MLP input is a vector, CNN input is a tensor. Therefore the spatial relation of input is preserved. In a CNN, the connections are sparse compared to MLP, this results in more efficient computations.

A CNN is composed of consecutive layers. In each layer, there can be multiple filters. Each filter is a set of kernels stacked together. These layers are called convolutional layers. A kernel can be thought of as a matrix with the weights as its parameters. Each filter is applied consecutively to local regions of input. This process can be thought of as convolving the input with the filter, hence the name convolutional. Figure 2.3 shows an illustrative example of a 2D convolution. This results in a filtered output. All the filtered outputs are stacked together, a bias term is added, and an activation function is applied to obtain a feature map. The weight of kernels and the bias term are learnable parameters that will be learned as the weight and bias parameters in MLP. The final layer is a layer composed of several neurons, with each neuron connected to all the activations in the previous layer. This layer is called a fully connected layer.

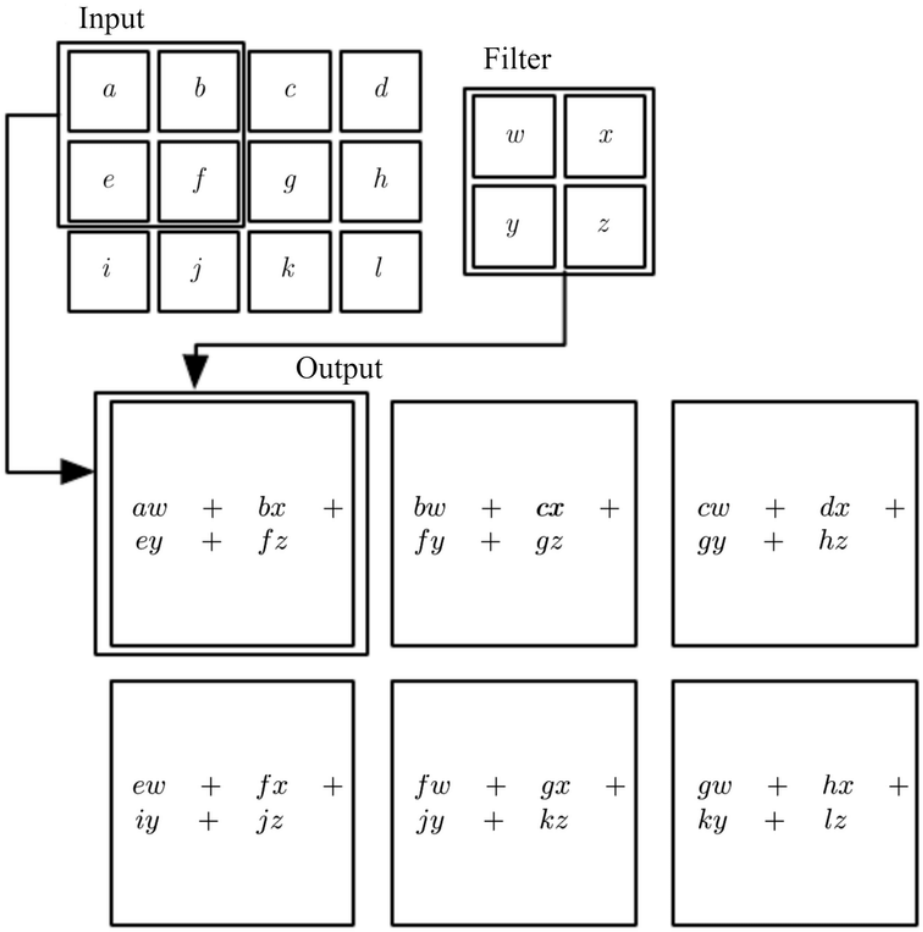


Figure 2.3: An example of 2D Convolution [12]. The filter slides through the input to obtain the output.

Max-pooling

Having too many features can make computing expensive as the number of parameters increases. To prevent that, a filter is applied to the feature map, and the maximum value extracted over the region to which the filter is applied. Max-pooling also extracts the most dominant feature, by discarding the less important features [20].

Pooling aims to achieve spatial invariance by reducing the resolution of the feature maps. Each pooled feature map is associated with one feature map of the previous layer. Their units combine the input from a small $n \times n$ patch of units, as indicated in Figure 2.4. This pooling window can be of arbitrary size, and windows can be overlapping.

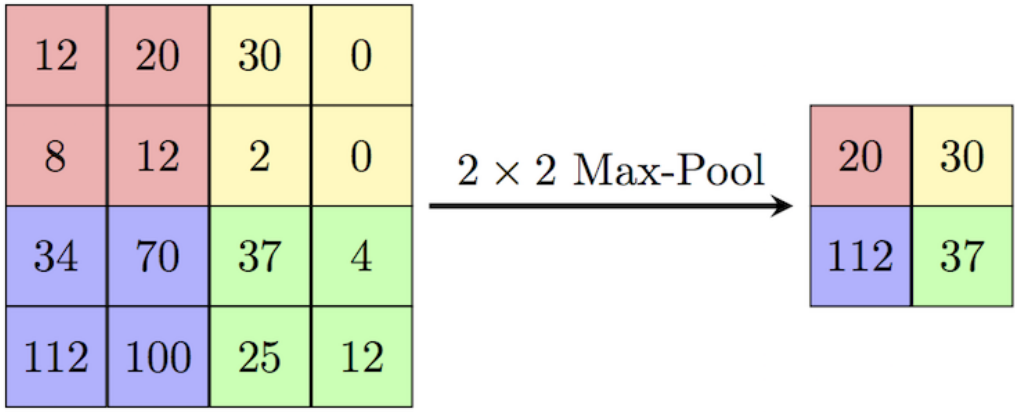


Figure 2.4: 2 by 2 Max-pooling operation, reducing a 4 by 4 feature map to a 2 by 2 feature map [21].

Batch normalization

Batch Normalization (BN) is one of the normalization methods for neural networks, namely, the process of normalizing the input of each layer, by shifting the mean to zero and variance to one [22]. Often inputs to neural networks are normalized to either the range of [0, 1] or [-1, 1] or to mean=0 and variance=1 (a.k.a. Whitening). This helps with speeding up the learning by making the convergence faster, therefore reducing the training times [22].

During training, a batch normalization layer does the following [23]:
1. Calculate the mean, Eq. 2.12 and variance, Eq. 2.13 of the layers' input (x_i).

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i. \tag{2.12}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 . \tag{2.13}$$

2. Normalize the layer inputs using the previously calculated batch statistics Eq. 2.14. Here ϵ is added to prevent division by zero.

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}. \tag{2.14}$$

3. Scale and shift to obtain the output of the layer.

$$y_i = \gamma\overline{x_i} + \beta. \tag{2.15}$$

Notice that in Eq. 2.15 γ and β are learned during training along with the original parameters of the network.

2.1.2 Object detection: problem definition

The problem of object detection can be summarized as finding and classifying a variable number of objects in a given image. Note that the variable number of objects in different images distinguishes this problem from classification. That is, unlike classification, object detection model output can have a different length for each sample, as the true number of objects varies across images.

In the 1960s Roberts [24] identified the need to match two-dimensional features generated from images with the three-dimensional representations of objects. Later works explored the practical difficulties for reliable and consistent matching, specifically with increasing scene complexity, illumination variability, and as time, cost, and sensor noise effects became more evident [25].

2.1.3 Metrics commonly used in computer vision and object detection

There are numerous metrics evaluating object detection model's performance. Below, we define those that have been used in this thesis. We use survey [26] as our reference.

Intersection Over Union (IOU)

Object detection is the process of locating and classifying instances of an object. For example, in an agriculture field, finding the plants, and their coordinates and classifying them (weed or crops). The intersection over union (IOU) metric is used to compare the predicted bounding box to the ground

truth bounding box. *IOU* is defined as the overlap area between the predicted bounding box B_p and ground truth bounding box B_{gt} over their union, which is depicted in Figure 2.5. The value varies from 0, the case where the predicted box does not overlap with the predicted box, to 1 where they overlap completely. Formally, the *IOU* is defined as

$$IOU(B_{gt}, B_p) = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}. \tag{2.16}$$

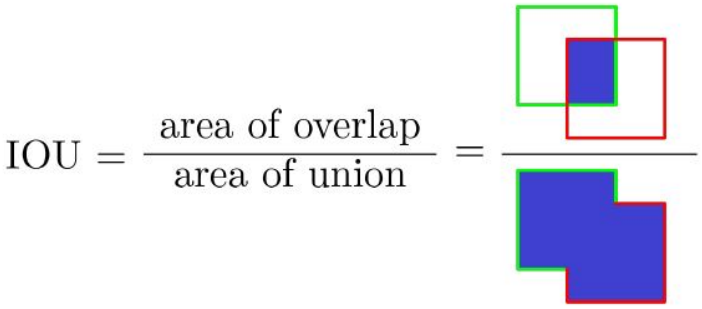


Figure 2.5: Intersection Over Union [27]. Diagram depicting intersection over union. The figure on the right contains a graphical representation of the overlapping area and union of two bounding boxes.

Precision and recall

If we have G ground truth instances of objects in a dataset, and an object detector model predicts N instances, of which S are correct, we can measure how many of the detections are relevant and how many of the ground truth instances have been detected correctly. In order to calculate the precision and recall, each bounding box detection is classified as one of the following items.

- True Positive (TP): A correct detection of the ground truth bounding box.
- False Positive (FP): An incorrect detection, detection of non-existing ground truth, detection of ground truth in the wrong coordinates
- False Negative (FN): An undetected ground truth instance

Precision is defined as the percentage of correct positive detections, and is given by

$$Pr = \frac{\sum_{n=1}^{S} TP_n}{\sum_{n=1}^{S} TP_n + \sum_{n=1}^{N-S} FP_n} \tag{2.17}$$

Recall is the percentage of all positive detections across all ground truth instances and is defined as

$$Rc = \frac{\sum_{n=1}^{S} TP_n}{\sum_{n=1}^{S} TP_n + \sum_{n=1}^{G-S} FN_n} \tag{2.18}$$

Average precision (AP)

When an object detection model predicts an instance of an object, it outputs a bounding box, a class and confidence. The confidence measures how confident the model is about the corresponding prediction. Therefore we can redefine precision and recall for each confidence τ, considering only predictions with confidence higher or equal to the τ as a valid prediction.

Formally, if the model predicts that the sample s belongs to class c and the ground truth label of s is gt, then the prediction is true positive as a function of threshold τ if

$$TP(\tau) = (pr(s \in c) \geqslant \tau) \wedge (gt = c). \tag{2.19}$$

Similarly, prediction is false positive if

$$FP(\tau) = (pr(s \in c) \geqslant \tau) \wedge (gt \neq c). \tag{2.20}$$

Also, prediction is false negative if

$$FN(\tau) = (pr(s \in c) < (\tau)) \wedge (gt = c). \tag{2.21}$$

Precision as a function of τ can be defined as

$$Pr(\tau) = \frac{\sum_{n=1}^{S} TP_n(\tau)}{\sum_{n=1}^{S} TP_n(\tau) + \sum_{n=1}^{N-S} FP_n(\tau)}. \tag{2.22}$$

Recall as a function of τ can be defined as

$$Rc(\tau) = \frac{\sum_{n=1}^{S} TP_n(\tau)}{\sum_{n=1}^{S} TP_n(\tau) + \sum_{n=1}^{G-S} FN_n(\tau)} \tag{2.23}$$

As we increase the confidence, true positives and false positive decreases. On the other hand, false negative increases as confidence increases. Therefore recall is a decreasing function of confidence. Since both true positives and false negatives change with confidence, nothing can be said about the relationship between confidence and precision. This results in zig-zag behaviour as confidence varies.

An ideal object detector has high precision and recall regardless of confidence. This means even when the confidence is increased, the recall and precision values remains high. In order to measure how the precision and recall values change with confidence, the metric average precision (AP) is defined as the area under the precision-recall curve.

In order to compute AP, K different confidence values $\tau(k)$ are ordered,

$$\tau(k), k = 1, 2, ..., K \text{ such that } \tau(i) > \tau(j) \text{ for } i > j.$$

Since recall has a one-to-one relationship with τ, the continuous $Pr \times Rc$ curve is sampled at $(Pr(\tau(k), Rc(\tau(k))$ points indexed by k.

Now, an ordered set of reference recall values $R_r(n)$ are defined such that

$$R_r(n), n = 1, 2, ..., N \text{ such that } R_r(m) < R_r(n) \text{ for } m > n.$$

AP is computed using the two ordered sets of the above. Since the $(Pr(\tau(k), Rc(\tau(k))$ points are discrete, we need to interpolate these points before computing the area under the curve. The function $Pr_{interp}(R)$, as in Eq. 2.24 is used where R is a real value in the interval $[0, 1]$.

$$Pr_i(R) = \max_{k|Rc(\tau(k))\geq R} \{Pr(\tau(k))\}. \tag{2.24}$$

The area under the $Pr \times Rc$ curve, AP, is calculated by a Reiman integral of $Pr_i(R)$ using the K recall values as sampling points as in Eq. 2.25.

$$AP = \sum_{k=0}^{K} (R_r(k) - R_r(k+1)) Pr_i(R_r(k)). \tag{2.25}$$

To compute the integral in Eq. 2.25, there are two approaches, N-Point Interpolation and All-Point Interpolation.

N-Point Interpolation

In this method, recall reference values are equally spaced in the interval $[0, 1]$

$$R_r(n) = \frac{N - n}{N - 1}, \text{ n=1,2,..., } N.$$

Therefore, the AP will be calculated as in 2.26.

$$AP = \frac{1}{N} \sum_{n=1}^{N} Pr_i(R_r(n)) \tag{2.26}$$

All-Point Interpolation

In this method, all the K values are used in computation of the Eq. 2.25 with their corresponding recall values. Confidence for points $k = 0$ and $k = K + 1$ is set to $\tau(0) = 0$ and $\tau(K + 1) = 1$ with their corresponding recall values being set to $Rc(\tau(0)) = 1$ and $Rc(\tau(K + 1)) = 0$.

Mean Average Precision

AP is obtained individually for each class. In order to have a sense of the performance of an object detection model across all the classes, AP is averaged over all the classes as in 2.27.

$$mAP = \frac{1}{C}\sum_{i=1}^{C} AP_i. \tag{2.27}$$

AP_i is the value of AP for the $i-th$ class and C is the total number of classes in the dataset.

