



Channel Islands

CALIFORNIA STATE UNIVERSITY

**Feasibility of Real-Time Weed Detection in
Turfgrass on an Edge Device**

A Thesis Presented to

The Faculty of the Computer Science Department

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by


Student Name:
Ricky MEDRANO

Advisor:
Dr. Jason ISAACS

May 2021

© 2021
Ricky Medrano
ALL RIGHTS RESERVED

APPROVED FOR MS IN COMPUTER SCIENCE



05/12/2021

Advisor: Dr. Jason Isaacs

Date



05/12/2021

Scott Feister (May 12, 2021 14:46 PDT)

Name: Dr. Scott Feister

Date



05/16/2021

Brian Thoms (May 16, 2021 17:36 PDT)

Name: Dr. Brian Thoms

Date

APPROVED FOR THE UNIVERSITY



05/24/2021

Jill Leafstedt (May 24, 2021 15:15 PDT)

Dr. Jill Leafstedt

Date

Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Feasibility of Real-Time Weed Detection in Turfgrass on an Edge Device

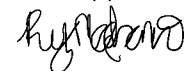
Title of Item

Computer Vision, Deep Learning, Object Detection, YOLO, Jetson Nano

3 to 5 keywords or phrases to describe the item

Ricky Medrano

Author(s) Name (Print)



5/17/2021

Author(s) Signature

Date

Feasibility of Real-Time Weed Detection in Turfgrass on an Edge Device

Ricky Medrano

May 5, 2021

Abstract

Weed control is a challenging issue in turfgrass management. While Precision Agriculture is robust in the literature and production, it is lagging in the Groundskeeping industry. One area that can benefit from Artificial Intelligence is weed management in turfgrass. Object Detection is one of the critical tasks in state-of-the-art autonomous systems. Recent developments in Deep Learning technology and software have allowed significant increases in detection accuracy and speed. A combination of lightweight Convolutional Neural Network models, platform-specific optimizations, and model quantization have contributed to the performance increase seen in object detectors. This thesis explores multi-dimensional trade-offs in Object Detection to assess the feasibility of real-time weed detection on an edge device. Two viable models were discovered that could run in real-time, at least 30FPS, on the Jetson Nano 4GB. These results imply weed detection within turfgrass is feasible using a low-cost energy-constrained edge device and serves as a precursor to an autonomous robotic implementation.

Contents

| | | |
|----------|----------------------------------------------|-----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 3 |
| 1.2 | Scope and Limitations | 5 |
| 1.3 | Objective and Contributions | 6 |
| 1.4 | Outline | 7 |
| 2 | Background | 8 |
| 2.1 | Deep Learning | 8 |
| 2.1.1 | Convolutional Neural Networks | 9 |
| 2.1.2 | Object Detection | 10 |
| 2.1.3 | YOLO | 11 |
| 2.1.4 | SSD and Faster R-CNN | 13 |
| 2.1.5 | Transfer Learning | 14 |
| 2.1.6 | Data Augmentation | 15 |
| 2.2 | CNN Performance | 15 |
| 2.3 | Real-Time Inference Speed | 17 |
| 2.4 | Object Detection Accuracy | 18 |
| 2.5 | Edge Device | 20 |
| 2.6 | Dandelions and Bermudagrass | 22 |
| 2.7 | Related Work | 23 |
| 2.7.1 | Precision Agriculture | 24 |
| 2.7.2 | Robotic Weeding | 24 |
| 2.7.3 | Plant Detection with YOLO | 25 |
| 2.7.4 | Relevant Studies | 26 |
| 3 | Methodology | 28 |
| 3.1 | Dataset | 28 |
| 3.1.1 | Data Collection | 30 |
| 3.1.2 | Data Preparedness and Augmentation | 35 |
| 3.2 | Training | 36 |
| 3.2.1 | Computation Environment | 37 |
| 3.2.2 | YOLOv4 Darknet | 37 |
| 3.2.3 | YOLOv5 Pytorch | 39 |
| 3.2.4 | TLT 3.0 | 40 |
| 3.3 | Inference | 41 |

| | | |
|----------|-----------------------------|-----------|
| 4 | Results and Analysis | 43 |
| 5 | Discussion | 49 |
| 5.1 | Limitations | 50 |
| 5.2 | Future Work | 51 |
| 5.3 | Conclusions | 52 |
| | References | 59 |

List of Figures

| | | |
|----|-----------------------------------------------------------|----|
| 1 | Weed Detection | 1 |
| 2 | Autonomous Mowing Robot | 2 |
| 3 | Sustainable Electric Grounds Equipment | 4 |
| 4 | Robotic Prototype Imaging Platform In The Field | 7 |
| 5 | Jetson Nano Specifications | 21 |
| 6 | High Quality Camera from Raspberry Pi | 22 |
| 7 | Photo of a Dandelion | 23 |
| 8 | Ground Truth Label Size Distribution | 30 |
| 9 | Pleasant Valley Fields | 31 |
| 10 | Camera Platform | 32 |
| 11 | Robot Prototype In The Field | 33 |
| 12 | Lighting Scheme on the Robot Prototype | 34 |
| 13 | Data Augmentation Example | 36 |
| 14 | Weights and Biases Developer Tools | 39 |
| 15 | Example Images in Dataset | 43 |
| 16 | Accuracy vs. Speed of CNNs | 46 |
| 17 | Yolov5s Predictions at 320 Resolution | 48 |

1 Introduction

Weed control is a high cost for Grounds departments at schools, public works, and golf courses. These costs extend to manpower, weed control products, and mechanical resources. These departments must hire dedicated weed control specialists as well as purchase high cost herbicides that are highly regulated. Currently, weeding is performed chemically or mechanically, both by human intervention. Chemical spraying has potential human health hazards, and mechanical removal is a large investiture in labor that can be used elsewhere on more complex tasks. The past few years have brought advancements in Artificial Intelligence (AI) that can transform the Grounds department, similar to what it has done for the agricultural industry. One specific area that could benefit dramatically is weed removal. The implementation of automatic weed detection, as shown in Figure 1, can be a solution to this problem and bringing the Groundskeeping industry into a more efficient modern technological era.



Figure 1: Detection of dandelion using Artificial Intelligence.

Plant recognition and detection has been one of the main focuses in precision agriculture. Being able to recognize a plant is the first step in performing intelligent decisions downstream. Technological advancements in this field have led to lowering production costs, increasing crop yield, and improving overall production efficiency. These comprehensive benefits can also be extended to the Groundskeeping industry. Many tasks within a Grounds department can be overhauled to benefit from technology. One such task that has seen a transition to smart automation is mowing. For example, California State University, Channel Islands (CSUCI) utilizes an autonomous robot mower by Husqvarna to mow its playing fields, as seen in Figure 2. One area that is lacking in technology in the Groundskeeping industry is weed management.



Figure 2: An autonomous mowing robot used on campus at CSUCI.

In order to use intelligent solutions, it has to be deployed on hardware that can support the high computational demands of Artificial Intelligence. The hardware must also support a software stack capable of running AI models. An example of this type of hardware that can solve problems out in the field is an edge device [1]. An edge device is a single board computer

designed to be compact, power efficient, and specialized in its use.

The increase in computing power in edge devices has increased tremendously in recent years. Teamed with a high-quality camera, these devices can process large amounts of visual data. They now can provide a balance within the constraints of size, weight, and power (SWaP); giving rise to their entry into the Deep Learning domain. To be realized in the public's everyday life, AI solutions need to achieve energy efficiency, objective accuracy, and small form-factor. Likewise, these solutions should be fast enough to run in real-time to be practical. While Deep Learning, a subset of AI, is solving the recognition problem, it has had its limitations due to size and complexity. The Deep Learning solutions are computationally complex and require modern hardware solutions to run effectively. They are also very memory intensive due to the nature of their input being images or video. These present challenges to deploy on edge devices due to resource constraints on embedded hardware. What follows is the why, what, and how an edge device could feasibly be used for weed management in the Groundskeeping industry using Deep Learning.

1.1 Motivation

The motivation for this thesis stems from a variety of factors. My undergraduate degree in Biology first introduced me to the floral domain. Afterwards I attained an internship with the National Park Service in natural resource management. Next, I spent eight years working for the Grounds department at CSUCI in various roles. It was here I developed a deeper understanding of weed and horticulture management. Over the years, I noticed sustainable practices and innovative technology were starting to creep into our daily routine to assist in working more efficiently and effectively. For example, most power equipment in the Grounds department at CSUCI is electric, like those seen in Figure 3. I want to continue this ongoing effort of rethinking how a Grounds department manages its efforts and introduce smart technology to achieve these goals.



Figure 3: CSUCI Grounds department employing sustainable electric equipment.

The campus has employed a solar powered autonomous robot mower, to cut grass at its playing fields as mentioned earlier. A conventional mower is loud, has toxic emissions, runs on gas, and can only be operated when an employee is available. The robotic mower is an example of innovative thinking and smart implementation for the Grounds department. It will not disturb classes since it is quiet, and will not contribute emissions or add to fuel costs as it is solar-powered. It can also be run during the night or when employees are out sick or unavailable. My goal is to provide a prototype and Computer Vision model for a similar type of smart robot but one which manages weeds in turfgrass.

The way Grounds departments manage weeds is they must still take the time to spot spray weeds within the turfgrass of its playing fields and other grassy areas. Spot spraying effectively saves on costs as 100% of the herbicide is being applied to 100% of the weeds, however it has many downsides. First, it takes a long time to walk the grassy areas, a heat safety concern on hot days for employees in full Personal Protective Equipment (PPE) such as long sleeve shirts, pants, masks, and heavy boots. Second, the search for weeds is subject to human perception and error. An employee out walking a soccer field will miss many weeds due to path navigation. Lastly, spot spraying by an employee means that they will be subject to spray drift from wind and splashback of the herbicide and is therefore a chemical exposure safety issue.

Grounds departments also have the option of herbigation, which is the method of injecting herbicides through an irrigation system. Herbigation is a speedy process and is great in weed elimination coverage as the whole grassy area will be subject to the irrigation. The downsides of herbigation are that it is costly to install and inject, and wasteful as no turfgrass is uniformly covered in weeds. It is also potentially toxic to the environment and humans - there can be significant runoff and high rates of drift as the herbicide is being broadcast high in the air through the sprinklers.

I want to develop a solution that takes the pros of spot spraying and herbigation and eliminates the cons while also being sustainable. Such a solution was a smart robot framework that could identify and spray weeds in grass that would be cost-effective, autonomous, and run off solar power. To narrow the scope of this into a thesis, I specifically want to create a Deep Learning model that could run in real-time on an edge device for detecting the weedy dandelions in turfgrass. My data and results could then set the framework for implementing an autonomous weed spraying robot. This robot could use the detected weed results with precision microsprayers for effective weed management in turfgrass.

1.2 Scope and Limitations

Specific constraints were used in this thesis to narrow the scope. The research is limited to dandelions and a commonly used type of turfgrass called bermudagrass. The growth habits of both these plants are reflective of growing in a Mediterranean climate.

This thesis is also only limited to the Computer Vision aspect of the weeding robot. It does not provide any mechanical aspects about the robot,

including path navigation, localization, and power requirements. It also does not explain how the robot would remove or kill the weed once recognized. This thesis is to serve as the pre-cursor to all of these.