2.1.4 Object detection through YOLO

With the publication of "You Only Look Once: Unified, Real-Time Object Detection" (YOLO) in 2016 [28], YOLO made its debut in the object detection area. YOLO aimed to unify all phases of object detection such as bounding box detection and classification. It outputs vectors for each object that it encounters in the input after passing through the network. In contrast to iteratively classifying different regions on an image, the YOLO computes all the features of the image and makes predictions for all objects at once. In YOLO, the image (see Figure 2.6) is divided into a grid of size $S \times S$ (7×7 by default). When the center of an object falls within a grid cell, the grid cell is responsible for detecting that object. Every cell in the grid has B bounding boxes with confidence scores indicating the likelihood of the presence of an object in the box. The confidence score is defined as in 2.28.

$$confidence = P(object) \times IOU_p^{gt} \tag{2.28}$$

In Eq. 2.28, $P(object)$ is the probability of the presence of an object and IOU is the intersection over the union of the predicted bounding box and the ground truth bounding box. Each bounding box has parameters (x, y, w, h). (x, y) that are the center coordinates of the bounding box relative to its corresponding grid cell. W and h are the width and height of the box. When multiple bounding boxes are predicted for the same object, YOLO measures the IOU of each box with the box with the highest score, and discards boxes having IOUs greater than a certain threshold.

When YOLO was first introduced, it was evaluated on the Pascal VOC dataset [29], which has 20 classes of objects ($i.e. C = 20$). The grid size is 7 ($S = 7$) and 2 bounding boxes per grid cell ($B = 2$). Each grid cell will have one output vector, each output vector has parameters of the box and the probability of the object belonging to each class in the box, therefore the dimension of the vector would be $(B \times 5 + 20)$. The grid is illustrated in Figure 2.6 with more details.

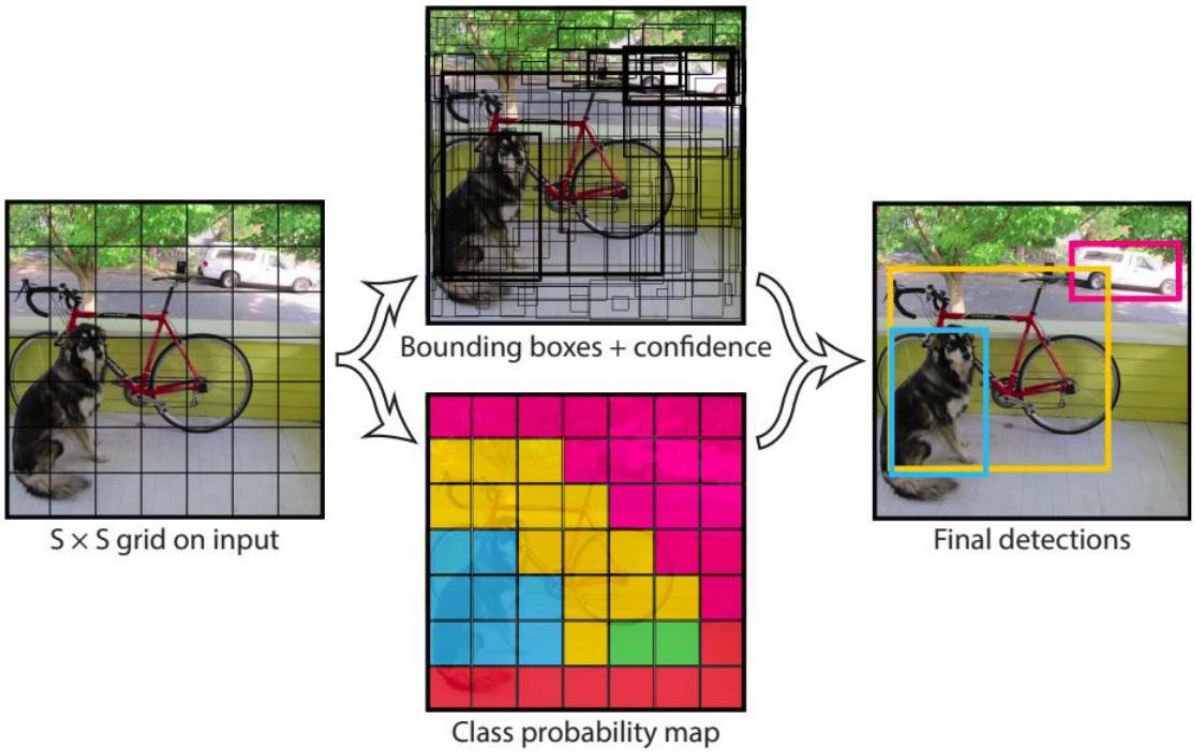


Figure 2.6: Dividing input into $S \times S$ grid, predicting B bounding boxes for each cell and confidence and C class probabilities for each box. We keep only the boxes that have the IOU and confidence beyond a certain threshold and discard the rest [28].

YOLO has 24 convolutional layers followed by 2 fully connected layers. The task of convolutional layers is to extract features from the image and the fully connected layers predict output parameters. The first 20 convolutional layers are used for feature extraction and the last 4 convolutional layers are responsible for finding objects. The output of the final convolutional layer is a tensor with shape (7, 7, 1024). Using 2 fully connected layers as a form of linear regression, it outputs 7×7×30 parameters. Figure 2.7 depicts the architecture of YOLO.

For training, YOLO uses the sum-squared as the loss function, as it is easy to optimize. The function objective is to optimize spatial parameters of the bounding box, (x, y, w, h), C objectness (presence of an object in the bounding box) and class probabilities $p(c)$. The loss function is defined as in eq. 2.29.

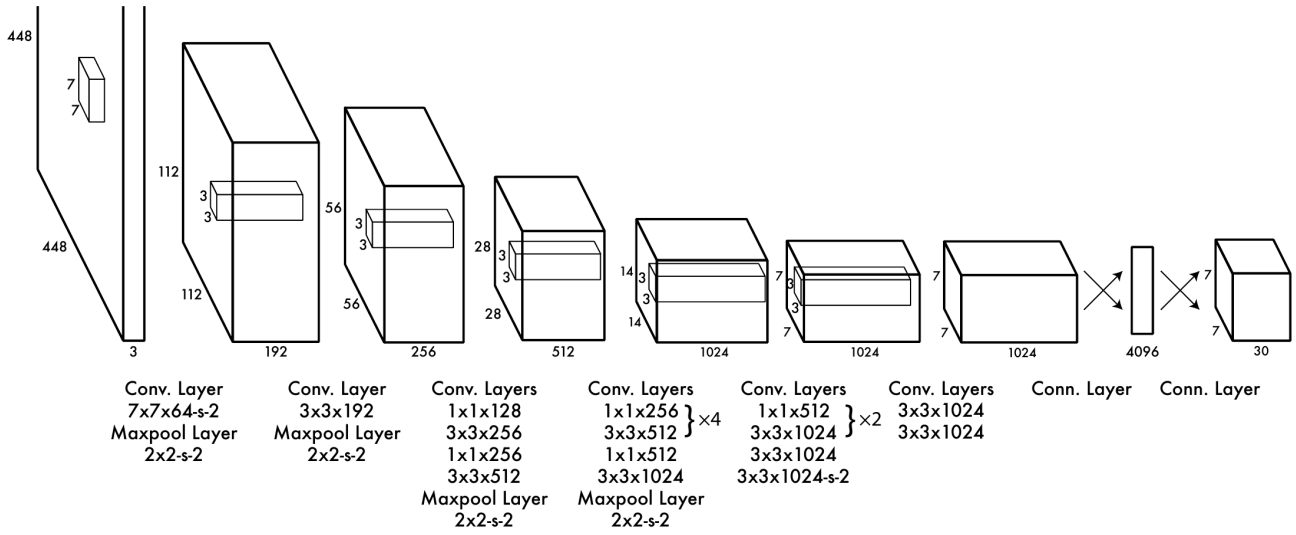


Figure 2.7: Overview of YOLO architecture [28]. It consists of 24 convolutional layers followed by 2 fully connected layers.

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2.
\end{aligned}
\tag{2.29}
$$

In Eq. 2.29, $\mathbb{1}_i^{obj}$ indicates whether there is an object present in grid cell i. It is set to 1 when there is an object. $\mathbb{1}_{ij}^{obj}$ indicates that the bounding box j within grid cell i is responsible for detecting the object. To ensure more weight is given to cells with object and localization parameters, λ_{coord} and $\lambda_{noobject}$ are used and set to 5 and 0.5, respectively. $\hat{x}_i$, $\hat{y}_i$, $\hat{w}_i$, $\hat{h}_i$, $\hat{C}_i$, $\hat{p}_i(c)$ denote the parameters predicted by the network.

YOLO Version 2

One of the shortcomings of YOLO was that it made more localization errors compared to its competitors. Furthermore, YOLO had a smaller recall. To overcome these shortcomings YOLO Version 2 (YOLOV2) [30] was introduced in 2017.

Batch normalization was used in YOLOV2 to boost the speed of learning and stabilize the parameters learned. For more discussion on batch normalization see Section 2.1.1.

Another idea used in YOLOV2 is increasing the resolution. In YOLOV1, the feature extraction network was trained in 244×244 and then the resolution was increased to 448×448 for the detection task. In YOLOV2, in the feature extraction phase, the network is trained 10 more epochs on the higher resolution, therefore the network filters are better adjusted to the higher resolution after which they are passed to the detection phase.

Finally, the idea of prior boxes (anchor box) was introduced in YOLOV2. They are a set of pre-determined bounding boxes. The authors propose that finding a set of good prior bounding boxes will improve the accuracy of the bounding boxes parameters prediction while improving the speed. Therefore, in the preprocessing step, finding a set of anchor boxes with relevant numbers was suggested by running the K-means algorithm [31] on the training dataset bounding boxes dimensions. The authors run an analysis of the number of clusters (the number of bounding boxes) versus average IOU (between the training dataset bounding boxes and pre-determined bounding boxes) and suggest that 5 clusters provide a good trade-off. Figure 2.8 shows the results of this experiment.

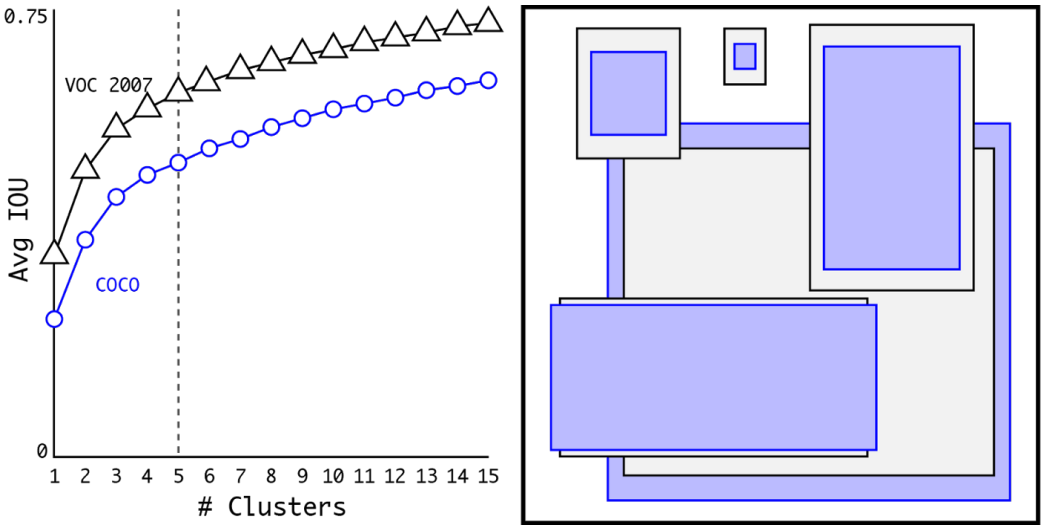


Figure 2.8: Analysis of the number of clusters vs IOU on the COCO and VOC 2007 dataset. Boxes on the right side are the dimensions of anchor boxes for k = 5 for both of the datasets. [30].

YOLO Version 3

YOLO Version 3 (YOLOV3) was introduced in 2018 [32]. The previous version of YOLOV3, YOLOV2, had shortcomings in detecting small objects and speed. In this version, the architecture was tweaked to overcome these shortcomings. In previous versions of architecture, more convolutional layers have been added, and as a general rule, more layers mean better performance, especially in feature extraction [32]. In YOLOV2 the image was downsampled as it was passed to forward layers, therefore fine-grained features were lost, which were detrimental to the detection of small objects. Residual networks (ResNet) [33] introduced skip connections (connecting an activation by skipping layers and connecting it directly to input of one of the next layers) to help the activation functions remain functional in deeper layers, which helps with remedying this problem.

YOLOV3 combined the feature extractor of YOLOV2 and ResNet [33] to obtain a better feature extractor named Darknet-53. The network is built with the structure of 1×1 layers followed by 3×3 convolution layers, with residual block plus a skip connection. See Figure 2.9 for more details about Darknet-53.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.9: Detailed schematic of Darknet-53 architecture [32]. This is a combination of YOLOV2 and ResNet architectures.

YOLOV3 predicts an objectness score (score for presence of an object in the bounding box) for each

bounding box using logistic regression. This score is 1 if the anchor box has the greatest overlap with the ground truth compared to any other anchor box.

In previous versions of YOLO, after training the feature extractor, the parameters for detection were determined in the last layers. However, YOLOV3 makes detections at 3 different scales. The features from the last 3 residual blocks were used for 3 different scale detectors. YOLOV3 makes predictions at 3 scales in layers 82nd, 94th, and 106th, by strides of 32, 16, and 8, respectively as in Fig. 2.10.

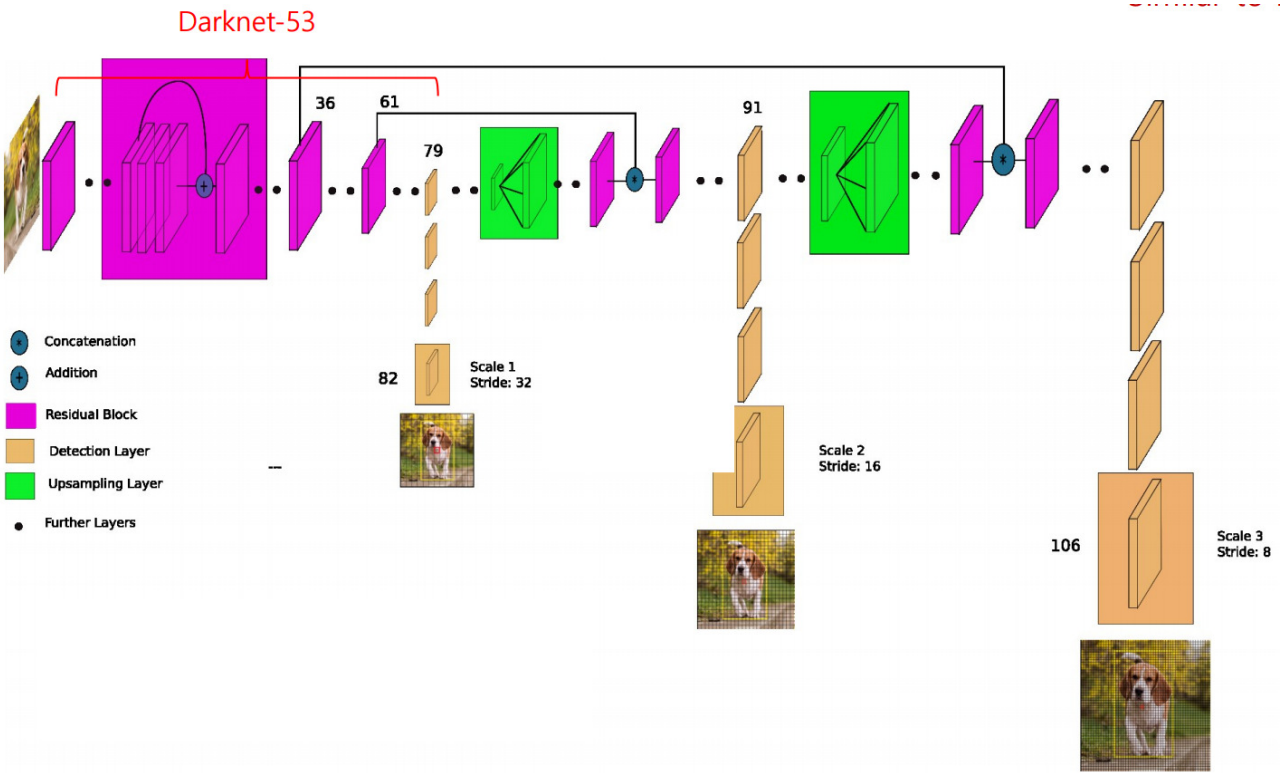


Figure 2.10: YOLOV3 Architecture [34]. YOLOV3 makes detections at 3 different scales.

The first detection is made by the 82*nd* layer. For ease of interpretation, the input image with a resolution of 416×416 is passed through the first 81 layers. At layer 82, it is downsampled by the stride of 32×32 and the resultant feature map would be of size 13×13 grid cell. In previous versions of YOLO, detection of small object had an adverse effect on the performance. At each detection layer, the detection is done by applying 1×1 detection kernels on feature maps. This is done again at further layers resulting in 26×26 and 52×52 feature maps. This helps to remedy the problem of detecting smaller objects as a bigger feature map is more suitable for detection. The overall architecture of YOLOV3 is depicted in Figure 2.10.

2.1.5 YOLO Version 4

YOLO Version 4 (YOLOV4) [35] provides a more robust model by integrating the latest innovations in computer vision, particularly in data augmentation, into the previous versions. Object detectors consist of two parts, the backbone and the head. Typically, the backbone is pre-trained on a larger dataset of image classification and extracts features to obtain a feature map. The head is tasked with the prediction of objects and bounding box parameters. The neck, which aggregates feature maps from different stages of the network, is another feature introduced in YOLOV4. Figure 2.11 shows these two parts.

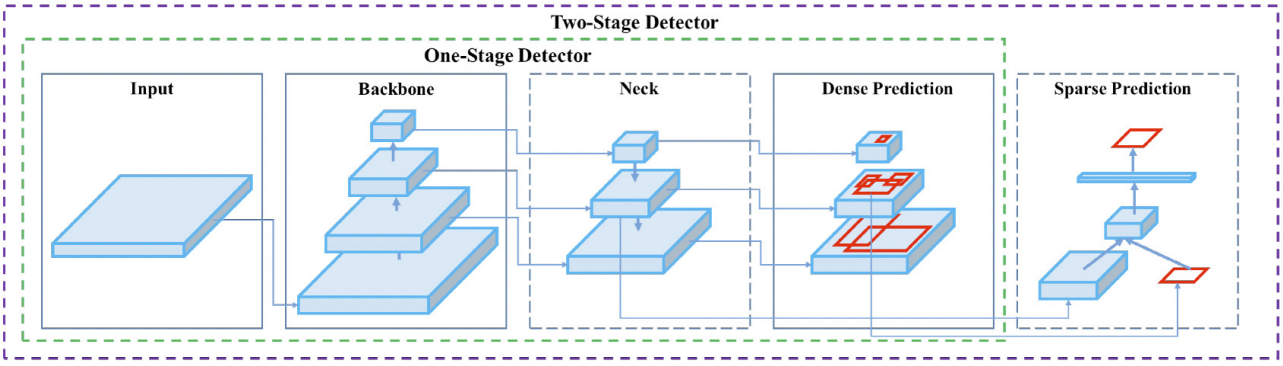


Figure 2.11: Overview of Object Detection Architectures Concepts [35]. Object detectors consist of two parts, the backbone and the head. The head consists of a dense prediction part and sparse prediction part.

Various architectures were considered for the backbone of the YOLOV4 as feature extractors. Cross stage partial (CSP) Darknet53 [36] was found to be optimal. CSP Darknet53 is inspired by DenseNet [37] architecture. The main concept behind DenseNet is to connect each layer to its previous layer. This would help with the learning in deep layers from earlier layers, propagating gradients and features through the whole network. The architecture has two main building blocks: dense blocks and transition layers. A dense block consists of fully connected layers where the input of each layer is the concatenation of the output of its previous layers. Transition layers are used to downsample the feature maps. See Figure 2.12 for a pictorial description of these two layers.

CSP (Cross Stage Partial) [36] works the same way as DenseNet, except that instead of using the full-size input feature map as the base layer, the input will be divided into two parts. As usual, a portion of the data will be forwarded through the dense block and another portion will be sent straight to the next dense layer.

CSP Darknet53 integrates these ideas with Darknet-53 architecture in YOLOV3, with residual blocks being replaced by dense blocks. As a result, CSP Darknet53 preserves features through propagation,

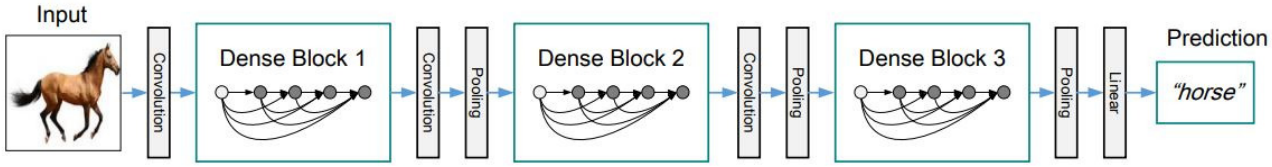


Figure 2.12: Dense blocks and transition layers interconnection [37]. A dense block consists of fully connected layers where the input of each layer is the concatenation of the output of its previous layers. Transition layers are used to downsample the feature maps.

encourages the reuse of features, and shrinks the number of parameters.

Additionally, the paper collects training methods that it classifies as "bag of freebies" (BoF) and "bags of specials" (BoS). BoFs are training methods that influence only training strategies or training costs. BoS, on the other hand, is a training strategy that increases inference cost by a small amount but also provides potential increases in model performance. Mosaics data augmentation and Self-adversarial training are among the novel methods found to be effective in YOLOV4.

In mosaics data augmentation, 4 different images are combined with certain weights. Thus, the model can identify objects on a smaller scale. Mixing images with different contexts also gives the model detection adaptability in different settings. See Figure 2.13 for an illustration of mosaic data augmentation.

Figure 2.13: Mosaics of images [35]. They are created by combining 4 images together with specific weights.

The Self-Adversarial Training technique [38] utilizes a two-stage forward-backward process to augment

data. Instead of altering the network weights, the neural network alters the original image in the 1st stage. Therefore, the neural network creates the illusion that there is no desired object on the image by executing an adversarial attack on itself by altering the original image. In the 2nd stage, the neural network is trained to detect an object on this modified image in the normal way.

YOLO Version 5

Glenn Jocher and his team [1] published a new version of the YOLO family, called YOLO Version 5 (YOLOV5) [39], one month after YOLOV4. The YOLO models were developed on a custom framework called Darknet by Alexey Bochkovsky, which is largely written in C. Glenn Jocher is a researcher and CEO of Ultralystics LLC. Using a Python language framework called PyTorch, Ultralystic converts previous versions of YOLO into one of the most popular frameworks in deep learning.

From a theoretical point of view, there is not much difference between YOLOV5 and YOLOV4, also the author of YOLO5 did not publish any paper. However, they provide a code repository, implementing the model in PyTorch. PyTorch is widely used among the computer vision community, so it gives more flexibility for deployment. The codebase provides 5 different models (YOLOV5 nano or YOLOV5n, YOLOV5 small or YOLOV5s, YOLOV5 medium or YOLOV5m, YOLOV5 large or YOLOV5l and YOLOv5 extra large or YOLOV5x) with different numbers of parameters, which can be used according to the problem size, computational budget and performance desired.

2.1.6 R-CNN

R-CNN stands for region proposal-based convolutional neural networks. It was first introduced by Ross Girshick et. al. [40]. R-CNN consists of three modules. The task of the first module is to create region proposals. These are regions in the image where the network proposes an object presence, irrespective of its class. In R-CNN, a selective search algorithm has been chosen to find the proposals.

It starts by segmenting the image into every minuscule object or object part and then grouping the segments based on their similarities. The similarity is measured as the linear combinations of colour similarity, texture similarity, size similarity and shape similarity.

The next module is feature extraction. In the output of the last module, the region of proposal is warped into a tight bounding box. Since AlexNet [41] is used as the feature extractor, the input size must be

[1] https://github.com/ultralytics/yolov5

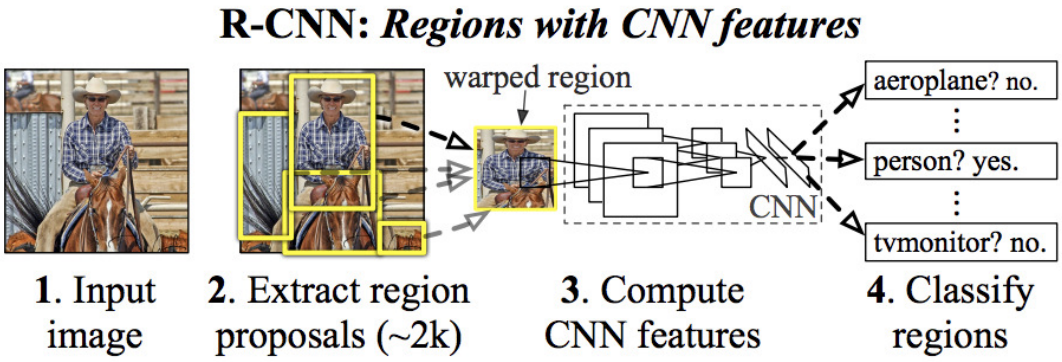


Figure 2.14: R-CNN Architecture workflow on a single image [40]. At the first stage, it extracts region proposals, then extracts features.

227×227. After passing the image through the feature extractor, a $4096 - dimensional$ feature vector is obtained for each region of proposal. Figure 2.14 shows the architecture of R-CNN.

In the final module, an SVM algorithm [42] is used to score each feature vector map. An SVM classifier is trained for each class present in the dataset and after running the feature map on the SVM, scores above a certain threshold are counted as positive examples.

Fast R-CNN

The process of finding 2000 regions for each image, generating a feature network for each region, and running SVM algorithm for detecting each object involves a vast amount of computation. In a situation where there is little sharing of computations R-CNN will be highly computationally expensive.

Gerschik [43] modified R-CNN by passing the entire image through CNN, instead of passing each region, to obtain the feature network. A spatial pyramid pooling network (SPPnets) was proposed for obtaining a feature map of the image, therefore increasing the amount of shared computation. After a feature map is obtained for the entire image, for each object proposal, a fixed-length feature vector is obtained using the region of interest pooling layer. Feature vectors are fed into a series of fully connected layers. Finally, the output is branched into two layers, one layer is used for detecting the class and the other layer is used for estimating the parameters of the bounding box. Figure 2.15 shows the overall architecture of fast R-CNN.

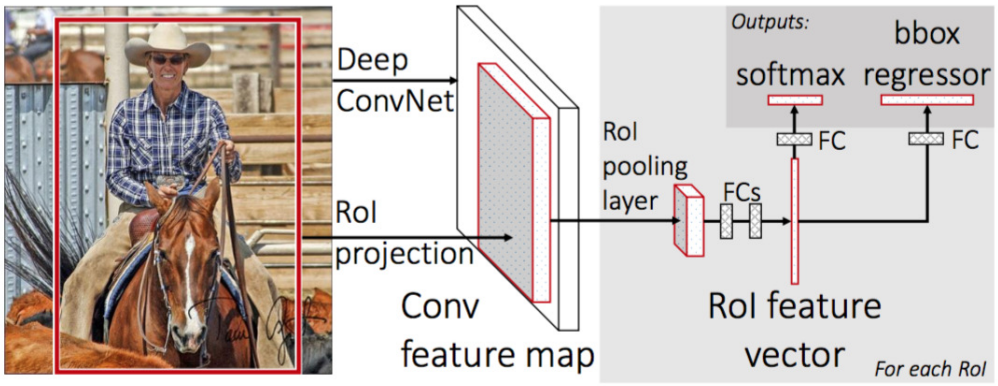


Figure 2.15: Fast R-CNN Architecture [43]. R-CNN was modified by passing the entire image through a CNN, instead of passing each region, to obtain the feature network.

Faster R-CNN

Although Fast R-CNN had improvements compared to its predecessor, it was still slow. The main bottleneck was using a selective search for region proposals. Faster-RCNN [44] unifies the region proposal network into the CNN architecture.

After passing the input image through a classifier, a feature map is obtained. The feature map is then passed through the region proposal network (RPN). A set of anchor boxes with different sizes and aspect ratios are already predefined. The RPN task is to determine two scores for each anchor box at each spatial place on the feature map, whether there's an object present and to estimate the bounding box of the object present. After, a Fast R-CNN is trained on the proposed regions to complete object detection.

2.1.7 Feature Pyramid Networks

Lin et. al. introduced feature pyramid networks for object detection in 2017 [45]. Feature pyramids are used for detecting objects at different scales. Feature pyramid network (FPN) is a topdown architecture with lateral connections to construct high-level semantic feature maps at multiple scales. This architecture resulted in significant improvement as a generic feature extractor in several applications, including on the Common Objects In Context (COCO) detection benchmark. Figure 2.16 shows the architecture of Feature Pyramid Networks.