This thesis is also scoped to Deep Learning only on a Graphical Processing Unit (GPU). While other AI-dedicated accelerator platforms exist, such as a Tensor Processing Unit (TPU) or Field-programmable Gate Array (FPGA), for now they have limited flexibility in the types of Deep Learning models they can support.

Lastly, due to time constraints, only one edge device will be examined. However, many Deep Learning models types and sizes were reviewed on this edge device to provide justifiable comparisons.

1.3 Objective and Contributions

The overall objective of this research is to investigate the feasibility of using a low-cost edge device for detecting dandelions within turfgrass in real-time. Recently in 2019, [2] concludes that real-time detection, meaning at least 30FPS, of broadleaf weeds in turfgrass is a prerequisite for site-specific management of weeds such as the dandelion. The edge device can run on a prototype robot that performs detection in the field without using the cloud. The AI model can be optimized to run in real-time and on an edge device with integration of a camera system. My specific contributions to the research are:

- 1) Provide a prototype robotic framework conducive to collecting in-situ images of dandelions and other weeds in turfgrass.
- 2) Perform comparative experiments by training optimized state-of-the-art lightweight Object Detection models that discriminate dandelions from bermudagrass, using fair comparison guidelines. These models would train on self-acquired data, as seen in Figure 4.
- 3) Evaluate for real-time inferencing speed of each neural network on the NVIDIA Jetson Nano 4GB edge device, and elucidate a feasible model that is optimized based on accuracy and speed.

The work performed in this thesis will be beneficial to future Deep Learning studies investigating the feasibility of deployment on edge devices. This thesis can also assist in providing a Deep Learning framework meant for research to those exploring private avenues of weed detection in precision agriculture and turfgrass management where price, power, and deployability are of concern.

In this thesis, the research question I am asking is: Can an edge device such as a Jetson Nano 4GB detect dandelions in bermudagrass with a mAP of at least 85% and a real-time detection speed of at least 30 FPS?



Figure 4: Collecting images in the field.

1.4 Outline

The rest of the thesis is structured as follows: I cover necessary background material about subsets of AI, edge device justification, and plant details. In the Methodology chapter, including how I trained, deployed, and evaluated each AI model. Next, the Results chapter will show key tables and figures of my findings, including analysis. Lastly, I reiterate the overall thesis aim, highlight key conclusions, expound on problems faced, and discuss future work.

2 Background

This chapter gives a summary of the many sub-fields this research is built upon. These include Deep Learning, Convolutional Neural Networks (CNN), Object Detection, and performing inference. It concludes with a quick summary about dandelions and bermudagrass, and related work in the literature.

Despite their exponential increase in computing capacity and memory, computers still have trouble with performing the most basic human tasks such as object and speech recognition. The brain is unrivaled in its ability to process visual and speech data. For example, while a human can easily discern the events unfolding before them in normal everyday conversation, a computer can struggle to identify basic objects in the scene.

The way a computer begins recognizing objects is by taking an image as input. An image is simply an array of numbers in a specific order to a computer. A computer will then perform some particular operation on those numbers and follow an algorithm to classify an object or objects existing in the image. It can not only tell you what object it detects but where exactly they are too. For example, autonomous vehicles use on-board computers and cameras to detect objects in its environment.

Under the umbrella of Artificial Intelligence in Computer Science is a field called Machine Learning. In Machine Learning, models are trained with data to complete tasks without explicitly being told how to specifically complete the task. These models can be trained with data that is labeled so the model knows what the correct answer is and is referred to as Supervised Learning.

2.1 Deep Learning

A subset of Machine Learning that grew within the field is Deep Learning. Deep Learning allows complicated problems to be solved with large amounts of data using artificial neural networks. It is inspired by the functionality of the brain, where its neurons are modeled by artificial neural networks (ANN). Deep Learning has been used in optical character recognition [3], object detection in self-driving cars [4], and forecasting financial trends [5].

Deep Learning came about when using multiple layers in an artificial neural network (ANN) was developed. While Deep Learning has been around for decades, it did not start to take root and be widely used until the 2000s. In the early 2000s, Computer Vision tasks relied on traditional Machine Learning approaches such as hand-picked features from Gabor Filters that

were discriminated against with Support Vector Machines (SVM) [6]. Deep Learning with images resurfaced in 2012 with the creation of AlexNet and its infamous win of the 2012 ImageNet LSVRC-2012 competition by a substantial margin [7], beating all other state-of-the-art traditional image classifiers at the time. AlexNet was able to excel and win as it finally had an enormous data source with ImageNet. It was optimized to perform matrix calculations within a Graphical Processing Unit (GPU), and no longer relied on hand-engineered features picked from classical Computer Vision algorithms. This the main reason Deep Learning was chosen for Object Detection over classical methods in this thesis.

The way Deep Learning detects objects is the computer will train an artificial neural network on a sufficiently large dataset consisting of images and labels. It will constantly provide feedback of its current performance through a backpropagation algorithm, at which point it will adjust its parameters to increase its accuracy by identifying which features are most important in an image. This Supervised Learning happens by minimizing the cost function and adjusting the ANN weights so accuracy starts to increase and loss eventually converges.

2.1.1 Convolutional Neural Networks

A special type of artificial neural network used in Computer Vision is called a convolutional neural network (CNN). A CNN uses an image or video frame as input and performs convolutional operations with filters. These filters are referred to as the weights of a CNN. The output of convolutional operations on the image creates feature maps to be passed onto subsequent layers in the network.

CNNs are used in 3 domains of Computer Vision: Image Classification, Object Detection, and Semantic Segmentation. Image Classification is the process of giving a label to a picture in an attempt to classify what is in the scene. For example, a classification model could be fed a picture of the beach, and it could output the label "sand". There is also a confidence score between 0 and 1 for how confident the model is in assigning that label. While Image Classification helps classify what is contained in an image, Object Detection adds an extra step of localizing objects in the image with the use of bounding boxes. Not only can multiple objects be detected, but multiple different types of objects can be detected. A popular research area in Object Detection currently is in driverless Artificial Intelligence. A driverless

car must be able to take in multiple objects simultaneously and make intelligent decisions on how to drive the car. Finally, Semantic Segmentation, also known as Object Segmentation, takes Image Classification and Object Detection one step further by producing a pixel-wise mask for each object detected in addition to generation of bounding boxes, labels, and confidence scores. A pixel-wise mask is simply a mask of an object where its boundary is at the pixel level. This localization at such a fine-grain is helpful for satellite imagery, precise decision-making in autonomous driving, and medical imaging. The precision required for detecting dandelions in turfgrass for this thesis is conducive to using Object Detection instead of Semantic Segmentation. This is because Semantic Segmentation requires more calculations for its output and therefore would be slower than Object Detection.

2.1.2 Object Detection

Object Detection is the process of combining recognition with localization in a picture or video. Object Detection plays a significant role in Computer Vision objectives and has been around for the last two decades. Localizing of objects is a crucial task in many applications such as autonomous driving, smart robotics, and video surveillance.

The groundwork for Object Detection was first realized with the Viola-Jones face detection framework in 2001 [8]. The algorithm used a simple approach for Object Detection by using sliding windows of different scales traversing all parts of the image. Next came the use of refined Histogram of Oriented Gradients (HOG) as a feature descriptor in localizing objects [9]. These early object detectors used handcrafted features and were based on Machine Learning. The traditional object detectors peaked when the Deformable Part-based Model (DPM) detector won the VOC-07, VOC-08, and VOC-09 detection competitions [10]. These competitions were used by many researchers to test the performance of their detectors at the time.

Object Detection finally entered the Deep Learning realm with a single convolutional network in 2014 with OverFeat [11]. The authors used a multi-scale sliding window approach and a greedy algorithm to aggregate bounding boxes to increase detection confidence. In addition, they had to attach a regression network at the top of their model in order to predict bounding box coordinates. Their work resulted in winning the ImageNet Large Scale Visual Recognition Challenge 2013 (ILSVRC2013).

With CNNs, Object Detection initially used a two-stage method. In the

first stage, it would generate region proposals. In the second stage, it would classify each proposed region if the response is strong enough. Finally, there are post-processing steps of refining bounding boxes, eliminating duplicate detections, and re-scoring boxes based on other objects in the image. This two-stage method has been employed by well-known R-CNN [12], Fast R-CNN [13], and Faster R-CNN [14] models where the R stands for "Region-Based". In Object Detection, each detector has a backbone. Many feature extraction networks exist that can serve as the backbone.

Next came single-stage methods for Object Detection with Single Shot MultiBox Detector (SSD) [15] and You Only Look Once (YOLO) [16]. These algorithms combine the localization and classification of objects all in one step creating a unified end-to-end learning framework. YOLO was seminal in the field of Object Detection due to reframing it as a single regression problem as going straight from image pixels to bounding box coordinates and class probabilities. While single-stage architectures have shown to be much faster at inference, they have exhibited lower accuracy compared to their two-stage counterparts [17].

The most recent type of object detectors are called Anchor-free. One issue is that they provide less flexibility to leverage large-scale data. Examples of Anchor-free detectors are CornerNet [18], CenterNet [19], Fully Convolutional One-Stage Object Detection [20], and Bottom-up Object Detection [21].

2.1.3 YOLO

YOLO, known as You-Only-Look-Once, is a singular neural network that predicts bounding boxes and class probabilities in one evaluation [16]. YOLO was the first one-stage object detector in the Deep Learning era [22]. While offering state-of-the-art inferencing times, the original YOLO suffered in accuracy compared to two-stage detectors such as Faster R-CNN.

The initial implementation of YOLO works by dividing an image into an $S \times S$ grid where each grid cell is responsible for predicting B bounding boxes and confidence scores. It uses Darknet as the feature extractor part of its backbone [16]. The author also created a faster version of YOLO called Fast YOLO which simply used fewer convolutional layers and fewer filters for those layers.

Since 2016, more iterations of YOLO have evolved. The major drawback of original YOLO was that it made a significant number of localization errors

and suffered low recall. YOLOv2 introduced batch normalization, variable resolution input, anchor boxes, fine-grained features, and an updated Darknet backbone [17].

YOLOv3 improved upon YOLOv2 with a better feature extractor, Darknet-53, as well as using a logistic classifier to calculate the probability of an object belonging to a specific class instead of a softmax function [23]. This change was made to remove the assumption each output only belongs to a single class. Another improvement was changing the last part of the loss function from Mean Squared Error (MSE) to a cross-entropy loss function. These improvements led to similar mean Average Precision (mAP) scores as Single-Shot Detector (SSD) but were 3 times faster on the Common Objects in Context (COCO) dataset [23]. COCO is a large-scale Object Detection dataset used by many researchers to assess their model’s performance.

YOLOv4 was able to further improve accuracy and detection speed by using the CSP (Cross-stage Partial Network) approach of scaling their network [24]. Scaling a network typically involves deepening it by adding more layers or widening by adding more convolutional filters. This approach can lead to higher accuracy but comes at the expense of more computation time, resulting in slower inference. YOLOv4 was scaled through the CSP approach, which partitions the backbone into two parts then merges them back through a cross-stage hierarchy [25]. In addition, it has label smoothing, dynamic mini-batch size for random shapes, grouped convolution, sigmoid scaling, and a new “mish” activation. For data augmentation, it allows one to use Mosaic, CutMix, or MixUp. It is important to point out that the most significant gains of YOLOv4 over YOLOv3 come from accuracy improvements. Speed saw only minimal gains in YOLOv4.