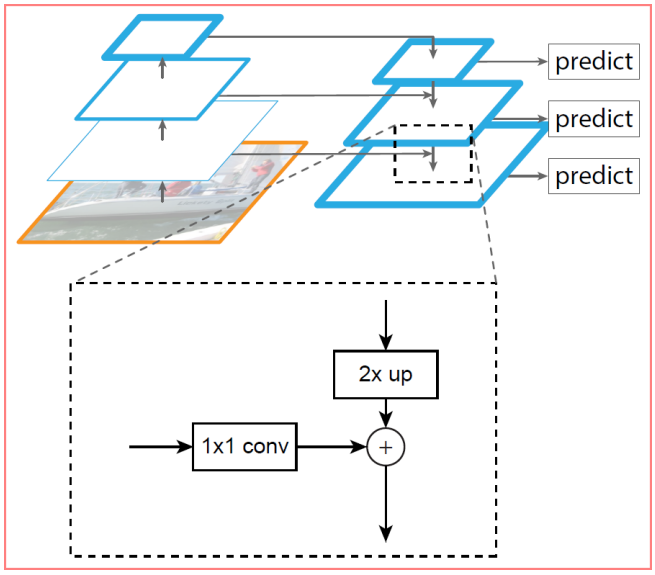


Figure 2.16: Architecture of Feature Pyramid Networks. The dotted lines shows the process of merging feature maps, after upsampling the smaller feature map and reducing the size of bigger feature map by applying a convolution filter.

2.1.8 Generative Adversarial Networks

Goodfellow et. al. [46] introduced generative adversarial networks (GANs) in 2014. There are two components to a GAN, a generator and a discriminator. The generator generates "fake" samples to mimic data from the real dataset, and the discriminator distinguishes between the samples from the dataset and those from the generator. In an adversarial learning model, these two networks are trained together so that the generator improves its ability to generate samples and the discriminator improves its ability to differentiate between real and fake samples.

This can be formulated as optimization problem

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1 - \log D(G(z))]. \tag{2.30}$$

In Eq. 2.30, z is a random vector sampled from a prior distribution $p(z)$. $\mathbb{E}_{z\sim p_z(z)}$ denotes the expected value (EV) of z over the distribution $p_z(z)$. z is fed into the generator, G, with the task of mapping the input to the other domain. The discriminator, D is tasked with differentiating between the real sample and the generated sample by G.

When the input of the discriminator is real-world data, the objective of the discriminator is to maximize $logD(x)$, outputting 1. Whereas when the input of the discriminator is noise, the objective is to maximize the function $(1 - \log D(G(z)))$, output 0. See Figure 2.17 for the general workflow of a GAN.

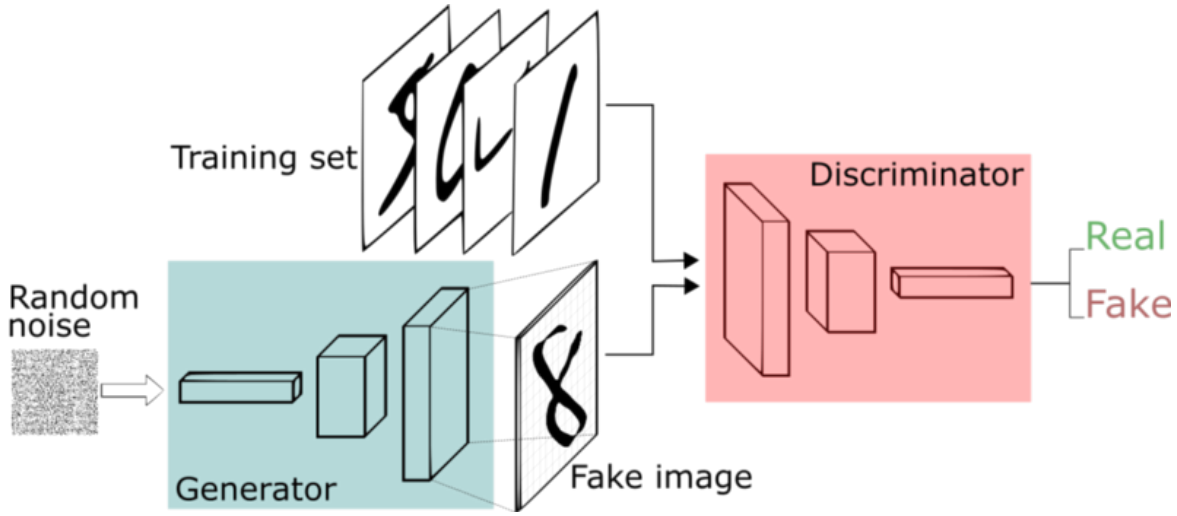


Figure 2.17: General workflow of GAN [47]. It consists of two parts, generator which generates fake samples and discriminator which differentiates between the real and fake sample.

Contrastive unpaired translation (CUT)

One of the most important applications of GANs is translating from the domain of one image to another, such as translating images of zebras to horses. Creating paired datasets that map every sample to another one is expensive, therefore using unpaired datasets is more reasonable in practice. Contrastive unpaired translations aim to translate one image from a domain to another domain while preserving the desired contents.

The problem is defined as translating input images from domain A to images in domain B. Domain A can be considered as the X, set of images of domain A and Y is the set of images of domain B. The generator is divided into two parts, the G_{encode}, and G_{decode}. The encoder learns to capture domain invariant features, such as the body of a horse, grass and sky while the decoder learns domain-specific features, such as stripes of the zebra.

Samples of patches of the input image feature stack are compared to patches of the feature stack of the generator, emphasizing that the corresponding patch pair are similar while the other pairs of the patch are considered negative. With this in mind, v, v^+ and v^- are defined as patches stacked as vectors. v is the patch sample from the output of the generator, v^+ is the corresponding patch and v^- is the negative patches.

L layers from the G_{encode} are chosen, the feature maps of these layers are passed through a multi-layer perceptron H_l, producing a stack of features $\{z_l\}_L = \{H_l(G^l_{encode}(x))\}_L$, where G^l_{encode} represents the output of the $l-th$ chosen layer. Layers are indexed,$l \in \{1, 2, ..., L\}$, and denote $s \in \{1, 2, ..., S_l\}$ where

S_l is the number of spatial locations in each layer. Corresponding feature is $z_l^s \in \mathbb{R}$ while the other features are $z_l^{S\setminus s} \in \mathbb{R}^{(S_l-1)\times C_l}$ where C_l is number of channels at layer l. Similarlly, the output image is encoded as $\{\hat{z}_l\}_L = \left\{H_l(G^l_{encode}(G(x)))\right\}_L$

The final patch loss is defined as in Eq. 2.31.

$$\varsigma_{PatchNCE}(G,H,X) = \mathbb{E}_{x\sim X}\sum_{l=1}^{L}\sum_{s=1}^{S_l} l(\hat{z}_l^s, z_l^s, z_l^{S\setminus s}). \tag{2.31}$$

In 2.31, z_l^s and $\hat{z}_l^s$ are the positive examples while $z_l^{S\setminus s}$ is considered to be the negative example and X is the input image.

The same idea can be used when using other images from the dataset as negative examples defining an external loss Eq. 2.32.

$$\varsigma_{external}(G,H,X) = \mathbb{E}_{x\sim X,\tilde{z}\sim Z^-}\sum_{l=1}^{L}\sum_{s=1}^{S_l} l(\hat{z}_l^s, z_l^s, \tilde{z}_l). \tag{2.32}$$

The final loss function would be as in Eq. 2.33.

$$\varsigma_{GAN}(G,D,X,Y) + \lambda_X\varsigma_{PatchNCE}(G,H,X) + \lambda_Y\varsigma_{PatchNCE}(G,H,Y). \tag{2.33}$$

Chapter 3

Plant detection literature review

This chapter reviews the literature and state of work done in plant detection and plant data augmentation using generative adversarial networks. Although, as far as we are aware, there has not been any work done in the area of augmenting and transforming indoor images of plants, more specifically, to create a dataset that can be used for object detection purposes. Here, we review the works done in the area of plant data augmentation for purposes such as disease detection, classification and leaf counting. This section provides evidence of the effectiveness of generative adversarial networks in data transformation for agriculture.

Zhu et al. [48] utilized conditional deep convolutional GAN (cDCGAN) [49] to augment a small set of orchid seedlings images to determine the seedling's vigour rating. The original dataset was partitioned into a dataset with 100 and 200 images. They added 700 and 600 synthetic images to the first and second datasets, respectively, which resulted in both datasets having 800 images. Training ResNet models on non-augmented datasets of sizes 100 and 200 images, the resultant models achieved accuracies of 0.625 and 0.855. After augmentation and training on new ResNet [33] models led to improved accuracies of 0.945 and 0.955.

Ubbens et al. [50] developed 1000 synthetic images of plants for a leaf counting model. Synthetic plant images were generated using an L-system-based plant simulator module. There are 285 real images in this dataset, which were split into 120 and 165 images, in the Ara2012-Canon and Ara2013-Canon subsets, respectively. These images differ in leaf count, lighting, camera zoom, and other parameters. Comparing the augmented training dataset with the 2013-Canon dataset, training a CNN on the additional 1000 synthetic images resulted in approximately 0.27 less mean absolute count error. Moreover, CNN's trained totally on synthetic data generalized better than those trained on the Ara2012 dataset and tested on the Ara2013-Canon dataset (and vice versa).

Giuffrida et al. [1] designed a GAN that generates synthetic images of the plant to augment a leaf counting dataset. Using the computer vision problems in plant phenotyping (CVPPP) 2017 LCC dataset, the GAN is trained on Arabidopsis plant images from the A1, A2, and A4 subsets . A leaf-counting model was trained on a subset of the A4 dataset (464 images) and an additional 57 images created by GANs. They evaluated the models on the A4 test set. The model trained on the non-augmented dataset had an average difference in the counting of 0.147, an average absolute difference in the counting of 0.942, and a mean square error of 1.865, and an R2 value of 0.947. Based on the same evaluation metrics, the model trained with the additional images obtained scores of 0.186, 0.891, 1.595, and 0.955, respectively.

Zhu et al. [51] utilized a Conditional GAN to synthesize 500 additional images of plants from the A4 subset of the CVPPP 2017 LSC dataset with 624 images. The Mask RCNN model was trained to count the number of plant leaves. Using the 500 additional synthetic images produced by the GAN improved the metric error by 16.67

Madsen et al. [52] used GAN to generate a synthetic dataset of seedling species from the segmented Plant Seedlings Dataset (sPSD). They used the synthetic dataset to pre-train a ResNet-101 model for classification purposes. After 175 training epochs, the classification accuracy of the model converged to 0.9 when trained on real images from the PSD. Pretraining on synthetic images (10000 per class) and fine-tuning on real images led to 0.9 classification accuracy after only 75 iterations.

Miao et al. [53] generated synthetic maize images for the purpose of leaf counting. 4633 real maize images were gathered and annotated. An additional 3655 images were generated using the maize module for Plant Factory Explorer. In comparison to models trained on a similar number of real images, the group found that CNN networks trained on synthetic images received poorer R2 and root mean square errors. However, combining real datasets with synthetic images often resulted in an improvement of both metrics.

Kuznichov et al. [54] developed an algorithm to generate synthetic plant images by placing segmented leaves on an image background to create new plants. They obtained segmented leaf images using arabidopsis and tobacco images from the A1, A2, A3, and A4 subsets of the CVPPP 2017 LCC dataset. To perform the leaf-counting task, a Mask-R-CNN model pre-trained on the COCO dataset is trained on synthetic images and evaluated on the A1, A2, A3, A4, and A5 datasets. Except for the A1 dataset, which met the state-of-the-art standards, other datasets, performance metric surpassed the reported metric in the CVPPP Leaf Counting challenge at the time authors wrote the paper.

Abbas et al. [55] generated synthetic tomato leaf images for disease detection by using GAN. The group used images from the PlantVillage which consists of 16012 total images. They generated 4000 additional

synthetic tomato leaf images. The classification accuracy of the DenseNet model for 5-, 7-, and 10-class classification experiments on the test set was 0.9816, 0.9508, and 0.9434, respectively. Test accuracy improved to 0.9951, 0.9865, and 0.9711 for 5-, 7-, and 10-class classifications when synthetic images were added the training process.

Tian et al. [56] use an improved version of YOLOV3 to increase the accuracy of detection of growing apples in orchard during different growth stages. They propose to change the architecture by using DensNet instead of the original transfer layers to enhance feature propagation. The input layer image size is doubled to increase the feature extraction phase.

Jun et al. [57] optimized YOLOV3 model to improve the detection of location and category of disease in tomato crops. They used a self-made dataset of tomato crops with 12 common diseases and a dataset size of 150K. They modify YOLOV3 by adding feature fusion that increases the number of feature maps, adding two additional residual connections. The improved version outperforms all the other settings by two percent in accuracy.

Tian et al. [58] use YOLOV3 in conjunction with CycleGAN to enhance the detection of apple lesion. First, they augment their dataset with traditional augmentation methods and generative adversarial methods to create synthetic images of apple lesions. They propose using DenseNet for better feature propagation at different scales. They improve the reported performance metric compared to the second-best performing model, which is the original YOLOV3.

Chapter 4

Outdoor plant identification, datasets and experimental settings

For localization and classification of plants in an outdoor setting, such as an outdoor agriculture field, we need models that are trained on datasets with similar features and distributions. This chapter provides detailed explanations of the construction of datasets that were explored for this purpose.

First, natural datasets (i.e. real-world datasets) are introduced. Natural datasets are collected with imaging devices and cameras, capturing samples of real-life data in diverse settings such as a growth chamber and outdoor agriculture field. The outdoor data is not annotated, and annotating it is labour-intensive and expensive therefore, it cannot be used to train a supervised model. On the other hand; indoor data lacks visual similarities such as soil background, shade and folds and changes in texture that are present in an outdoor setting. In this chapter, we apply different data augmentation techniques (image processing techniques and generative adversarial networks) to natural data to create synthetic outdoor data that is annotated and visually similar to outdoor data. We also annotate a small subset of outdoor data to evaluate the performance of the models trained on different datasets. We use the indoor dataset which is an annotated dataset of plants in a green house setting. We use this dataset since the process of generation and annotation of it is fully automatic. We also use the outdoor dataset, this dataset is not annotated. We use this dataset since our aim is to have a detection model in an outdoor setting. As for synthetic datasets, we use a composite dataset created through image processing techniques and synthetic datasets generated with generative adversarial algorithms.

4.1 Indoor dataset

This thesis is built on the indoor laboratory dataset from the TerraByte group [1] at the University of Winnipeg. The dataset contains more than 1.2 million labeled images of 14 different kinds of crops and weeds common in Canadian prairies and in the US [59]. This dataset includes images of plants from a variety of angles and capturing different stages of their growth in a greenhouse setting with high resolution.

EAGL-I [9], an automated system consisting of cameras mounted on a gantry moving autonomously, was used to collect and automatically label the data in a greenhouse setting. During various stages of plant growth, this robotic system can capture and label images with high resolution, capturing a wide range of visual details about a plant. Figure 4.1 shows the EAGL-I system operating in the chamber growth. This dataset contains master images, containing multiple plants in one single shot with a blue background shown in Figure 4.2a. As the coordinates of the plants are already known to the system, cropped images of single plants are easily obtained as depicted in Figure 4.2b and accurate labels can be attached to the image.

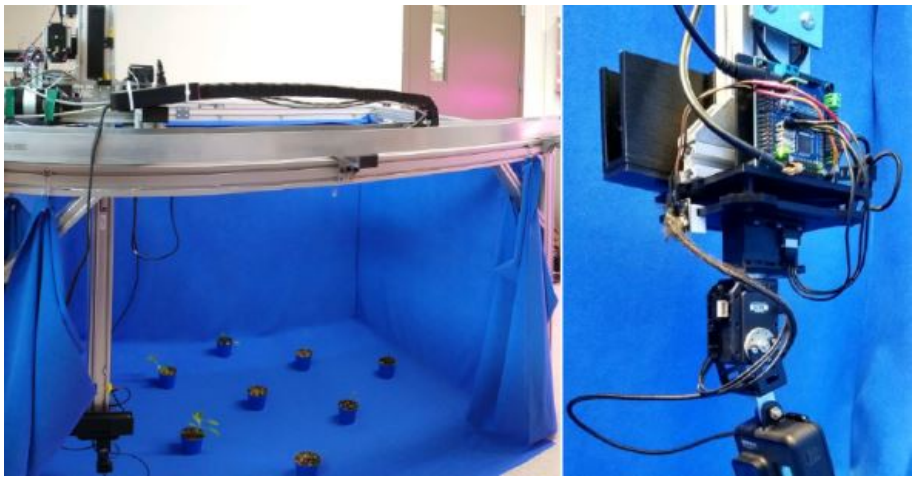
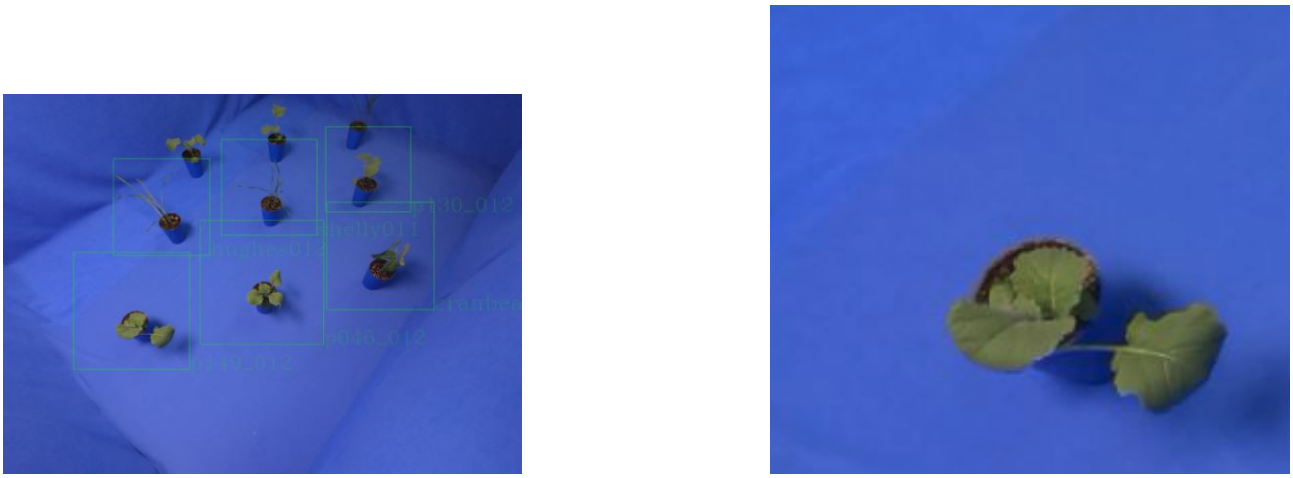


Figure 4.1: EAGL-I mounted on the gantry. The figure on the right shows the system working in the chamber.

[1] https://terrabyte.acs.uwinnipeg.ca/



(a) Master Image. **(b)** Single plant image.

Figure 4.2: The indoor dataset contains master images with multiple plants together and single plant images cropped from the master images.

This thesis focuses on soybeans and canola images. We have images of soybeans and canola during different stages of growth in both outdoor and indoor habitats. We can leverage this to apply data augmentation techniques to the indoor images of these plants and later on the outdoor data and evaluate the performance of the object detection models trained on the generated synthetic data.

4.2 Outdoor dataset

We use the outdoor dataset collected by the TerraByte group. A tractor-mounted camera was used to collect outdoor data. As the camera passes through the field, it records a video. Afterward, the image dataset is created by sampling frames from the videos. The dataset contains images of canola, soybean, wheat, oat, and fava bean. The field was divided into distinct parts, and only one crop is cultivated in each part. Every few days, data is collected to keep track of each plant's growth. The tractor usually travels one full circuit of the field as part of the collection process. There are multiple instances of each of the crops in most of the images with possible weeds growing among them. Figure 4.3 shows examples of outdoor dataset. We don't know the exact coordinates and number of instances of each crop within each image.

(a) Soybean outdoor image.

(b) Canola outdoor Image

Figure 4.3: Samples of outdoor data. The figure on top is the soybean in outdoor setting and on the bottom is the canola in outdoor setting.

Within the scope of this thesis, we used the data collected on June 9 and June 13 of the year 2020. The ability to detect crops when they are at this stage is critical since the plants and weeds are competing for the resources necessary for their growth.

4.3 Validation outdoor dataset

An annotated dataset matching the outdoor data condition was necessary in order to measure the performance of each model, as the purpose of training is to have a model that can localize and classify plants available in the outdoor data.

We chose a subset of the outdoor dataset and annotated it. Images captured from June 13 that were not used elsewhere in this thesis were used to form the outdoor validation dataset. Images of canola and soybean taken on June 13 were sampled with a sample rate of 50 (choosing 1 image out of every 50 image). This resulted in 130 canola images and 158 soybean outdoor images. Images were annotated by hand using Roboflow [60], , which is an online platform used to annotate datasets. We chose canola and soybean since they are very similar in appearance and it would be harder for the model to differentiate between them. After annotation, we had in total 1103 instances of canola and 672 instances of soybeans. This introduced an imbalance between two different classes. We rebalanced the dataset by omitting some of the samples. This resulted in 710 instances of canola and 672 instances of soybean. Table 4.1 summarizes the properties of indoor, outdoor and validation outdoor dataset.

Natural Datasets			
Name	Size	Classes	Annotated
Indoor	1.2 million	14 Classes	Yes
Outdoor	500k	Canola, Soybean, Fava bean, Oat, Wheat	No
Validation outdoor	238	Canola and Soybean	No

Table 4.1: Natural datasets properties.

4.4 Composite dataset

It is extremely costly and time-consuming to obtain annotated images of outdoor plants, as discussed earlier. The outdoor data background is soil as in Figure 4.3. The first step to augment the indoor images, with the aim of making them similar to outdoor data, is to replace the blue background with a soil background. Single indoor images can be placed onto a soil background to create a dataset that better mimics outdoor conditions. For this purpose, we gather 20 soil images. In addition, we sampled 177 images of single plants consisting of soybean and canola (129 soybeans and 49 canola). Since all of the outdoor dataset images are taken from a top-down angle, we select only s with the top-down angle.

Blue backgrounds are used for all indoor data, including single plant images. The strong contrast of the blue background makes thresholding a highly effective tool for identifying plants [9]. As a result, we can create a new dataset by manipulating the plants in a different setting. We name images that are generated by this method the Composite data.

The first step is to convert a single indoor image from RGB space to CIELAB (Commission Internationale de l´Eclairag laboratoire) color space. The blue background can be removed in this space by setting a threshold, which results in keyed-out plants. Figure 4.4 shows picture of a single indoor canola and its corresponding mask that can be used to remove the blue background [61].

In the next step, to better match the color of plants in outdoor data, single plant images are color corrected. Specifically, this is done by calculating the weighted average of the pixels in a specific channel, where the weights correspond to pixel values in the masked image. Each image is randomly scaled, then padded to match the background size. Figure 4.5 shows samples of composite dataset. Figures 4.7 and 4.6 depict the flowchart and process of generation of this dataset.

Figure 4.4: Single indoor image of canola and its corresponding mask.

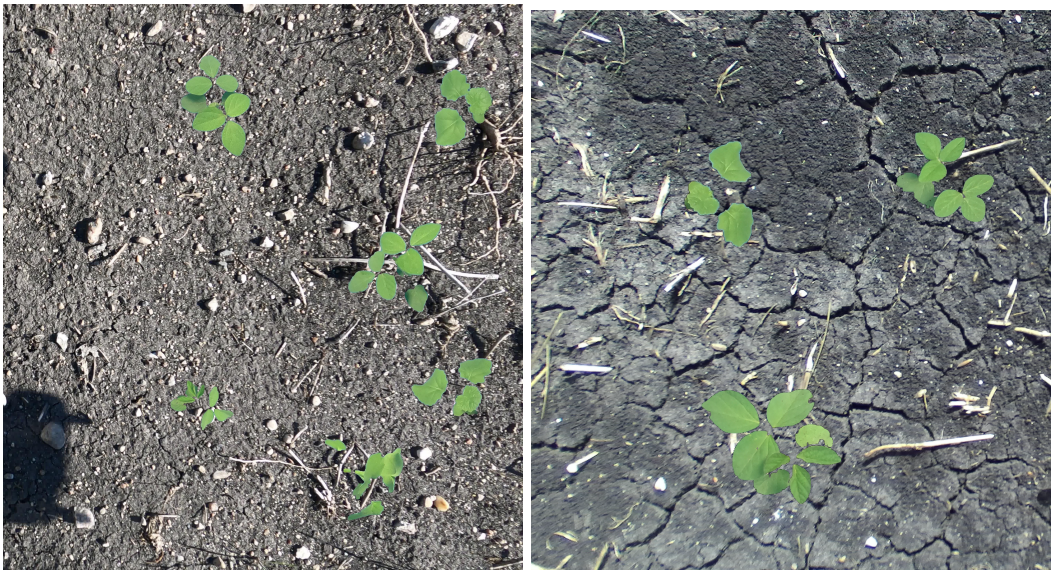


Figure 4.5: Samples of the Composite dataset. Separating leaves from blue background and placing them on soil background.

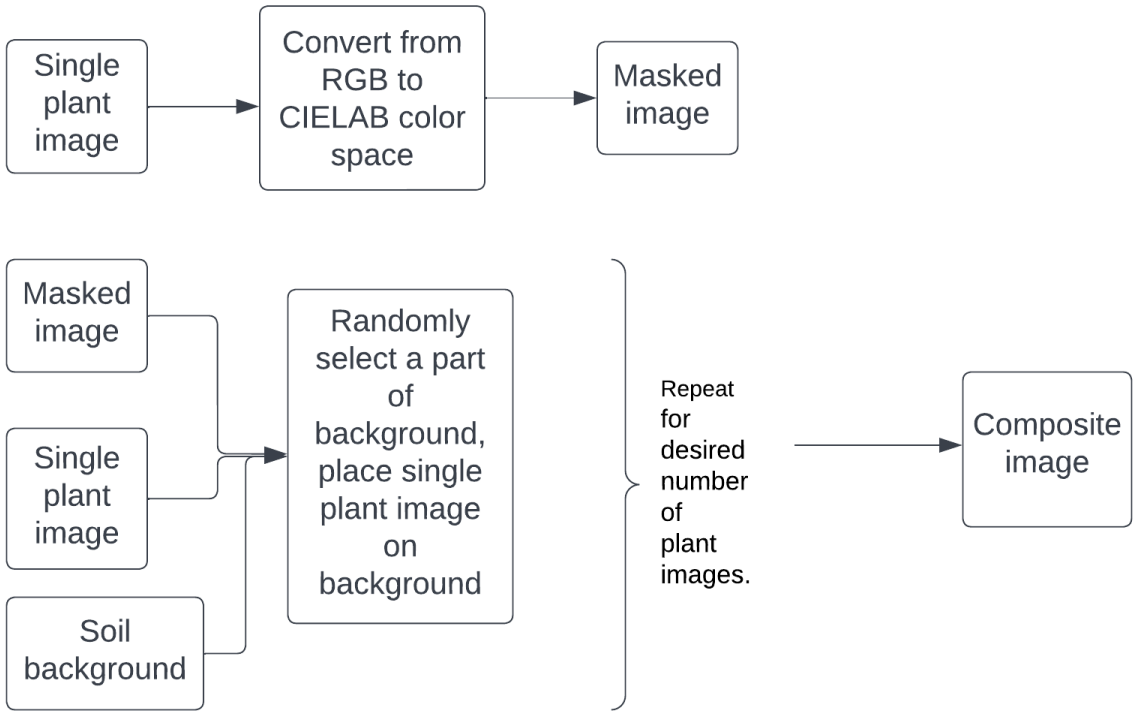


Figure 4.6: Flowchart of composite data generation. Obtaining masked image and single plant image and placing them on soil background.

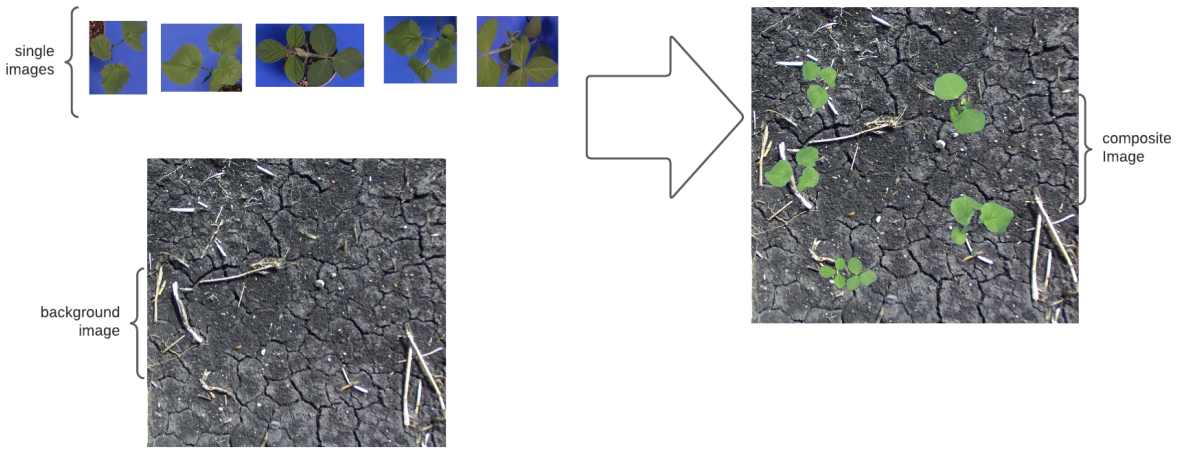


Figure 4.7: Composite dataset generation. Placing single plant images randomly on a soil background.

4.5 Synthetic 1 dataset

Although the composite dataset is one step closer to generating an annotated dataset of outdoor conditions, it still has shortcomings, mainly that it doesn't capture how outdoor conditions affect the plants. Plants growing outdoors are susceptible to direct sunlight, as well as weather conditions like wind and rain. This can cause features such as leaf deconstruction, tears, or color changes. Figure 4.8 illustrates an example of the non-monotonicity of the color, tears on leaves, folds and shades of the soybean in outdoor setting versus indoor setting.