YOLOv5 was introduced in May 2020 by Glenn Jocher, only one month after YOLOv4 [26]. Jocher was not an original author of YOLO but did provide the mosaic data augmentation technique to YOLOv4. YOLOv5 was created to improve accessibility, faster training, faster inference, and easier deployability. Unlike the previous iterations of YOLO, which were compiled with the Darknet framework, YOLOv5 is implemented in PyTorch. Moving to PyTorch was one of the biggest factors in YOLOv5 being easier to set up and configure, whereas Darknet can have many complicated dependencies and is less production-ready. Moreover, YOLOv5 trains much faster due to the PyTorch ecosystem being a more established research framework and using the Python programming language.

YOLOv5 also implements the CSP approach in its network architecture.

It uses PANet for its neck to generate feature pyramids. In an object detector, the neck is where feature maps are collected from different stages. Feature pyramids assist models in generalizing objects at varying scales. This translates to better performance when detecting objects on unseen data. The detection head of the network stayed the same as the one used in YOLOv4 and YOLOv3. It utilizes the Leaky ReLU activation function throughout the network. Anchor boxes are auto-learned based on the distribution of bounding boxes in the custom dataset.

YOLOv5 uses a cosine learning rate (LR) scheduler. Learning rate adjustments are critical in training Deep Learning models. Traditionally learning rate is updated through a step decay in the training process. The issue with this is that these steps are the same at the beginning and the end of training. A cosine learning rate scheduler will take larger steps in the beginning to approach local minima faster, but over the training will gradually decrease as not to cause divergent behavior in the loss function. This cosine decay in the learning rate helps improve accuracy [27]. All these features make it a CNN worth testing in finding a feasible detector that can run on an edge device in real-time.

Currently, YOLOv5 suffers from a couple of drawbacks. It is still under active development and can potentially be buggy as the author is still testing new code and fixing old code. It borrowed the "YOLO" moniker and iterated its version to 5, which was not looked upon favorably in the research community as Glenn Jocher is not an original YOLO author, nor has he published a paper on YOLOv5. Lastly, YOLOv5 is a port of YOLOv4 but with a handful of questionable novel improvements that would garner its iterated title.

2.1.4 SSD and Faster R-CNN

Single Shot MultiBox Detector (SSD) was the second one-stage object detector to emerge from the Deep Learning era. SSD introduced multi-resolution and multi-reference techniques which significantly increased detection accuracy over YOLO at the time. When SSD premiered in 2016, it boasted a 74.3% mAP on the VOC2007 dataset along with a real-time inference speed of 59 FPS on an NVIDIA Titan X [15].

The backbone of SSD is MobileNetV2, a high-performing feature extractor. At the top of the SSD detector are convolutional feature layers decreasing in size which all allow for predictions. This is different than YOLO which

only operates on a single scale feature map. Similar to YOLOv3 and Faster R-CNN, SSD uses anchor boxes to predict bounding box offsets and per-class scores. SSD stands out from those other two detectors in that it applies these default anchor boxes to several feature maps of different resolutions, with the intent of making their detector scale-invariant.

An area that SSD struggles in performance is on smaller objects. This is due to small objects having no representations on the smallest convolutional layers at the top of the detector. Even with this shortfall, SSD is still a very attractive object detector due to its balance of accuracy and speed. It can offer real-time inference at state-of-the-art accuracy, making it an easy choice to test weed detection in this thesis.

The R-CNN model was influential in serving as the base for object detectors that came after it. R-CNN would first extract proposals with a selective search, then extra features with a feature extraction backbone, and lastly classify using a Support Vector Machine (SVM) [12]. Fast R-CNN improved the speed of R-CNN by combining the extraction of features and classification all in a single CNN [13]. Faster R-CNN used a more sophisticated approach by using a novel Region Proposal Network (RPN) that discarded the use of time-intensive selective search, thus improving speed. Despite these improvements, it still suffered from the high computational overhead of its RPN [14]. Its 2 stages give it superior accuracy but comes at the expense of detection speed as more computation is required.

2.1.5 Transfer Learning

It is common for Deep Learning-based object detectors first to be pre-trained on large datasets such as ImageNet [7] or COCO [28]. The reasons for this are a) data collection and labeling can be out of scope and time frame for many research projects, b) generalization is improved using a pre-trained network, and c) it cuts down training time as many features were already learned elsewhere.

The process of using a pre-trained network and then retraining it based on newly introduced classes is called Transfer Learning. Specifically, Transfer Learning is achieved in Object Detection by importing the weights that were optimized on another dataset. The model architecture used to train, for example on MS COCO, must be the same architecture for the model that will be used for Transfer Learning. Not all the weights have to be used. For example, only the weights from the first 10 convolutional layers of a YOLO

model can be frozen and imported into a subsequent new untrained YOLO model.

2.1.6 Data Augmentation

Data augmentation is the process of artificially increasing the amount of data to make a more robust model that is less susceptible to overfitting and making it more accurate. It is a data analysis technique that either creates slightly modified copies of the data or newly created synthetic copies.

For Object Detection, data augmentation means adding more images or video frames to increase the size of the dataset, specifically images used in training. The reason to create augmented images is that it is impossible to capture images that are genuinely reflective of all the real-world scenarios that a CNN is trying to solve. Typical image augmentations would be adding rotations, flips, changing color scale, and blurring. The authors in [15] were able to see an 8.8% improvement in mAP, a common accuracy metric in Object Detection, when training their model using data augmentation techniques.

2.2 CNN Performance

Convolutional Neural Networks consist of many layers, with most of them being convolutional layers. Each of these convolutional layers will have k kernels, also known as weights. A kernel is simply a filter that is used to extract features from the input image using convolutions. The kernel slides over the image performing the dot product within each sub-region of input values. Each of these kernels will be of varying sizes with a width (w) and height (h). The convolutional layer can then transform the input image with width (W) and height (H) and number of channels (C) into $W \times H \times k$ dimensions. The kernels can be calculated individually and simultaneously. This is the reason GPUs are the preferred processor with CNNs as they benefit from the GPU's powerful parallel processing abilities, seeing order-of-magnitude improvements over CPUs [1]. This leads to a decrease in training time and increase in inferencing speed from the convolutional layers. This parallelization is what becomes a deciding factor in choosing an edge device to deploy the trained model to. This is because each edge device will have different underlying architecture where some are more beneficial than others for performing Object Detection. The Jetson Nano, for example, can process

up to 128 kernels in parallel before seeing any significant reduction in frames per second (FPS) [29].

CNNs can have very high accuracy but it comes at the cost of higher computational complexity, which requires a high-performance GPU. This increased accuracy comes from an increase in the number of convolutional layers and kernels, thus making the network deeper and more complex. More convolutional layers and kernels allow the model to learn more discriminative features of the image and therefore be able to detect objects more precisely. However, this increase in complexity of the model means an increase in weights and number of calculations needed to do one forward pass through the network to detect an object. As edge devices lack the performance of a standard high-end GPU such as the NVIDIA Tesla V100, there needs to be a compromise in model depth and complexity in order to perform in real-time. This trade-off relationship between accuracy and speed can be seen in the YOLOv5 models where the smallest model has the fastest speed with the lowest accuracy, and the largest model has the slowest speed but the highest accuracy [26].

A library that can significantly increase CNN performance is Compute Unified Device Architecture (CUDA), developed by NVIDIA. CUDA is a software layer API that allows a developer to perform general-purpose processing on CUDA-enabled GPUs. This allows computer scientists to program the convolutions of all the kernels in a neural network onto the GPU, which has much greater parallel processing power than a CPU. Therefore, it is worth exploring edge devices that had a CUDA-enabled GPU.

TensorRT is also a framework by NVIDIA, written in CUDA, that optimizes Deep Learning models' inference on a GPU. One of the primary functions of TensorRT is providing quantization to models through the reduction in the precision of data types representing the weights and parameters of the models. Traditionally inference is run at single-precision floating-point (FP32) but using half-precision floating-point (FP16) speeds up a forward pass through a network with minimal reduction in accuracy. This is because FP16 precision reduces the number of cache accesses by allowing two times the amount of data to fit in the same cache line. Faster inference can further be achieved over FP16 precision with 8-bit integer (INT8) precision. TensorRT functions in 2 stages, optimization and deployment with their Runtime engine. The optimization step is only performed once, similar to building a program, and then is executed to perform inference with the Runtime engine.

Finally, there exists a deep neural network library that is purported to be

optimized for inference on NVIDIA Jetson line of boards called tkDNN. It is built with cuDNN and TensorRT primitives that can exploit Jetson boards to gain maximum performance [30]. This library was tested, along with TensorRT, to determine the feasibility of real-time inference in my custom trained object detectors.

Choosing a lightweight CNN, and pairing it with a compatible inferencing library, is therefore the important first step in achieving real-time detection on an edge device. Next, the trade-off relationship between model complexity and speed must be explored on specific edge devices to fully determine the feasibility in real-time detection with appropriate accuracy.

2.3 Real-Time Inference Speed

In Object Detection, inference is the process of using a trained CNN to make predictions on previously unseen data. Inference time is typically measured in frames per second (FPS) or milliseconds (ms). Device-specific inference speed is a measurement that starts from the time an image is captured to when a predicted output is made. For research purposes and comparability, inference time can also be measured as execution only, which does not include pre and post-processing steps.

An example for measuring inference speed is using the training platform Darknet’s inference function [31]. Inference speed is dependent on the computational power of either the CPU, GPU, or TPU. When considering Computer Vision applications dealing with low-resource edge devices, faster inference times are needed for real-time detection. The authors of YOLOv4 [32] defines real-time as 30 frames FPS or greater, and therefore will be the threshold attempted to meet in the trained object detectors. While 30 FPS is an arbitrary number, it is a common threshold used by others in the research. Real-time inference is important for this thesis as well since the model will be deployed to an autonomous robot. The faster the model can perform inference, the faster the robot can move and process data. In addition, if the model was deployed to a UAV as well, inference speed becomes even more important.

The total execution time of running an object detector occurs in 3 phases. First is pre-processing where the image or video frame is converted to a proper input required by the network. Second is the actual inference where the image does a forward pass through the network. Lastly is post-processing that involves fine-tuning final bounding box outputs. The total of these 3

phases is the end-to-end latency which is the time elapsed from giving a CNN an input to bounding box determination.

While the scope of this thesis does not involve further post-processing of bounding boxes such as creating $[x,y]$ coordinates for precision spraying, it is important to consider end-to-end latency for achieving real-time inferencing.

To optimize inference speed, there are three main strategies to take into account, outlined in [30]. These are network model design, model compression, and platform. Model design can be configured to help execution and memory latency by reducing the number of parameters in the CNN model. This can be achieved by selecting models with fewer convolutions, fewer kernels, or choosing a smaller input size. Model compression can come in the form of quantization, parameter pruning after training, and knowledge distillation. All these methods can provide a significant increase in the throughput of the CNN model. Lastly, the platform a model is implemented on can significantly effect model performance. Most recently, with the advent of General-Purpose computing on Graphical Processing Units (GPGPU), CPUs have been dominated by GPUs as the platform of choice in Deep Learning applications due to its superior parallelization performance.