(a) Soybean indoor **(b)** Soybean outdoor

Figure 4.8: Soybean in indoor setting versus outdoor setting.

In this case, a dataset that captures these characteristics better is needed. To generate this dataset, we

need a model that translates single composite images into single outdoor images. Generative adverserial networks have shown success in domain translation [62].

Contrastive unpaired translation (CUT) [63] (see Section 2.1.8), a GAN network that can translate images from one domain to another domain is used for this purpose. This model is trained to translate single composite images to single outdoor images. Figure 4.9 shows the overall workflow of Synthetic 1 dataset.

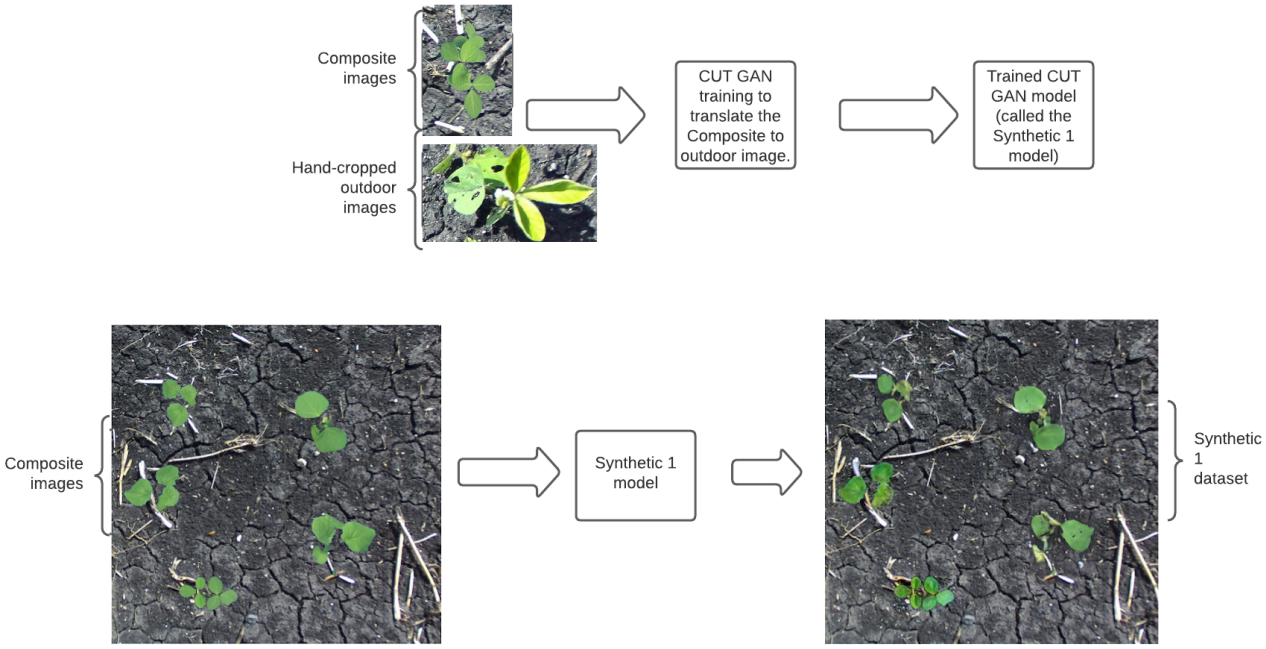


Figure 4.9: Synthetic 1 dataset generation. Top row: Figure depicting CUT GAN training process, where the input is both composite images and hand-cropped outdoor images. This process created the Synthetic 1 model. Bottom Row: using the Synthetic 1 model to translate from the composite domain to the outdoor domain.

For this purpose, 64 single composite images, constructed from single indoor images are gathered with 64 single plant images from the outdoor dataset. This is done for four different crops: soybean, canola, wheat and oats which results in 512 images in total. Single plant images from the outdoor dataset are cropped manually, hence the limited size of the outdoor single plant images. The generator is trained on this dataset using CUT [63] to translate images from domain A (single composite images) to domain B (single outdoor images).

Once we have a generator model capable of translating images from composite to outdoor setting, we can use multiple composite images to create new augmented images. Since the position of each plant within the composite image is already known, we can crop the single plants, feed it to the trained GAN, which outputs the corresponding translated plant into outdoor setting and place the translated image into the cropped position. We consider the dataset generated in this way as our Synthetic 1 dataset. Figure 4.10 shows samples of Synthetic 1 dataset.

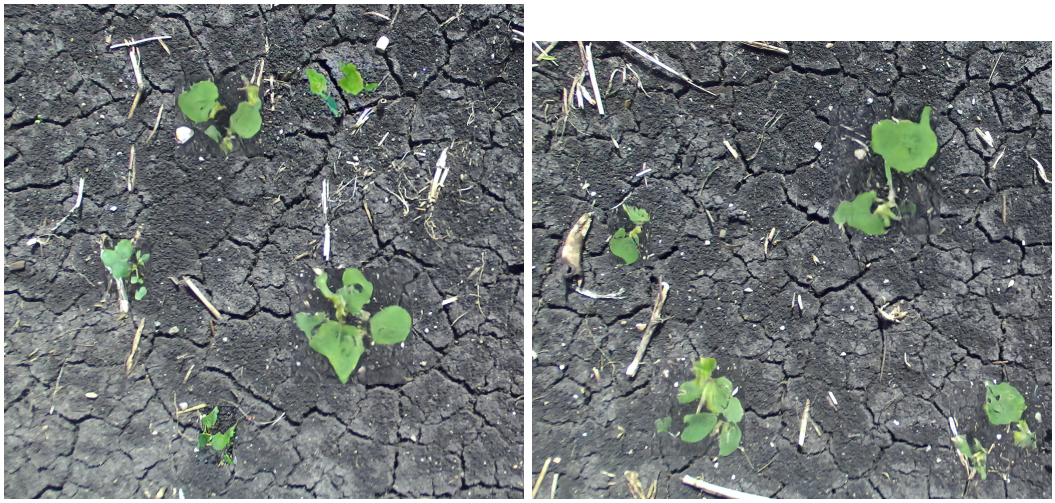


Figure 4.10: Samples of the Synthetic 1 dataset. They are generated by translating the Composite dataset samples to outdoor, using the CUT GAN.

4.6 Synthetic 2 dataset

The Synthetic 1 dataset has some improvements in capturing the outdoor features compared to the composite dataset. In some cases though it fails to properly translate the indoor single plant images and makes them look jittery and distorted as shown in Figure 4.11.

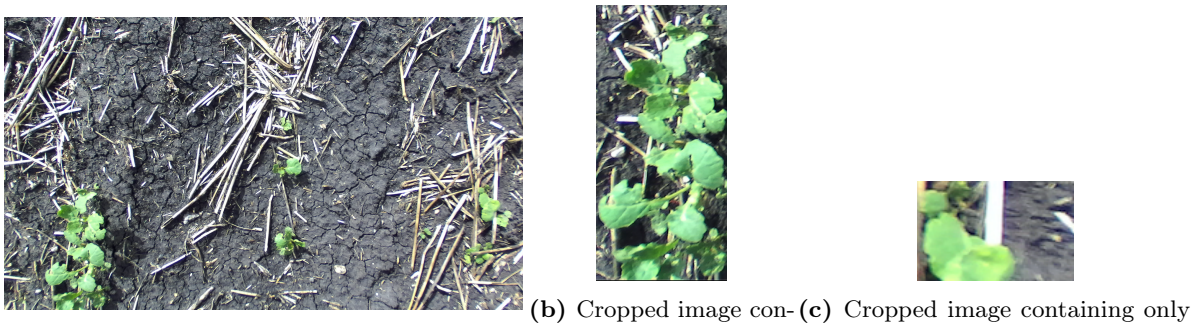
Figure 4.11: Synthetic 1 canola with clear signs of distortion around the leaves.

To improve the generator, first we increased the number of single plant images of outdoor data. Since cropping plants by hand is an arduous task, a plant detection model was trained with the task of detecting plants (identifying plant versus non-plants). The YoloV5 model from Section 2 was used as the

plant detection model. The training data for this model was the Synthetic 1 dataset. For this purpose, a fully synthetic dataset of soybean and canola with 80k images was created.

In the next part, the trained object detection model was fed outdoor images of canola and soybean captured on June 11. Since the outdoor annotated dataset, used for evaluation, is created of images taken on June 13, we chose the date June 11. Plants on this date are at the same stage of growth (2 days difference) yet they are captured in a different day, therefore they are not the same data as the evaluation data but they are similar enough to capture the features we need for training. Detected plants were automatically cropped given the predicted bounding boxes that were detected by the object detection model.

Some of the cropped images did not fully contain the plants such as Figure 4.12c, had multiple instances of plants in them as in Figure 4.12b, had low resolution or they were weed (not canola or soybean). These images were removed as they hindered the GAN model training. After that we obtained 5000 single outdoor images of canola and 4000 single outdoor images of soybean, which is a 10-fold increase in the amount of data available compared to the Synthetic 1 dataset. We used the same generative network architecture as the Synthetic 1, the CUT model. We trained a separate GAN network for each crop (one model for the soybean and another separate model for canola) as opposed to Synthetic 1 training one network for all the four different crops at the same time. This gives the network the ability to learn features inherent to one specific crop and adapt to features that are unique to each species.



(a) An outdoor image.

(b) Cropped image contains multiple plants

(c) Cropped image containing only parts of the plant

Figure 4.12: Samples of cropped images that were removed. The first image from the left is the main image and the right images were cropped from the main image by the plant detection model.

Using these generative models, we translate the composite images into outdoor settings. As the new models are improved after learning new features from their previous generation (due to having access to a larger pool of training data as a result of the first generation of data they produced), we call them Synthetic 2 dataset. Figure 4.13 shows an example where synthetic 1 fails to translate the composite

image of canola and successful translation of the same image by synthetic 2 model, preserving the edges of the leaves, while adding folds and shades. Figure shows the overall process of Synthetic 2 dataset. Table 4.2 shows some properties of synthetic datasets.

(a) Canola (Synthetic 1) **(b)** Canola (composite) **(c)** Canola (Synthetic 2)

Figure 4.13: Single outdoor canola in different synthetic datasets. The canola in the Synthetic 2 dataset is the most realistic among them.

Figure 4.14: [
Samples of Synthetic 2 dataset.]Samples of Synthetic 2 dataset. Placing single plant images generated by GAN on the soil background.

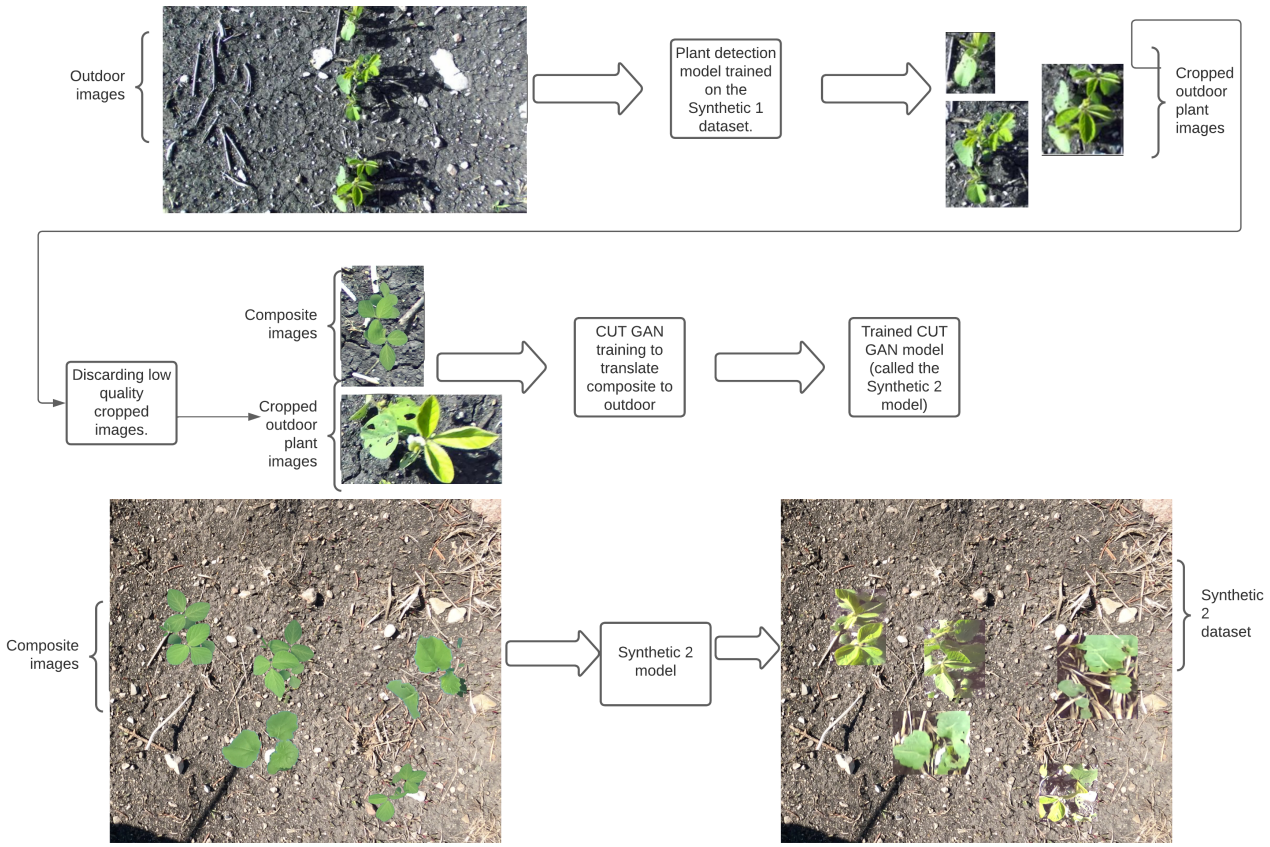


Figure 4.15: Synthetic 2 dataset generation. The top row shows the process of obtaining more cropped outdoor plant images by training a plant detection model using the Synthetic 1 dataset. Middle row shows that after discarding the low quality cropped plant images, a CUT GAN model is trained. The bottom row shows the process of using the Synthetic 2 model to obtain the Synthetic 2 dataset.

Synthetic Datasets		
Name	Dataset size	Built from
Composite	80k	20 soil backgrounds and 80k indoor single plant images
Synthetic 1	80k	Composite dataset (512) and single plant outdoor images (512)
Synthetic 2	80k	Composite dataset (9k) and single outdoor images (9k)

Table 4.2: The Composite dataset was produced by using 20 soil backgrounds and indoor single plant images. This dataset was used to train a CUT GAN (called Synthetic 1 model) to produce the Synthetic 1 dataset. The Synthetic 1 dataset was created using 512 different images from the Composite dataset and 512 outdoor single plant images as input to the Synthetic 1 model. The Synthetic 1 dataset was used to train another CUT GAN model (called Synthetic 2 model). The Synthetic 2 dataset was created using 9k different images from the Composite dataset, and 9k outdoor single plant images as input to the Synthetic 2 model.

Chapter 5

Improved field plant classification using synthetic plant image generation

In this chapter we discuss the experiments we have conducted. In the last chapter, we developed synthetic datasets, attempting to capture the features of outdoor image through annotated synthetic data. Our main goal is to have models that can classify and localize plants in an outdoor setting. In order to assess the usability of synthetic data, we train different object detection models using only synthetic data and evaluate the performance of the trained model on the annotated outdoor data. Since during the training process, none of the outdoor datasets are used, these experiments are a rigorous method to measure how well the synthetic data distribution matches the outdoor data distribution. We experiment with different models and dataset combinations and compare the final results.

We set up our experiments in two settings. The first setting is plant detection. We call these experiments binary problems (plant versus not plant). The model is trained to find all the plants in the image but does not classify plant types. The other setting is when the model is responsible to detect plants and their types. We name these experiments the multiclass problems.

We examine two different object detection models, YOLOV5 and Faster-RCNN to assess the general ability of synthetic data in training across different algorithms. Detailed descriptions of these algorithms are given in Section 2.1.3. In our evaluation, we mainly consider the metrics $maAP.5$ and $mAP.5:.95$. Detailed descriptions of these metrics are given in Chapter 2. These metrics gives us good insight regarding the performance of each setting as they are dependent on both the precision and recall over a variety of IOU threshholds. The problem of localizing an object is highly dependent on the threshold value of IOU of predicted bounding box and groudtruth bounding box. The acceptable threshhold can vary according to the application and desired performance. $mAP.5$ provides a good insight into the performance when this threshold is set to 0.5, while $mAP.5:.95$ is the average value over a set of

varying thresholds $(0.5 - 0.95)$. Finding the optimal value for practical agriculture applications should be studied in a survey which is out of the scope of this thesis.

5.0.1 Experiments settings

All of the experiments in this thesis were run on a Linux Machine, with 4 NVIDIA-Quadro RTX 8000 Graphical Processing Units and an Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz CPU cores. For experiments concerning the YoloV5 model, the PyTorch framework version 1.10.1 was used and for experiments concerning the Faster-RCNN PyTorch [64] version, 1.10.0 were used.

5.0.2 YOLOV5 train technical setting

YOLOV5 provides 5 different models at different scales, (YOLOV5n, YOLOV5s, YOLOV5m, YOLOV5l and YOLOVx). As their names suggest, they have different numbers of parameters. Models with a higher number of parameters have a better ability at capturing features, especially for complex datasets with multiple numbers of objects, therefore these model have more accurate and better performance at detection. On the other hand, more parameters mean more time needed for training and slower inference speed. Figure 5.1 depicts this trade-off for different models, where the x-axis, GPU speed, shows the average inference time per image on the COCO val 2017 dataset on a V100 GPU vs COCO AP Val which denotes mAP@0.5:0.95 on the COCO val 2017 dataset.

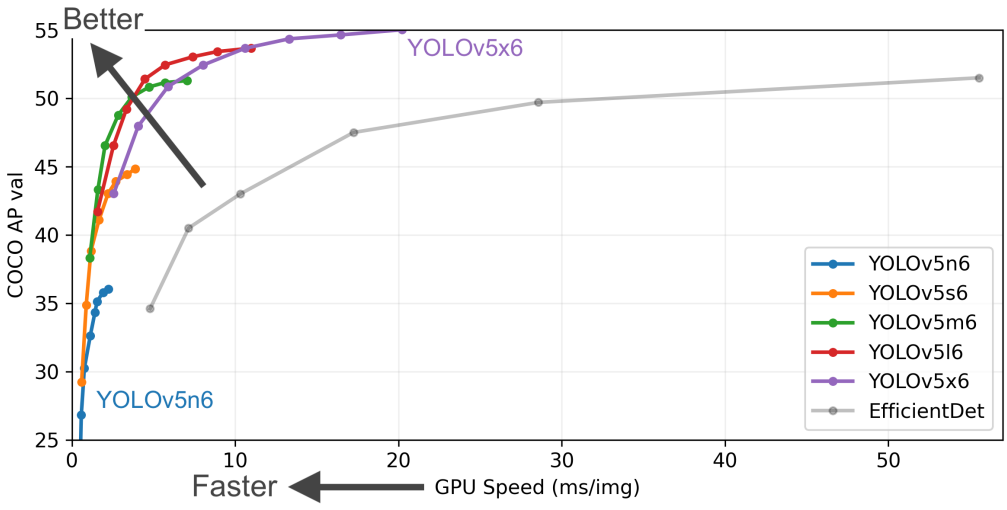


Figure 5.1: Different YOLOV5 models performance and speed trade off [39]. As the number of parameters of a model increases, the average inference speed increases.

Table 5.1 shows the number of parameters for each model. Since all of our experiments have fewer than 4 classes and the variability of the data is limited, we only use the model YOLOV5n as it has a lower number of parameters and is more suitable for less complex problems.

Model	**Number of parameters**
YOLOV5n	1.9M
YOLOV5s	7.2M
YOLOV5m	21.2M
YOLOV5l	46.5M
YOLOV5x	86.7M

Table 5.1: Number of parameters of different YOLOV5 models.

Images in both training and validation steps are resized to 640 pixels on the longest side and the other size is resized by keeping the original aspect ratio of the image.

For all the training purposes, we always start the training with pre-trained weights instead of randomly initializing parameters. This helps us with accelerating the speed of training. All of the models are pre-trained for 300 epochs on the COCO dataset.

All of the training is done using stochastic gradient descent (SGD) as the optimizer, with a learning rate of 0.01.

5.0.3 Faster R-CNN technical settings

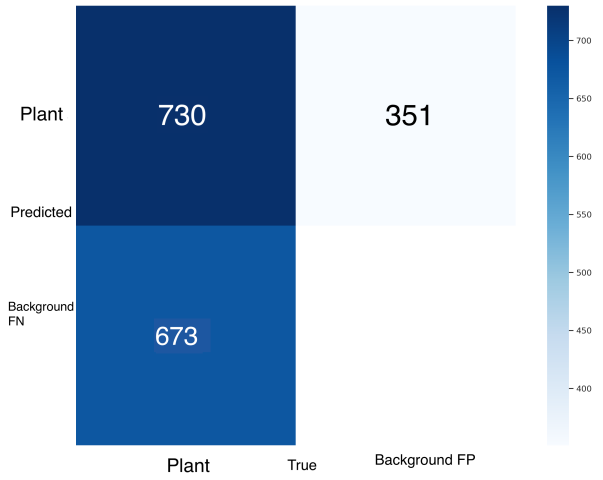
All of the experiments with Faster R-CNN architecture have been conducted in the Detectron2 [65] library. Detectron2 is a library with implementations of R-CNN family architectures. Models are provided with different configurations of backbones, pre-trained networks and learning schedules. We ran all of our experiments using a Resnet with a feature pyramid network (FPN) as the backbone followed by a fully connected layer as the head. The model is already pre-trained with ImageNet. We use the docker container provided by the detectron library.

5.1 Binary Problem

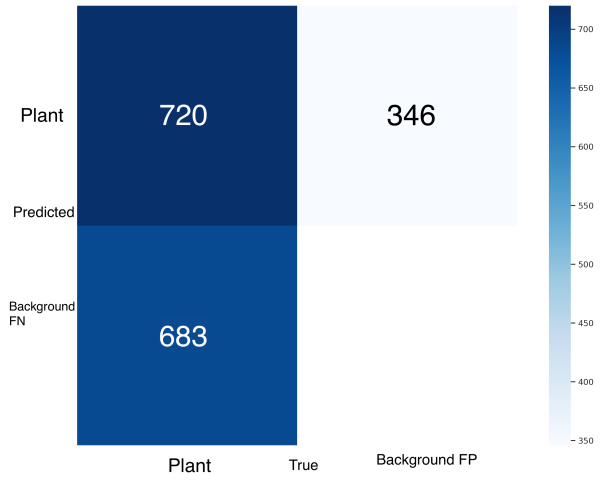
The first step in measuring the effectiveness of synthetic data is to use it for the task of training a plant detection model with the task of locating all occurrences of existing plants, irrespective of their kind or variety. In order to avoid introducing any bias in our evaluation, we only use synthetic data to train and validate the models. Later, once the model is trained, we will use the annotated outdoor dataset to evaluate the performance. We use the composite dataset, Synthetic 1 and Synthetic 2 datasets to train models YOLOV5n and Faster R-CNN. We train all of our models with a batch size of 512. Training is done within 40 epochs. Table 5.2 shows the results for plant detection using the YOLOV5 and Faster-RCNN models.

For the metric $mAP.5$, YOLOV5 trained on Synthetic 1 is performing the best with 0.48. After that YOLOV5 trained on composite comes in the second place with a value of 0.45.

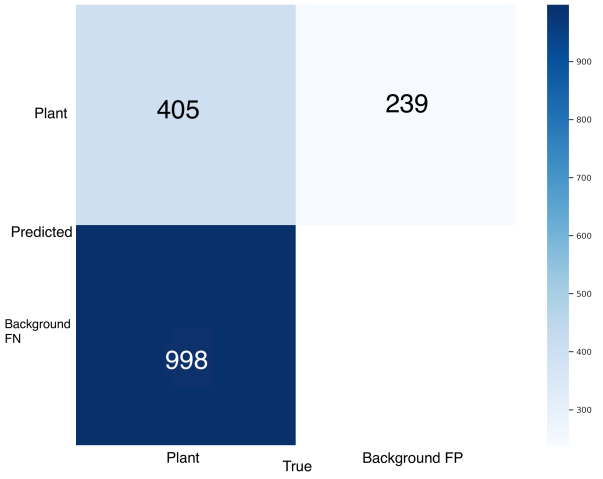
When inspecting metric $mAP.5 : .95$, FasterRCNN trained on composite shows the best performance with the value of 0.23 followed by YOLOV5 trained on synthetic 1 and composite coming second and third with values of 0.21 and 0.20 respectively.



(a) Confusion matrix for the Composite dataset evaluation.



(b) Confusion matrix for the Synthetic 1 dataset evaluation.



(c) Confusion matrix for the Synthetic 2 dataset evaluation.

Figure 5.2: Confusion matrix for the binary problem. The top row are the confusion matrices for the Composite and Synthetic 1 datasets. The bottom row is the confusion matrix for the Synthetic 2 dataset.

Figure 5.2a shows the confusion matrix for the YOLOV5 trained on the composite. 730 instances of plants have been correctly identified. From Figures 5.3b and 5.3c we can see that these numbers are 720 and 405 for Synthetic 1 and Synthetic 2 datasets, respectively. The Synthetic 2 dataset performence in binary class problem is poor compared to other datasets with 998 instances of plant being missidentifed as background.

In overall, we can see that YOLOV5 trained on both the Synthetic 1 dataset and Composite performs better compared to other settings. It should be noted that the images in the outdoor annotated dataset, used for evaluation, have been annotated by a non-expert, therefore there is the possibility of mislabelled data, introducing unwanted noise to the data. There are also instances of weeds growing among the

Dataset	mAP 0.5 (YOLOV5)	mAP@.5:.95 (YOLOV5)	mAP 0.5 (Faster-RCNN)	mAP@.5:.95 (Faster-RCNN)
Composite	0.45	0.20	0.45	0.23
Synthetic 1	0.48	0.21	0.13	0.06
Synthetic 2	0.33	0.10	0.29	0.13

Table 5.2: Binary problem evaluation results.

crops in this dataset. These weeds introduce some noise especially in the binary class problem where we aim to detect all kinds of plants present.

5.2 Multiclass

We trained both the YOLOV5 and Faster-RCNN models on each synthetic dataset. The training subset size for all of the datasets was 60k. At the end of training the model, we validated the updated model on a 10K subset of data. Metrics $maAP0.5$ and $mAP.5:.95$ for each validation were tracked, and we ensured training was stopped only after these metrics had reached a stable value.

Based on Table 5.3, the YOLOV5 model trained on the Synthetic 2 dataset performs better than all the other settings for both $mAP0.5$ and $mAP@.5:.95$ metrics. The improvement for $mAP.5$ is 0.44 compared to 0.25, the second-best performing dataset, which is a significant leap over the Synthetic 1. Tables 5.4 and 5.5 show the results for each individual class. Observing individual classes, we see an improvement of 0.41 from 0.19 for canola. We also see an improvement of 0.46 from 0.30 for the soybean class.

We continue seeing this trend for $mAP@.5:.95$, with an improvement of 0.17 from 0.11 when averaged over all classes, with 0.16 from 0.07 for canola and 0.17 from 0.14 when comparing Synthetic 2 to Synthetic 1.

Table 5.3 shows the results for Faster-RCNN. Composite and Synthetic 2 show similar results in metric mAP0.5 and composite is performing slightly better in metric mAP0.5:.95. We can still see that Synthetic 2 performance is very close to the best performing dataset.

The confusion matrix for the Composite dataset can be seen in the Figure 5.3a. It can be seen that a large number of canola instances, 442, have been detected as soybean. The plant instances trained on

Dataset	mAP 0.5 (YOLOV5)	mAP@.5:.95 (YOLOV5)	mAP 0.5 (Faster-RCNN)	mAP@.5:.95 (Faster-RCNN)
Composite	0.19	0.08	0.31	0.16
Synthetic 1	0.25	0.11	0.18	0.08
Synthetic 2	0.44	0.17	0.31	0.13

Table 5.3: Evaluation results for the multiclass problem.

Dataset	mAP 0.5	mAP@.5:.95
Composite	0.17	0.08
Synthetic 1	0.19	0.07
Synthetic 2	0.41	0.16

Table 5.4: Evaluation results of YOLOV5 trained on synthetic datasets and evaluated on annotated outdoor dataset for canola.

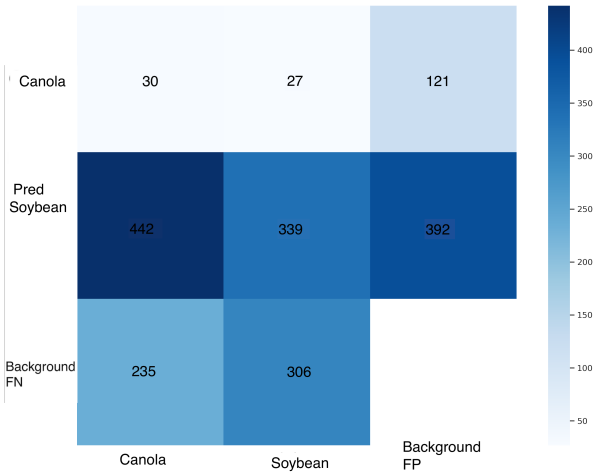
the composite dataset are highly likely to be detected as soybeans. Also, only 30 instances of canola have been correctly detected.