2.4 Object Detection Accuracy

In Image Classification, the goal is to determine whether the image belongs to a specific class or not. Its performance will determine two main metrics, its precision and recall. Object Detection is more complicated as the question must be asked whether the image has the correct class, and if it does, was it localized properly. Therefore, it also uses precision and recall, along with Intersection over the Union (IoU).

In the context of this thesis, precision is the measure of successful weed detections out of all detections the model makes. In formula, precision is the number of true positives (TP) out of the number of true positives plus false positives (FP):

$$precision = \frac{TP}{TP + FP} \quad (1)$$

A TP in the context of dandelion detection means the object detector made a prediction about the location of a dandelion and that the prediction was correct. For the prediction to be correct about the location of a dandelion, a specific threshold must be met in bounding box overlap between

the ground-truth label and predicted bounding box, which will be discussed shortly. An FP means the detector made a prediction about the location of a dandelion, but the prediction was incorrect. The following table shows the confusion matrix output based on predictions of an object detector. Note, true negatives (TN) are ignored since it denotes there was a failure to detect an object that does not exist. A truth table on Object Detection accuracy is seen in Table 1.

| Labeled | Predicted | Confusion Matrix |
|----------|-----------|------------------|
| Positive | Positive | TP |
| Positive | Negative | FN |
| Negative | Positive | FP |
| Negative | Negative | TN |

Table 1: Object Detection truth table.

Recall is the measure of successful weed detections taking into account weeds that were failed to be detected. In formula, recall is the number of true positives out of the number of true positives plus false negatives (FN):

$$recall = \frac{TP}{TP + FN} \quad (2)$$

Overall, a high precision indicates a high success rate of successful detection of dandelions when the model makes a prediction. A high recall indicates a high success rate of detecting dandelions with a low failure in detecting target weeds.

As mentioned, IoU is used to determine accuracy in Object Detection. The IoU threshold used in the evaluation of the MS-COCO dataset is 0.5 [28], which will also be used for this thesis. This threshold means anything above it counts as a TP and anything below counts as a FP. IoU calculation is the overlap ratio between the predicted bounding box and the ground truth label over the total area of both the predicted bounding box and ground truth label box:

$$IoU = \frac{BB_{overlap}}{BB_{union}} \quad (3)$$

Currently, the “de facto” metric in object detectors is mean Average Precision (mAP). It is the primary metric used in YOLOv4 [32] and SSD [15],

specifically mAP@0.5. Evaluating mAP@0.5 means measuring the Average Precision at a 0.5 IoU threshold across all classes in the dataset. Recall and mAP@0.5 are the two metrics used for this thesis.

In the case of dandelion detection and precision spraying, recall is not as important as precision. It is better for the robot to miss a weed than it is to detect a weed that is not there, meaning a false positive. This is because the robot will make more than one pass in turfgrass, so it will have more opportunities to make a correct prediction. A mapping algorithm would handle preventing spraying weeds more than once. If it has lower precision and is making many false positives, the robot is spraying for weeds that do not exist. This results in wasted product, wasted power, and reduced efficiency.

2.5 Edge Device

When choosing an edge device to perform Deep Learning, it is important to select those that offer a GPU or TPU. A GPU and TPU both offer superior parallel processing ability over a CPU. Extending the edge device to be used on an autonomous robot creates new requirements to consider, such as power consumption, connectivity, and adaptability.

The Jetson line of devices from NVIDIA offers excellent flexibility for a robot performing Deep Learning. The scope of an autonomous weeding robot in Groundskeeping means it is physically small and is intended to be used in smaller commercial and consumer environments. With these requirements in mind, the edge device used for this thesis was the Jetson Nano 4GB Developer Kit. The Developer Kit option means the Nano comes integrated with a board with connections for USB, Ethernet, HDMI, audio, and power. The Jetson Nano offers the right balance of power, performance, and price, achieving 472 GFLOPS on 5W of power for only \$99. The more advanced Jetson TX2 requires 1.5x more power, while the top-of-the-line Jetson NX requires 2x the power. The Jetson Nano is also suitable for serving as the central computer for a mobile robotic platform as its equipped with a 40-pin header with onboard GPIO, I2C, I2S, UART, power, and PWM. Specifically, the Jetson Nano 4GB is NVIDIA's entry-level embedded device of their Jetson line of products. It is a System on Chip (SOC) with a Quad-core ARM CPU, 128-core Maxwell GPU, 4GB of DDR4 memory, and ability to attach 4 cameras [33]. The Jetson Nano 4GB is shown below in Figure 5.

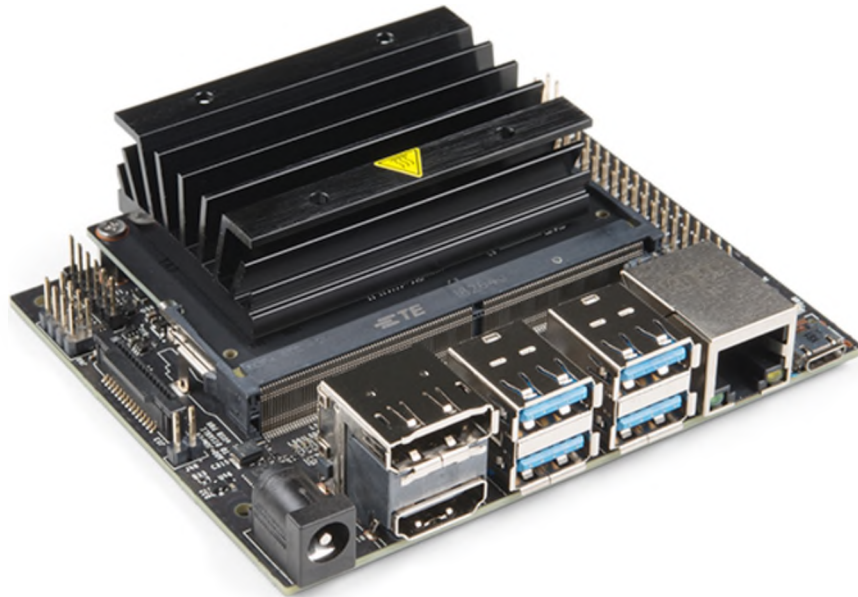


Figure 5: Jetson Nano 4GB Developer Kit [33]

NVIDIA also offers a 2GB version of the Jetson Nano for a slightly lower price of \$59. The double memory capacity you get in the 4GB Nano for only \$40 more makes it a much better option when dealing with the high memory needs of a CNN.

The camera used to collect the dataset was the Raspberry Pi High Quality Camera. The Raspberry Pi High Quality Camera is the newest camera offering from the Raspberry Pi Foundation, with significant improvements over its predecessor, the Camera Module V2. The High Quality Camera boasts an upgraded 12.3 megapixel Sony IMX477 sensor, adjustable back focus, and C/CS mount. It is important to note that the Raspberry Pi High Quality Camera does not have an actual lens, unlike the Camera Module

V2, which must be purchased separately. An Arducam 8mm focal length lens with manual focus and adjustable aperture was used to get clear shots on the robot prototype.



Figure 6: Raspberry Pi High Quality Camera with lens, attached to Raspberry Pi 4.

2.6 Dandelions and Bermudagrass

Weed management in lawns is a common task performed in the Groundskeeping industry. A particularly invasive weed in lawns is dandelion (*Taraxacum officinale*), seen in Figure 7. It has a basal rosette base with a strong taproot. The leaves are deeply serrated and have a lance shape [34]. Managing dandelions in lawns can be a large monetary and time cost. It can be removed manually by severing its aggressive taproot, or sprayed with a herbicide. If the tap root is not removed during mechanical removal, the dandelion will grow back. If only part of the dandelion is sprayed, it can survive and grow back as well. Therefore, it must be sprayed with full coverage or have its tap-root removed to be killed off fully. Using an object detector to identify dandelions with bounding boxes means precision microsprayers could apply full coverage of herbicide application to each one detected. It should be noted that there exists dicot-specific herbicides, meaning they will kill broadleaf weeds like dandelions but will not harm the grass.



Figure 7: Dandelion weed in grass.

Bermudagrass (*Cynodon dactylon*) is a type of turfgrass that is also considered invasive in the United States. Its invasiveness makes it a great grass for playing fields and golf courses because it self-repairs. For example, if a chunk of grass gets removed from a golf swing, bermudagrass will eventually grow laterally to cover it up due to its horizontally creeping stolons. Bermudagrass is known for how well it looks manicured, exhibiting a uniform blanket-like appearance, as seen at golf courses. If dandelions start to take root, it throws off the appearance and no longer is aesthetically pleasing. In addition, dandelions cause unevenness in bermudagrass which can be a tripping issue on sports fields or can cause a golf ball to veer off its intended course. This means managing dandelions in bermudagrass is a worthwhile task for Grounds departments. Having an affordable autonomous robot that could take over this task is the basis for this thesis.

2.7 Related Work

Implementations of plant detection in the field and lab have been a thorough research topic dating back to the 1970s. Some of the earliest work in Machine Vision for plant detection came from [35]. Hooper, Harries, Ambler developed a light sensor that could distinguish plant material from soil based on light absorption ratios. In the 1990s, research on plant recognition happened with

a real-time intelligent weed control system that used Computer Vision and precision chemical application of in-row weeds in commercial tomato fields [36]. In the early 2000s [37] used a hybrid machine learning and neural network approach to classify leaves using Ontology.

2.7.1 Precision Agriculture

Research in precision agriculture using CNNs has been abundant. For example, [38] trained a CNN to detect weeds among sugar beets. They used Near Infrared (NIR) photos to create a segmented soil mask before training their CNN. Where these authors only captured images of plants in early growth stages, [39] used optimized Transfer Learning parameters to retrain ResNet50 based CNN to detect weed and crop species at differing growth stages. Machine Learning and Deep Learning approaches were compared to detect weeds within the canola crop [40]. They used a novel Local Binary Patterns (LPB) approach to show it could get better accuracy and faster inference than VGG16, VGG19, ResNet50, and InceptionV3 CNNs for images taken in the field. A Pattern Recognition algorithm was developed by [41] to classify weeds from crops in lettuce rows, along with building a spray map that can be used for precision application of herbicide to the weeds. This algorithm was implemented into a Computer Vision platform mounted on a tractor that could travel 1.5 MPH. The main drawback of their recognition system is their reliance on spraying the crop early on with a special compound that is easily distinguished from weeds in their algorithm.

2.7.2 Robotic Weeding

There have been numerous studies on Machine Vision in robotic weeding, ranging from path guidance [42] to detection through LIDAR [43]. The authors developed an autonomous robot using LIDAR to detect over 20 different species with 98% accuracy. The point clouds from the LIDAR sensor were used as features in their Machine Learning algorithm, making it robust against illumination and atmospheric conditions. A more recent example, [44] demonstrated a robot that could kill weeds with lasers. It used a color segmentation algorithm to crop plants from the frame, then size estimation to distinguish the weed from the crop. The weed detection coordinates were translated into real-world coordinates and sent to two lasers on gimbals. The robot had limited mobility though, and could only drive forward, stop, take

a picture, processing the data and perform weed elimination, then move forward again. A Siamese Convolutional Neural Network for generic object tracking was trained by [45] and deployed it on a Jetson Nano on a robotic platform, achieving 10 FPS.

A novel bark image dataset was curated and trained with a CNN to classify trees based on their bark with an accuracy of 97.81% with majority voting using Transfer Learning with ResNet34 trained on ImageNet [46]. Another example is from [47], which trained a CNN from scratch to perform Image Classification on 6 classes of fruit growing on trees. They showed real-time detection using a CNN was possible with an inference time of 0.03 ms using a high-performance GPU. Similarly, [48] investigated the feasibility of near real-time Object Detection on an Underwater Autonomous Vehicle (UAV). They trained a Faster R-CNN model and deployed it on an NVIDIA Jetson TX1, GTX 1080, and GTX970 on a mobile platform on the UAV. The made for mobility platform, Jetson TX1, only managed to achieve 0.55 FPS, while the GTX 1080 achieved 5.8 FPS.

2.7.3 Plant Detection with YOLO

There is extensive literature on Object Detection using YOLO, but here I mention a few relevant papers on plant detection using YOLO that helped shape part of this thesis. A YOLO object detector was used by [49], and produced marginal increases in mean Average Precision (mAP) while suffering significant inference speed reduction when increasing image input size from 320×320 to 416×416 and 608×608 . In addition, they were able to get substantial inference speedup, 2.5x, from 16-bit floating point quantization via TensorRT implementation in their model. Using an improved YOLOv3 model based on DenseNet, [50] detected varying growth stages of apples in orchards. While they achieved real-time detection using 512×512 input image size, this was done using an NVIDIA Tesla V100 server. This is a common theme in plant detection studies, where they focus mainly on accuracy and performance in the lab. They rarely test their models in an environment where they would be used in production, such as on an edge device. Another example is [51] whom successfully deployed their YOLOv3 algorithm to a Jetson Nano to be used in a student UAV competition, though failed to report inference results. YOLOv3-tiny has been used to detect goosegrass within strawberry and tomato crops [52]. Their model was pretrained with the COCO dataset and used data augmentation techniques such as color

alteration, cropping, and resizing. Most recently, [53] used a YOLO object detector with a variety of backbones to detect weeds among lettuce crop rows using a Jetson TX2, getting real-time inference of 33 FPS and achieving up to 0.93 mAP. Lastly, [29] demonstrated the superiority of the GPU on an edge device over a CPU by measuring inference times in a YOLO-based model.

2.7.4 Relevant Studies

Finally, I highlight some key studies that my thesis builds upon in the research. The authors in [54] used Computer Vision to detect dandelions and their centers using the Mean Centroid Method. They used classical image processing steps in the detection process such as sharpening, color thresholding, low pass filter, and binarization. The limitations of this study are that it used lab-like conditions to boost their accuracy of centroid detection as well as using hand-picked image processing steps, which is not as robust as Deep Learning. Also using traditional image processing, [55] used edge detection to identify weeds in ornamental grass and sports turfgrass. They identified different filters to convolve the image, resulting in higher accuracy of weed detection. A shortfall of this study is that the authors only saw high accuracy after they first removed dirt, leaves, and non-uniform objects in the images, thus not making their algorithm very robust. The first research to investigate the feasibility of weed detection in turfgrass using CNNs was done by [2]. They trained two types of CNNs, GoogLeNet and VGGNet, for Image Classification of dicot weeds in actively growing bermudagrass. They also trained an object detector using DetectNet to identify a weedy monocot in dormant bermudagrass. Later that year, the same authors did a similar study but investigated detection in perennial grass, instead of bermudagrass, of different types of monocot and dicot weeds [56]. The following year they finally examined broadleaf weed detection in actively growing bermudagrass [57] using Image Classification.

Until now the authors showed the possibility of weed detection in turfgrass using Image Classification and Object Detection. None of their papers have shown detection results on an edge device, having only used powerful Desktop or Server GPUs to measure inference. While they have performed Object Detection on grassy weeds in actively growing bermudagrass, they have not done so with broadleaf weeds in actively growing bermudagrass. My thesis fills in this gap by showing object detection of a broadleaf weed like dandelion is possible in actively growing bermudagrass, as well as eval-

uating the feasibility of real-time detection performance on an edge device.

3 Methodology

To test the feasibility of a neural network’s real-time performance, data must be gathered. For a CNN, the data input are images. This chapter describes the data gathering process, CNN model training and optimization, and finally inference evaluation. As noted in the last chapter, the research into dandelion weed recognition within bermudagrass stops at the lab during inference. This chapter will extend the research by demonstrating deploying an optimized CNN model, trained on a self-gathered dataset, to an edge device. Results gathered from the model training and inferencing will be discussed.

Many studies involving CNN model evaluation leave out key details needed for providing a fair comparison. This is because either the author failed to recognize the importance of such details, or some of these details are abstracted away in the training and inferencing process. To keep a fair comparison while evaluating my models, I kept a couple key configurations constant across each evaluation. First, when comparing separate object detector architectures, I use the same resolutions. I train on the same dataset and measure metrics using the same validation images across all models. All training is done via Transfer Learning using pre-trained weights on the MS COCO dataset. All training is performed using the stochastic gradient descent (SGD) optimizer. I provide inference speed all as one metric, FPS. I run inference for all models on the same edge device, the Jetson Nano. The Jetson Nano was set up to use MAXN power mode for each evaluation, along with `jetson_clocks` being enabled. I use FP16 precision and batch size of 1 while performing inference. The exception to this is running inference using Darknet on the Jetson Nano. It is impossible to use FP16 precision with Darknet as Tensor Cores are required, which is lacking on a Maxwell GPU. Lastly, all FPS results include pre-processing and post-processing operations and were measured on the same video file.

3.1 Dataset

While some research has been done on dandelion detection in turfgrass, either the dataset was private or not conducive to my application as the ground sampling distance was too large. Studies mostly took pictures 3 to 5 feet off the ground, whereas my proposed robot prototype would take pictures at less than 1 foot. Therefore, I decided to curate my own dataset. They would be reflective of dandelions growing in bermudagrass in the Southern California

region.

Photos of dandelions were collected at varying growth stages. Dandelions were captured from as young as its two-leaf seedling stage to full basal rosette flowering stage, which includes a yellow flower. It is important to capture images that are indicative of real-world situations, and I made sure to collect images at almost all plant sizes. Figure 8 demonstrates the distribution of ground-truth label sizes in the dataset.

In checking how big of a dataset to build, [50] showed any number of images exceeding 3000 in the training set did not have a further significant influence on their YOLO model. They used a modified YOLOv3 model to detect varying growth stages of apples in an orchard.

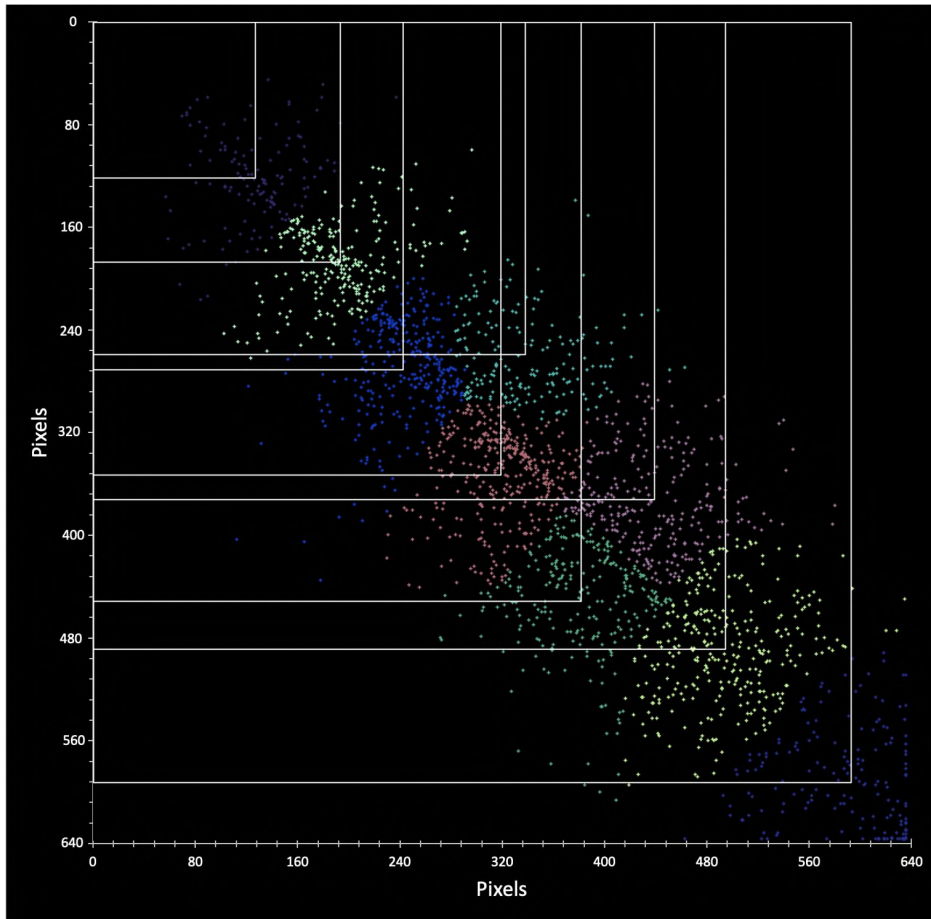


Figure 8: The distribution of ground truth label sizes based on 640x640 image size. Each box represents an ideal anchor box size for training YOLO models.

3.1.1 Data Collection

Initially, I wanted to collect images from California State University Chanel Islands, which has two playing fields on its campus. One of them, Potrero Field, contains kikuyu grass. While this is a turfgrass, it had little to no dandelions present in it as the grass is maintained at higher than average mowing height. By letting the kikuyugrass grow high, it chokes out and denies sunlight to other weeds hoping to take root. The other primary playing field, North Field, contains a multi-species mix of grasses, including rye,

bermudagrass, and fescue. While North Field is typically maintained more often, it was not being maintained most of 2020 due to Covid-19 causing personnel and resource reduction in the Grounds department.

Therefore, I decided to find a playing field close by with bermudagrass that still had proper upkeep. I discovered Pleasant Valley Fields, located 5 miles from the campus of CSUCI, shown in Figure 9. It is a large playing field consisting of 12 FIFA-sized soccer fields with a hybrid bermudagrass that is mowed under 1". Bermudagrass is traditionally mowed at 1" or below at golf courses, recreation fields, and sports fields.



Figure 9: Pleasant Valley Fields - location of data gathering.

A hybrid bermudagrass is the prominent turfgrass used in natural playing fields and golf courses across the United States. Due to the field's size, there was plenty of dandelions to take pictures of as well. This made for a suitable location to collect data and be representative of general sports fields for future studies conducting similar research on Object Detection in bermudagrass.

I did not opt for taking photos by hand while walking the field, as many studies do. Instead, I collected the photos that would replicate how an autonomous robot weeder would collect data. This required building a prototype robot equipped with a computer and the same type of camera that would be used in production. I also framed the photos using this prototype to replicate lighting conditions which can be a major source of error in Deep Learning training when not accounted for. Using in-situ data, and not artificially creating it, will make my trained model more robust and transferable to real-world production.