As illustrated in Figure 5.3b, these problems have been somewhat remedied in the Synthetic 1 dataset. The number of canola instances misidentified as soybean has decreased to 262 and the number of canola instances correctly detected has increased to 106.

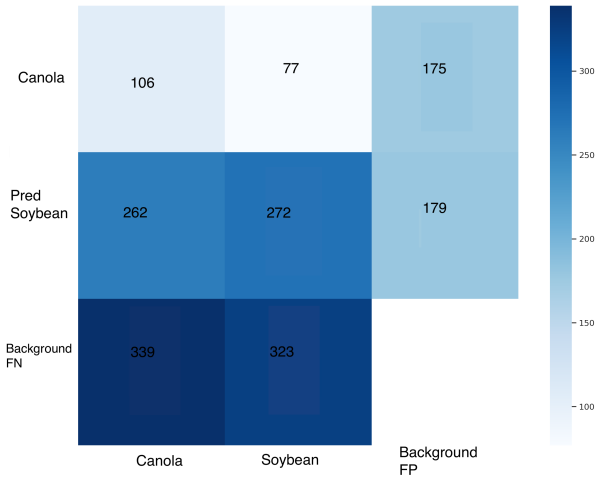
These numbers are improved in the Synthetic 2 regime, as evident from Figure 5.3c. The number of canola instances misidentified as soybean has decreased to 14 and the number of canola instances correctly detected has increased to 324.

Dataset	mAP 0.5	mAP@.5:.95
Composite	0.21	0.08
Synthetic 1	0.30	0.14
Synthetic 2	0.46	0.17

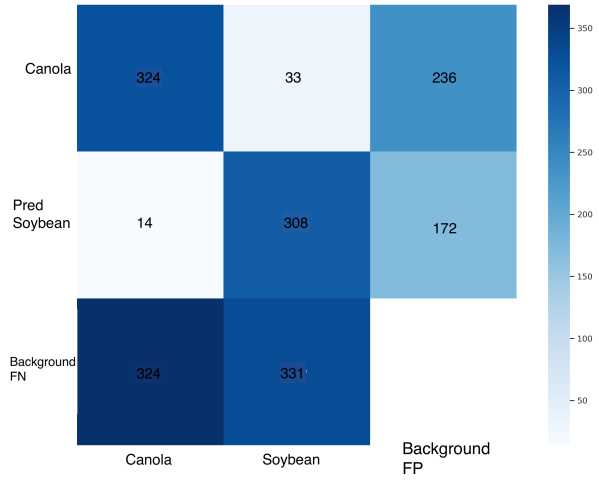
Table 5.5: Evaluation results of YOLOV5 trained on synthetic datasets and evaluated on annotated outdoor dataset for soybean.



(a) Confusion matrix for the Composite dataset evaluation.



(b) Confusion matrix for the Synthetic 1 dataset evaluation.



(c) Confusion matrix for the Synthetic 2 dataset evaluation.

Figure 5.3: Confusion matrix for the binary problem. The top row are the confusion matrices for the Composite and Synthetic 1 datasets. The bottom row is the confusion matrix for the Synthetic 2 dataset.

In summary, we observe that Synthetic 2 datasets perform well, particularly when coupled with YOLOV5 models in multiclass problems. The results are in line with what we expected since we trained separate generative models for each class. Additionally, we have increased the input images of generative models in Synthetic 2 data, which has helped to create more realistic outdoor images.

Chapter 6

Conclusion

The main goal of this thesis was to study the feasibility of using synthetic datasets to train object detection models for the purpose of plant detection in an outdoor setting. We chose two object detection models, YOLOV5 and Faster-RCNN. We defined two problem settings. Binary class problem (plant detection) and multi-class problem (soybean and canola detection). We trained the object detection models on two available datasets, the composite dataset and the Synthetic 1 dataset. We also developed another dataset, Synthetic 2 dataset that boosts predictive performance of plant identification models.

We found promising results in our experiment. In binary setting, Faster-RCNN trained on composite dataset results in mAP.5 of 0.45 and mAP.5:.95 of 0.23. In the multiclass setting, YOLOV5 trained on the Synthetic 2 dataset resulted in mAP.5 of 0.44 and mAP.5:.95 of 0.17. Considering the scale of our experiment, these results show great potential for further refinement and expansion of the dataset created by the generative adversarial networks.

6.1 Future work

This thesis provides great results, however, they were conducted in a limited setting. In order to obtain a more conclusive result, experiments involving datasets with more classes of plants are necessary. It would be potentially useful to increase the number of soil background images, the number of plants during different stages of growth, and the number of outdoor single plant images for the GAN network.

The size of the evaluation outdoor dataset is extremely small (235 images). Another direction that must be pursued in future is creating a bigger outdoor dataset. One possible approach for creating a bigger annotated outdoor dataset is to use the binary model on the unlabeled outdoor dataset and run the

detected plant through a classifier that has been trained to classify plants into their specific class. If this method is pursued, there should be a detailed study on the accuracy and error rate of this method.

Another direction that is of utmost importance is to study the system in a practical setting. As mentioned in this thesis, the problem of object detection is highly dependent on the threshold of IOU of ground truth and predicted bounding box and the confidence score. In order to deploy this system to practical uses, the acceptable threshold should be carefully studied.

Finally, with recent advances in active learning, employing this concept for generating more datasets should also be considered. Active learning [66] approaches will find the types of samples that accelerate the learning performance and therefore can lead to having synthetic datasets resulting in a more robust plant detection system. It will also give us more insight into what kind of samples are better for the task of training.

Bibliography

[1] Hugo Valin et al. "The future of food demand: understanding differences in global economic models". In: *Agricultural Economics* 45.1 (2014), pp. 51–67.

[2] Francisco Yandun Narvaez et al. "A survey of ranging and imaging techniques for precision agriculture phenotyping". In: *IEEE/ASME Transactions on Mechatronics* 22.6 (2017), pp. 2428–2439.

[3] K Dokic, L Blaskovic, and D Mandusic. "From machine learning to deep learning in agriculture–the quantitative review of trends". In: *IOP Conference Series: Earth and Environmental Science*. Vol. 614. 1. IOP Publishing. 2020, p. 012138.

[4] Francis J Pierce and Peter Nowak. "Aspects of precision agriculture". In: *Advances in agronomy* 67 (1999), pp. 1–85.

[5] Christos A Damalas and Ilias G Eleftherohorinos. "Pesticide exposure, safety issues, and risk assessment indicators". In: *International journal of environmental research and public health* 8.5 (2011), pp. 1402–1419.

[6] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. "Recent advances in deep learning for object detection". In: *Neurocomputing* 396 (2020), pp. 39–64.

[7] Jana Wäldchen et al. "Automated plant species identification—Trends and future directions". In: *PLoS computational biology* 14.4 (2018), e1005993.

[8] Adam Binch and CW Fox. "Controlled comparison of machine vision algorithms for Rumex and Urtica detection in grassland". In: *Computers and Electronics in Agriculture* 140 (2017), pp. 123–138.

[9] Michael A Beck et al. "An embedded system for the automated generation of labeled plant images to enable machine learning applications in agriculture". In: *Plos one* 15.12 (2020), e0243923.

[10] Amir Kalali et al. "Chapter 16 - Digital Biomarkers in Clinical Drug Development". In: *Translational Medicine in CNS Drug Development*. Ed. by George G. Nomikos and Douglas E. Feltner. Vol. 29. Handbook of Behavioral Neuroscience. Elsevier, 2019, pp. 229–238. DOI: https://doi.org/10.1016/B978-0-12-803161-2.00016-3. URL: https://www.sciencedirect.com/science/article/pii/B9780128031612000163.

[11] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[13] Georg Keilbar and Weining Wang. "Modelling systemic risk using neural network quantile regression". In: *Empirical Economics* 62.1 (Mar. 2021), pp. 93–118. DOI: 10.1007/s00181-021-02035-1. URL: https://doi.org/10.1007/s00181-021-02035-1.

[14] Tim Menzies et al. "Chapter 24 - Using Goals in Model-Based Reasoning". In: *Sharing Data and Models in Software Engineering*. Ed. by Tim Menzies et al. Boston: Morgan Kaufmann, 2015, pp. 321–353. ISBN: 978-0-12-417295-1. DOI: https://doi.org/10.1016/B978-0-12-417295-1.00024-2. URL: https://www.sciencedirect.com/science/article/pii/B9780124172951000242.

[15] *Multi-Layer-Perceptron*. https://scikit-learn.org/stable/_images/multilayerperceptron_network.png. (Accessed on 08/31/2022).

[16] "Gradient Descent". In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1765–1766. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_2075. URL: https://doi.org/10.1007/978-3-540-29678-2_2075.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[18] Niall O' Mahony et al. "Deep Learning vs. Traditional Computer Vision". In: *CoRR* abs/1910.13796 (2019). arXiv: 1910.13796. URL: http://arxiv.org/abs/1910.13796.

[19] Neil C. Thompson et al. "The Computational Limits of Deep Learning". In: *CoRR* abs/2007.05558 (2020). arXiv: 2007.05558. URL: https://arxiv.org/abs/2007.05558.

[20] Dominik Scherer, Andreas Müller, and Sven Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". In: *Artificial Neural Networks – ICANN 2010*. Ed. by Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101. ISBN: 978-3-642-15825-4.

[21] *File:MaxpoolSample2.png - Computer Science Wiki — computersciencewiki.org*. https://computersciencewiki.org/index.php/File:MaxpoolSample2.png. [Accessed 31-Aug-2022].

[22] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[23] Federico Peccia. *Batch normalization: Theory and how to use it with tensorflow*. Jan. 2019. URL: https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad.

[24] Roberts G Lawrence. "Machine perception of three-dimensional solids". PhD thesis. 1963.

[25] Alexander Andreopoulos and John K. Tsotsos. "50 Years of object recognition: Directions forward". In: *Computer Vision and Image Understanding* 117.8 (2013), pp. 827–891. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2013.04.005. URL: https://www.sciencedirect.com/science/article/pii/S107731421300091X.

[26] R. Padilla, S. L. Netto, and E. A. B. da Silva. "A Survey on Performance Metrics for Object-Detection Algorithms". In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242.

[27] Rafael Padilla et al. "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit". In: *Electronics* 10.3 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10030279. URL: https://www.mdpi.com/2079-9292/10/3/279.

[28] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[29] Mark Everingham et al. "The pascal visual object classes challenge: A retrospective". In: *International journal of computer vision* 111.1 (2015), pp. 98–136.

[30] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[31] J. MacQueen. *Some methods for classification and analysis of multivariate observations*. English. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967). 1967.

[32] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[33] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[34] *YOLO3.png (1400×778)*. https://miro.medium.com/max/1400/1*d4Eg17IVJ0L41e7CTWLLSg.png. (Accessed on 08/31/2022).

[35] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[36] Chien-Yao Wang et al. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". In: *CoRR* abs/1911.11929 (2019). arXiv: 1911.11929. URL: http://arxiv.org/abs/1911.11929.

[37] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[38] Chia-Jung Chou, Jui-Ting Chien, and Hwann-Tzong Chen. "Self adversarial training for human pose estimation". In: *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE. 2018, pp. 17–30.

[39] Glenn Jocher et al. *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*. Version v6.1. Feb. 2022. DOI: 10.5281/zenodo.6222936. URL: https://doi.org/10.5281/zenodo.6222936.

[40] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[42] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/bf00994018. URL: https://doi.org/10.1007/bf00994018.

[43] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[44] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).

[45] Tsung-Yi Lin et al. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[46] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[47] *GAN workflow(800×348)*. https://cdn-media-1.freecodecamp.org/images/m41LtQVUf3uk5IOYlHLpPazxI (Accessed on 08/31/2022).

[48] Fengle Zhu, Mengzhu He, and Zengwei Zheng. "Data augmentation using improved cDCGAN for plant vigor rating". In: *Computers and Electronics in Agriculture* 175 (2020), p. 105603.

[49] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[50] Jordan Ubbens et al. "The use of plant models in deep learning: an application to leaf counting in rosette plants". In: *Plant methods* 14.1 (2018), pp. 1–10.

[51] Yezi Zhu et al. "Data Augmentation using Conditional Generative Adversarial Networks for Leaf Counting in Arabidopsis Plants." In: *BMVC*. 2018, p. 324.

[52] Simon L Madsen et al. "Generating artificial images of plant seedlings using generative adversarial networks". In: *Biosystems Engineering* 187 (2019), pp. 147–159.

[53] Chenyong Miao et al. "Simulated plant images improve maize leaf counting accuracy". In: *BioRxiv* (2019), p. 706994.

[54] Dmitry Kuznichov et al. "Data augmentation for leaf segmentation and counting tasks in rosette plants". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2019, pp. 0–0.

[55] Amreen Abbas et al. "Tomato plant disease detection using transfer learning with C-GAN synthetic images". In: *Computers and Electronics in Agriculture* 187 (2021), p. 106279.

[56] Yunong Tian et al. "Apple detection during different growth stages in orchards using the improved YOLO-V3 model". In: *Computers and Electronics in Agriculture* 157 (2019), pp. 417–426. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2019.01.012. URL: https://www.sciencedirect.com/science/article/pii/S016816991831528X.

[57] Jun Liu and Xuewei Wang. "Tomato diseases and pests detection based on improved Yolo V3 convolutional neural network". In: *Frontiers in plant science* 11 (2020), p. 898.

[58] Yunong Tian et al. "Detection of apple lesions in orchards based on deep learning methods of cyclegan and yolov3-dense". In: *Journal of Sensors* 2019 (2019).

[59] Michael A. Beck et al. "Presenting an extensive lab- and field-image dataset of crops and weeds for computer vision tasks in agriculture". In: *CoRR* abs/2108.05789 (2021). arXiv: 2108.05789. URL: https://arxiv.org/abs/2108.05789.

[60] *Give your software the power to see objects in images and video*. URL: https://roboflow.com/.

[61] A. E. Krosney et al. *Inside Out: Transforming Images of Lab-Grown Plants for Machine Learning Applications in Agriculture*. 2022. DOI: 10.48550/ARXIV.2211.02972. URL: https://arxiv.org/abs/2211.02972.

[62] Jun-Yan Zhu et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

[63] Taesung Park et al. "Contrastive learning for unpaired image-to-image translation". In: *European Conference on Computer Vision*. Springer. 2020, pp. 319–345.

[64] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[65] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[66] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. "Active learning with statistical models". In: *Journal of artificial intelligence research* 4 (1996), pp. 129–145.