Photos were taken on 6 days in December 2020, February 2021, and March 2021. In total, 1714 raw images of dandelions and background were collected. These images served as the training, validation, and test sets.

The dataset was balanced with 50% of the photos containing dandelion and 50% containing only bermudagrass as background. It is important to have a balanced dataset as training only on images with dandelion can lead to a high amount of false positives. The photos were taken with a Raspberry Pi High Quality camera attached to a Raspberry Pi 4 8GB computer, as seen in Figure 10.



Figure 10: Raspberry Pi High Quality Camera with Arducam 8mm lens, on top of robot prototype, along with a power bank.

An Arducam C-Mount lens, with 8mm focal length, was attached to the camera. The photos were captured at 640×640 resolution with the PiCameraApp Version 0.2 [58] running on Raspbian 10 Buster. A JPG format was used. A 640 resolution was chosen for two reasons. First, the YOLO object detector requires a resolution divisible by 32 for its input. Second, a 640 resolution seemed the upper bound for possibly achieving real-time inference based on previous studies. I used a WiFi hotspot on my Motorola G6 Android phone and connected both my Apple Macbook Pro and Raspberry Pi to it. This allowed me to use VNC Viewer on my laptop to remotely connect to VNC Server on the Raspberry Pi and share its desktop. I was then able to use PiCameraApp to preview photos for consideration before capturing them in the field.

While all images were captured with the robot stationary, video capture was also investigated. Using the same 640×640 resolution, video was able to capture images of dandelions clearly, without blur, by using a sufficiently high shutter speed. This was done with the Raspberry Pi's "raspivid" library. Shutter speed is also adjustable using the Jetson Nano.



Figure 11: Robot prototype at Pleasant Valley Fields.

The robot prototype's main body consisted of a box with dimensions $229\text{ mm} \times 229\text{ mm} \times 330\text{ mm}$. The Raspberry Pi High Quality camera, attached to the Raspberry Pi, was mounted on top of the box with a bird's eye view of the grass. The grass to camera distance was approximately 280 mm. The measured ground sampling distance was 2.79 pixels/mm. The box body had 2 sets of wheels mounted on the bottom of it for mobility. An LED ring light was mounted on the inside top of the box to provide a uniform lighting scheme of the turfgrass. Utilizing artificial light, and blocking out natural light using the box body, is important in collecting a consistently illuminated dataset. Using the LED light also means future inference will not be subject to the variability of sun and cloud conditions. It can also be used during the night as it is not dependent on sunlight. The camera lens was placed through the hole of the LED ring light to provide parallel lighting to the lens, as seen in Figure 12. An Anker 20,000 mAh 18W power bank was used to power the Raspberry Pi and LED ring light.



Figure 12: How the camera lens is positioned within the LED ring light.

The Raspberry Pi was used instead of the Jetson Nano for data collection. I originally had planned to test inference on the Raspberry Pi as well. Time constraints and further reading into the literature showed the Raspberry Pi would be an inferior edge device for Object Detection over the Jetson Nano. Fortunately, both the Raspberry Pi and Jetson Nano run on a Debian-based distribution of Linux. I can replicate the exact photo collection process on the Nano as I did on the Pi, including using the same CSI-based camera and lens.

3.1.2 Data Preparedness and Augmentation

In Object Detection, training data must have ground truth labels. These ground truth labels are bounding box coordinates denoting the location of objects in an image. I utilized the software web application Roboflow [59] to annotate the ground truth labels manually.

A common occurrence in training Deep Learning models is overfitting. Overfitting is when a model starts to memorize the patterns in the training data too well. When a model is overfitting is introduced to new data, it will have subpar accuracy compared to the data it originally trained on. One way to help with overfitting is through data augmentation. Data augmentation is a way to increase the amount of data the network is trained on by performing specific techniques that generalizes the data [15]. For a CNN that uses images, data augmentation means the images are altered to be beneficial to model training.

Another option to help reduce overfitting and help with model accuracy is performing a pre-processing step to each image. My goal is to achieve real-time performance in Object Detection which is dependent on the Computer Vision pipeline throughput. I opted not to include a pre-processing step as it would add increased time in the pipeline. Also, the attractability of using a Deep Learning model is that it learns features on its own, and it cuts out the need to hand chose them in a pre-processing step.

In Roboflow, I performed four data augmentations to my data. These included flipping horizontally and vertically, rotating 90 degrees clockwise and counter-clockwise, rotating between -45 degrees to +45 degrees, and blurring at 0.25 pixels. On some datasets, augmentation can be harmful to accuracy and getting the training to converge. For example, you would not want to use the horizontal flip data augmentation technique to a self-driving car dataset, as the model would never come across cars driving upside down in real life. The weeding robot would view the dandelion in many different orientations though. The dandelions it approaches can be in any type of rotation and positioned anywhere in the frame. Thus, I chose the data augmentations to make my dataset more robust. There was no need to perform any type of illumination-based data augmentations as my dataset was uniformly lit by the LED light.

After augmentation, the dataset was expanded to 4285 images. The dataset was now more robust and less likely to overtrain on irrelevant features. The training set consisted of 3858 images. The validation set consisted

of 342 images, and the test set had 85. Only the training set had augmented images included in it.

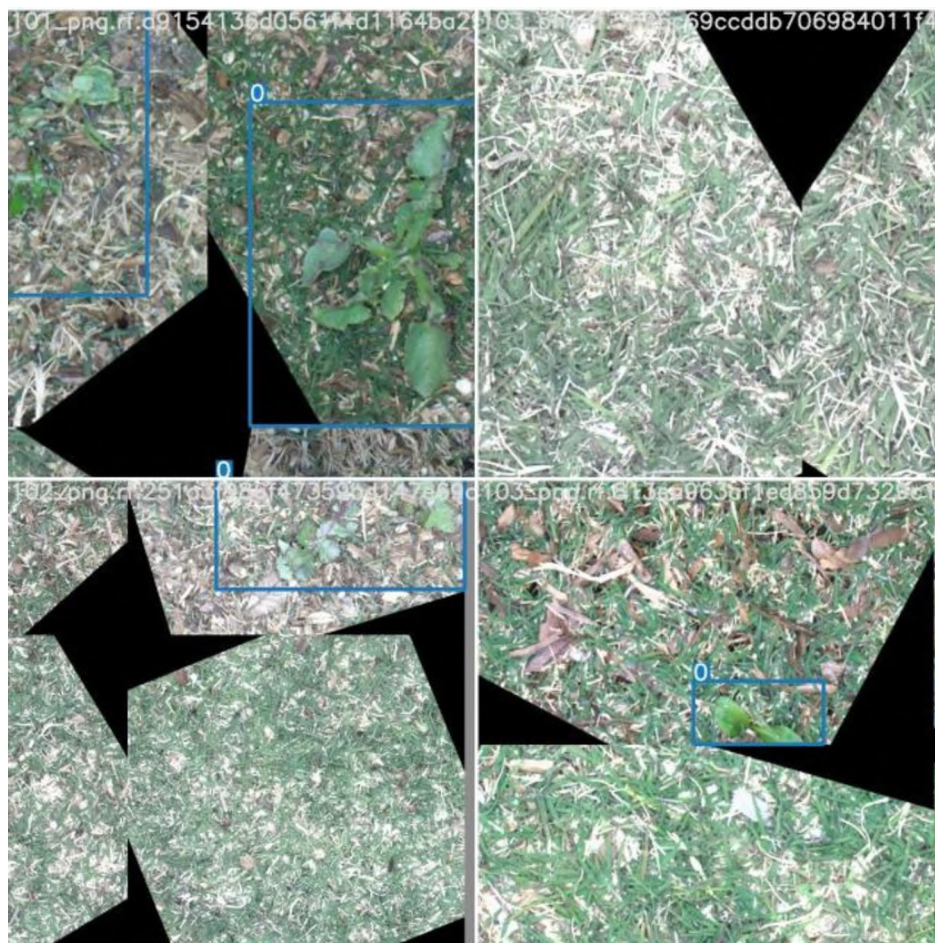


Figure 13: Example of augmentations done on the dataset.

3.2 Training

A variety of software frameworks were used for training. They were Darknet, an open-source framework written in C that takes advantage of CUDA in training and inference. I also used PyTorch, a library in Python that is easy to use and also integrates with CUDA. Lastly, I used NVIDIA's Transfer Learning Toolkit version 3.0. This is NVIDIA's most recent offering in building CNN models that are ready for production.

Varying training hyperparameters such as learning rate and epochs were examined in preliminary work (data not shown). Eventually, I converged on a set of hyperparameters that were maximizing accuracy, that are discussed at the beginning of this chapter. When training an object detector, a specific batch size is chosen. Images are randomly selected in each batch which makes the gradient descent problem a random process as well. To filter out noise in this random process, it is better to pick larger batch sizes as they better represent the dataset as a whole. These bigger batch sizes enable you to increase the learning rate as progress will be larger along the gradient [27]. Therefore, I chose the largest batch size for each training that the video card would allow in memory. There was no single batch sized used across trainings as training on different resolutions meant having to use different batch sizes.

3.2.1 Computation Environment

The environment where training of Deep Learning models generally happens locally on a workstation, in the cloud, or a containerized environment such as Docker. The cloud environment has the least amount of time setting up but can result in high costs long term. A containerized environment is beneficial in setup time and proven compatibility of its software stack. I ended up choosing a local environment for training to maximize my understanding of the entire training process and was already in possession of a CUDA-capable graphics card.

The Operating System (OS) used was Ubuntu Desktop 20.04.1 LTS Focal Fossa. All training was performed on an NVIDIA RTX 2070 Super. CUDA version 10.2, cuDNN version 8.0.5, and NVIDIA 455.45.01 drivers were installed as well.

The computation environment that performed inferencing was on the Jetson Nano. The Nano comes with Ubuntu 18.04.5 LTS along with NVIDIA Jetpack 4.5.1 SDK. This includes CUDA version 10.2, cuDNN version 8.0.0, TensorRT version 7.1.3, and NVIDIA L4T version 32.5 graphics drivers.

3.2.2 YOLOv4 Darknet

Darknet is the standard training platform for YOLO and was written originally to train YOLO object detectors. YOLOv4 was chosen over YOLOv3 as it provides many new features that provide an increase in accuracy and speed on the same MS COCO dataset. One such feature used was mosaic,

a data augmentation operation introduced in YOLOv4. It mixes 4 training images into one to train the classifier on objects outside their normal context [32].

Within YOLOv4 are many derivative models that consist of changes in architecture. One such derivative is Yolov4-tiny, released in June 2020 [60]. YOLOv4-tiny is a model with a smaller backbone and only 2 detection heads instead of the 3 found in YOLOv4. A detection head is where a specific sized feature map is used to make bounding box predictions. This reduction in the number of layers in Yolov4-tiny means there are less parameters and weights, and thus less computation time is needed to make a forward pass through it. I used the official "YOLOv4 pre-release" version that features an SPP/PAN neck, the YOLOv3 head architecture, and a CSP-based Darknet backbone. A summary of YOLOv4 models are seen in Table 2.

| Model | Number of Layers | Number of Heads |
|----------------|------------------|-----------------|
| YOLOv4 | 162 | 3 |
| YOLOv4-tiny | 38 | 2 |
| YOLOv4-tiny-3l | 45 | 3 |

Table 2: Number of layers and detection heads in common YOLOv4 models

After preliminary model training, I ended up choosing a modified version of YOLOv4-tiny called 'Yolov4-tiny-3l', which has 3 YOLO detection heads instead of 2. This means the model predicts bounding boxes at three different scales instead of 2. It was shown by [29] that the extra YOLO detection head led to only a minor reduction in speed when deployed on a Jetson Nano. Having the 3rd YOLO head means higher accuracy due to a 3rd feature map used in detection.

Input resolutions of 320, 416, and 640 were chosen to compare accuracy, recall, and inference time. New anchor boxes were generated for each resolution, specific to my dataset. New anchor boxes are not generated automatically and must be done with Darknet's "calc_anchors" function. It uses k-means clustering to generate new anchors based on input dimensions, the number of detectors in the YOLO head, and existing ground truth labels in the training dataset.

I also trained a full YOLOv4 model called 'Yolov4'. I wanted to compare the standard model to the tiny models. I knew inference performance would

suffer in the standard model, so I only trained this model at the 320 input resolution.

Each model training took varying amounts of time depending on the input resolution. The models were trained for 6000 iterations using the recommended formula from the Darknet documentation. All trainings were verified they converged, meaning they reached a minima in the loss function. After model training, precision, recall, and model size were recorded.

3.2.3 YOLOv5 Pytorch

YOLOv5 training was performed in PyTorch version 1.7.1, along with torchvision 0.8.2. I used the official YOLOv5 “v4.0” repository, which is the latest YOLOv5 implementation in PyTorch made training easy to set up and perform over Darknet.

YOLOv5 has four different models of varying sizes. I used the ‘Yolov5s’ model, which is the smallest one they offer. Like ‘Yolov4-tiny-3l’, this small model also features 3 YOLO heads and has a reduced layer backbone over the larger models. To have a fair comparison to YOLOv4, I also trained models on 320, 416, and 640 input resolutions for 100 epochs each.

YOLOv5 offers a superior way to measure metrics and track hyperparameters compared to the other training platforms tested. It is integrated with the web application Weights and Biases [61]. Each training run has all related training measures consolidated, making comparisons easy, as seen in Figure 14. By comparing differences in runs visually, overall time in finding an optimal configuration is highly reduced. Like the YOLOv4 process, I recorded the precision, recall, and model size of each trained ‘Yolov5s’ model.

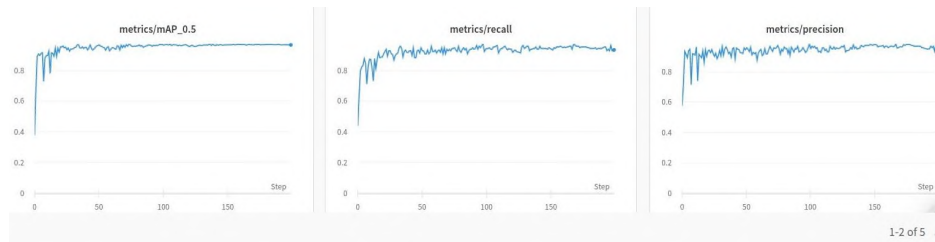


Figure 14: An example of a training run measures in Weights and Biases.

3.2.4 TLT 3.0

Transfer Learning Toolkit (TLT) is NVIDIA's Transfer Learning framework for training a variety of object detection models. I used the latest version, 3.0. The main benefit of using version 3.0 is the support of YOLOv4. One benefit of training YOLOv4 in version 3.0 is that there is no need to resize the training images up front. It is also beneficial because its support of Jupyter notebook examples that make it easier to train a model compared to only using its Command Line Interface (CLI).

TLT uses a TensorFlow with Keras backend for training. It only allows Transfer Learning by using NVIDIA's pretrained models in a Hierarchical Data Format 5 (.hdf5) extension. TLT has extensive documentation and makes training more streamlined and easier to use. During training, each epoch is saved in a weights file which allowed choosing the weights that gave the best performance metrics.

Unlike Darknet-based YOLOv4 and PyTorch-based YOLOv5, TLT only provides one base model for YOLOv4, but with the ability to use a variety of backbones such as Darknet, Resnet, VGG, MobileNet, and SqueezeNet. To provide a fair comparison, I used the Darknet backbone. To make it a "tiny" model and be comparable to the CSP-based backbones of 'Yolov4-tiny-3l' and 'Yolov5s', I specifically selected the 'CSPDarknet19' backbone for feature extraction. In addition, the YOLOv4 version on TLT uses 3 detection heads, making it a fair comparison to 'Yolov4-tiny-3l' and 'Yolov5s'. Finally, I trained these models using 320, 416, and 640 input resolutions for 100 epochs.

In addition to the YOLO models, I trained SSD-based models using MobileNet as the backbone, called 'SSDMobileNetV2'. TLT did not support using the CSPDarknet19 backbone with the SSD detector. MobileNet is a practical choice in edge inference as it employs depthwise-separable and group convolutions which increase performance. MobileNet's reduced architecture, paired with the SSD detector, makes it a suitable option when choosing a lightweight CNN. The SSD models were trained with an input resolution of 320, 416, and 640 for 100 epochs each.

Finally, I trained two Faster-RCNN models with TLT, called 'FRCN-NDarknet19'. Faster-RCNN is a supported model in TLT that has state-of-the-art accuracy in Object Detection. I paired it with the CSPDarknet19 backbone for a more fair comparison with the YOLO models. I trained the models using 320 and 416 input resolutions for 100 epochs each. I left out

640 resolution after it became apparent how poor detection speed was in the 416 resolution model.

One benefit of TLT is that it offers a pruning function that trims away weights not providing any benefit to feature extraction. This removal of weights results can provide even faster inference times for ‘Yolov4CSPDarknet19’, ‘SSDMobileNetV2’, and ‘FRCNNDarknet19’. I did not pursue pruning of these models as it would be unfair if not done to all the other models being evaluated.

3.3 Inference

Detection speed was tested on 4 different inferencing platforms. These were Darknet, PyTorch, tkDNN, and Deepstream. For the ‘Yolov4-tiny-3l’ and ‘Yolov4’ models, they were tested in Darknet, tkDNN, and Deepstream. ‘Yolov5s’ was tested within PyTorch and Deepstream. Finally, the TLT models, ‘Yolov4cspdarknet19’, ‘SSDMobileNetV2’, and ‘FRCNNDarknet19’ were all tested only in Deepstream. While a more fair comparison would allow all models to be tested across all inferencing platforms, major incompatibilities in file formats, training platforms, and software versions did not allow it.

Inference with Darknet was straightforward. Darknet inference functionality directly supports Darknet weights and its parameter file. It is run with its “map” function call. I used a 640x640 video taken with the robot prototype of bermudagrass and dandelions to measure FPS for each of the models. One important note about using Darknet on the Jetson Nano is that it does not support its “CUDNN_HALF” flag being enabled when building it, thus not performing inference at FP16 precision. As mentioned earlier, the Jetson Nano has a Maxwell GPU, which lacks Tensor Cores, which are needed to run FP16 precision. Therefore, all inference run in Darknet on the Jetson Nano was run at FP32 precision.

Inference with PyTorch was similarly a direct approach. It used the native weight and parameter files to calculate FPS on the same video used with Darknet.

Deepstream is NVIDIA’s production-ready inference platform which required much more nuance to set up. At this time, the latest version of Deepstream, version 5.1, was used. Depending on the model architecture, different bounding box parsing libraries had to be used. Another difference is that, unlike Darknet and PyTorch, which used weight files directly, Deep-

stream required its model to be converted to a TensorRT engine. This can be done within TLT or natively by Deepstream. Deepstream natively supported weights trained in Darknet, but weights from YOLOv5 in PyTorch first had to be converted to ONNX format. After, TensorRT's function "trtexec" had to be used to convert the ONNX based model into a TensorRT engine. The same video file was used to measure FPS that was used in Darknet and PyTorch.

The final inferencing platform used was a library called tkDNN. While Darknet does not allow FP16 precision on the GPU, tkDNN solves this problem with a workaround specifically for Jetson-based products. Set up is the most complicated on this platform, requiring Darknet weights to be exported and converted to a compatible format. Next, tkDNN must be built with specific configuration parameters set.

4 Results and Analysis

In the previous chapter, I presented the implementation of the data gathering, processing, training, and evaluation process. This section, I present and analyze the results from training and inference evaluation. I first go over inference speed results and then analyze accuracy vs. speed.

To reiterate, the data used in this thesis were novel images taken in the field of dandelions in bermudagrass at a local sportsfield park. The data was consistent in that all images were captured at the same 640×640 resolution, same ground sampling distance, with the same Raspberry Pi High Quality Camera, and under the same lighting conditions. I also made sure to get an almost even distribution of dandelion sizes, as seen from Figure 8, with smaller sizes lacking. An example of some images from the dataset with dandelion and background can be seen in Figure 15. The top row shows clear examples of dandelions while the bottom row shows what a background looks like.

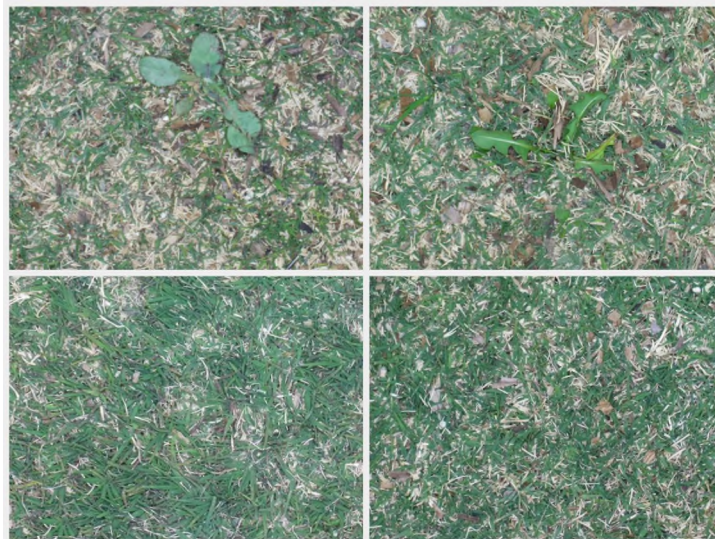


Figure 15: Top row featuring dandelions and bottom row featuring background images.

After training each CNN on the custom dataset, their precision and recall were measured. The precision metric, mAP, was recorded where it was highest throughout many epoch checkpoints. The models were then transferred

to the Jetson Nano, where they were optimized to run inference. Inference speed was then measured in FPS. These results can be seen in Table 3. As mentioned in Chapter 3, three different training platforms were used and four different inferencing libraries. Finally, model size was recorded as well.

| Model | Resolution | Training Platform (x86) | Inference Platform (aarm64) | Model Size (MB) | mAP@.5 | Recall | FPS |
|--------------------|------------|-------------------------|-----------------------------|-----------------|--------|--------|------|
| Yolov4 | 320 | Darknet | Darknet | 210.2 | 0.97 | 0.94 | 2.7 |
| Yolov4 | 320 | Darknet | tkDNN | 118.8 | 0.97 | 0.94 | 6.9 |
| Yolov4 | 320 | Darknet | Deepstream | 119 | 0.97 | 0.94 | 8.9 |
| Yolov4-tiny-3l | 320 | Darknet | Darknet | 24.5 | 0.86 | 0.84 | 15.6 |
| Yolov4-tiny-3l | 320 | Darknet | tkDNN | 14.6 | 0.86 | 0.84 | 39.8 |
| Yolov4-tiny-3l | 320 | Darknet | Deepstream | 14.6 | 0.86 | 0.84 | 49.6 |
| Yolov4-tiny-3l | 416 | Darknet | Darknet | 24.5 | 0.87 | 0.85 | 10.9 |
| Yolov4-tiny-3l | 416 | Darknet | tkDNN | 33.7 | 0.87 | 0.85 | 26.1 |
| Yolov4-tiny-3l | 416 | Darknet | Deepstream | 33.7 | 0.87 | 0.85 | 33.1 |
| Yolov4-tiny-3l | 640 | Darknet | Darknet | 24.5 | 0.77 | 0.65 | 5.3 |
| Yolov4-tiny-3l | 640 | Darknet | tkDNN | 34.3 | 0.77 | 0.65 | 12.2 |
| Yolov4-tiny-3l | 640 | Darknet | Deepstream | 34.3 | 0.77 | 0.65 | 16.1 |
| Yolov4CSPDarknet19 | 320 | TLT3.0 | Deepstream | 101.9 | 0.88 | n/a | 38.9 |
| Yolov4CSPDarknet19 | 416 | TLT3.0 | Deepstream | 190.7 | 0.907 | n/a | 27.8 |
| Yolov4CSPDarknet19 | 640 | TLT3.0 | Deepstream | 193 | 0.88 | n/a | 12.5 |
| Yolov5s | 320 | Pytorch | Deepstream | 16 | 0.97 | 0.91 | 41.2 |
| Yolov5s | 320 | Pytorch | Pytorch | 14.3 | 0.97 | 0.91 | 10.8 |
| Yolov5s | 416 | Pytorch | Deepstream | 23.6 | 0.97 | 0.94 | 26.3 |
| Yolov5s | 416 | Pytorch | Pytorch | 14.4 | 0.97 | 0.94 | 9.6 |
| Yolov5s | 640 | Pytorch | Deepstream | 20 | 0.96 | 0.94 | 12.5 |
| Yolov5s | 640 | Pytorch | Pytorch | 14.4 | 0.96 | 0.94 | 5.6 |
| SSDMobileNetV2 | 320 | TLT3.0 | Deepstream | 5.2 | 0.91 | n/a | 41.8 |
| SSDMobileNetV2 | 416 | TLT3.0 | Deepstream | 6.6 | 0.90 | n/a | 27.5 |
| SSDMobileNetV2 | 640 | TLT3.0 | Deepstream | 9.6 | 0.76 | n/a | 12.6 |
| FRCNNDarknet19 | 320 | TLT3.0 | Deepstream | 49.2 | 0.85 | 0.91 | 0.7 |
| FRCNNDarknet19 | 416 | TLT3.0 | Deepstream | 70.4 | 0.97 | 0.92 | 0.7 |

Table 3: CNN Metrics - Precision, Recall, and FPS

Table 3 shows six different CNN models at varying resolutions. Two of them were trained in Darknet: ‘Yolov4’ and ‘Yolov4-tiny-3l’. ‘Yolov4’ is the standard YOLOv4 model and ‘Yolov4-tiny-3l’ is the standard Yolov4-tiny model but with an extra YOLO head. Of these two different models, they were inferenced on three different platforms: Darknet, tkDNN, and Deepstream. It is of note that Darknet has three different connotations. Darknet is a training platform, an inferencing platform, and a backbone CNN used in Image Classification and Object Detection. The model ‘Yolov4’ was only trained at 320 resolution as preliminary studies showed higher resolutions were not feasible in attaining real-time performance. This was proven as any of these three models achieved no more than 9 FPS. ‘Yolov4-tiny-3l’ was

trained at three different resolutions: 320, 416, and 640. These resolutions were picked based on prevalence of similar resolutions in the literature using object detectors. ‘Yolov4-tiny-3l’ showed promising results on tkDNN and Deepstream at 320 and 416 resolutions, either getting close to or exceeding real-time inference performing of 30 FPS. Inference on Darknet showed the most inferior performance but this is due to a lack of FP16 precision, as mentioned in the previous chapter.

The third model in the table, ‘Yolov4CSPDarknet19’, is a YOLOv4-tiny derivation from NVIDIA that also features 3 YOLO heads like ‘Yolov4-tiny-3l’. Due to compatibility limitations, it was only trained in TLT and inferred in Deepstream. Similar to ‘Yolov4-tiny-3l’, it showed pleasing results at 320 and 416 resolutions, achieving 38.9 and 27.8 FPS, respectively.

The fourth model is ‘Yolov5s’, the smallest model offering from YOLOv5. It was only trained in PyTorch and inferred in two platforms: PyTorch and Deepstream. Analysis of FPS shows the most promising speed comes from 320 and 416 resolutions using Deepstream. Inference in PyTorch exhibited surprisingly slow speed, even at 320 resolution, only managing 10.8 FPS.

The fifth and sixth models, ‘SSDMobileNetV2’ and ‘FRCNNDarknet19’, were also trained only in TLT and inferred in Deepstream due to compatibility limitations with other platforms. Similar to the other YOLOv4-tiny variants, ‘SSDMobileNetV2’ showed compelling FPS results at 320 and 416 resolutions in Deepstream, managing 41.8 and 27.5 FPS, respectively. ‘FRCNNDarknet19’ managed dismal speed results, with both 320 and 416 resolutions achieving 0.7 FPS. I decided not to train a 640 FRCNN model as previous research showed it would be slow, but it was illuminating how poor speed was. FRCNN is not a viable object detector for an edge device like the Jetson Nano.

Next, I map out precision vs. inference speed of the CNNs to determine which is most optimal, in Figure 16. It is a common phenomena in Object Detection studies where accuracy and speed exhibit Pareto efficiency [15], [17], [32]. This means model accuracy cannot be improved without making speed worse off, and vice versa. Mapping out metrics in this way will show which CNN is most feasible for real-time inference while also exhibiting reasonable accuracy. It will also demonstrate if these trained models will exhibit Pareto efficiency.

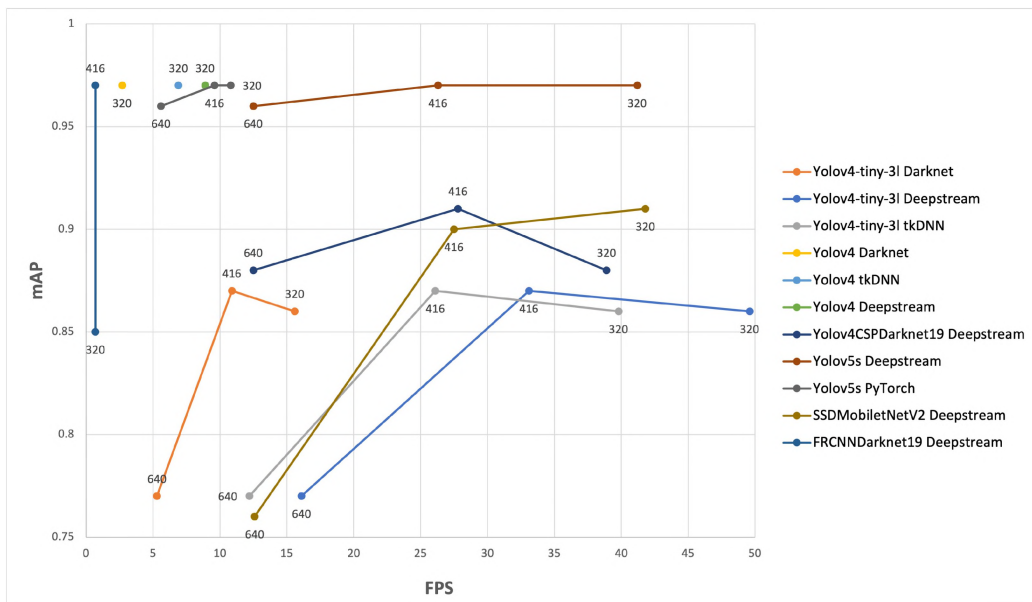


Figure 16: Comparison of accuracy (mAP) vs speed (FPS) of trained CNNs. Each series represents a different model inferenced on a specific library, across varying resolutions.

With mAP on the y-axis and FPS on the x-axis, the ideal model with the highest accuracy and fastest inference would exist in the top right of the graph. First, six of the models achieved real-time inference speeds, surpassing the 30 FPS threshold. Two models have the most promising results when accounting for accuracy: ‘Yolov5s’ and ‘Yolov4-tiny-3l’, both at 320 resolution and inferenced on Deepstream. The trade-off between the two means the ‘Yolov5s’ model has higher accuracy, while choosing ‘Yolov4-tiny-3l’ means faster speed. It is important to point out that the y-axis (mAP) scale has a minimum bound of 0.75 and maximum bound of 1, while the x-axis (FPS) has a minimum bound of 0 and a maximum bound of 50. This means the magnitude of change visualized in mAP is not the same as FPS. Both have high accuracy, 0.86 mAP for ‘Yolov4-tiny-3l’ and 0.97 mAP for ‘Yolov5s’. ‘Yolov4-tiny-3l’ has a 20.4% increase in speed and a 12.8% reduction in accuracy compared to ‘Yolov5s’.

The best inference platform was Deepstream compared to Darknet, PyTorch, and tkDNN. For each model that was inferenced on tkDNN, there existed a faster occurrence at the same resolution, on Deepstream. This

demonstrates tkDNN was inferior to Deepstream. Their library claims to be written specifically to speed up Deep Learning inference on Jetson products, yet it fails to achieve what NVIDIA’s Deepstream implementation does. This does not mean it exhibited poor performance as it still achieved real-time inference with the ‘Yolov4-tiny-3l’ model at 320 resolution. PyTorch and Darknet both failed to achieve real-time performance.

An interesting observation from Figure 16 is that peak accuracy for most models happened at 416 resolution and dipped for 320 and 640 resolutions. This is in contrast to the phenomena of increasing accuracy with increasing resolution in other Object Detection studies. There are a couple reasons I attributed to the trends seen in Figure 16. First, the dataset trained on is much simpler compared to the ones used in [15], [17], [32]. They trained on MS COCO, which has 80 classes and complex objects, whereas the dataset I curated has one class with a simple object shape and orientation. Second, I believe the lack of small objects in my dataset are causing a decrease in mAP at 640 resolution. You usually want to increase the resolution to help detect smaller objects but most of my ground truth labels skewed towards medium to large size objects, relative to frame size. Finally, I attribute the peak in accuracy at 416 resolution to the anchor box choices. Anchor boxes have to be selected based on data distribution and input image size, and I attribute the lack of wise anchor box choices to decreases in mAP.

Overall, the feasibility of real-time dandelion detection with a CNN is best realized using ‘Yolov5s’ or ‘Yolov4-tiny-3l’ while using Deepstream for inference. Figure 17 shows example detection results of the ‘Yolov5s’ model with 320 input resolution. It is a combination of sixteen images, with eight containing dandelion and eight containing background images. In the background images, no false positives were observed. In the images with dandelion, they were all localized correctly.

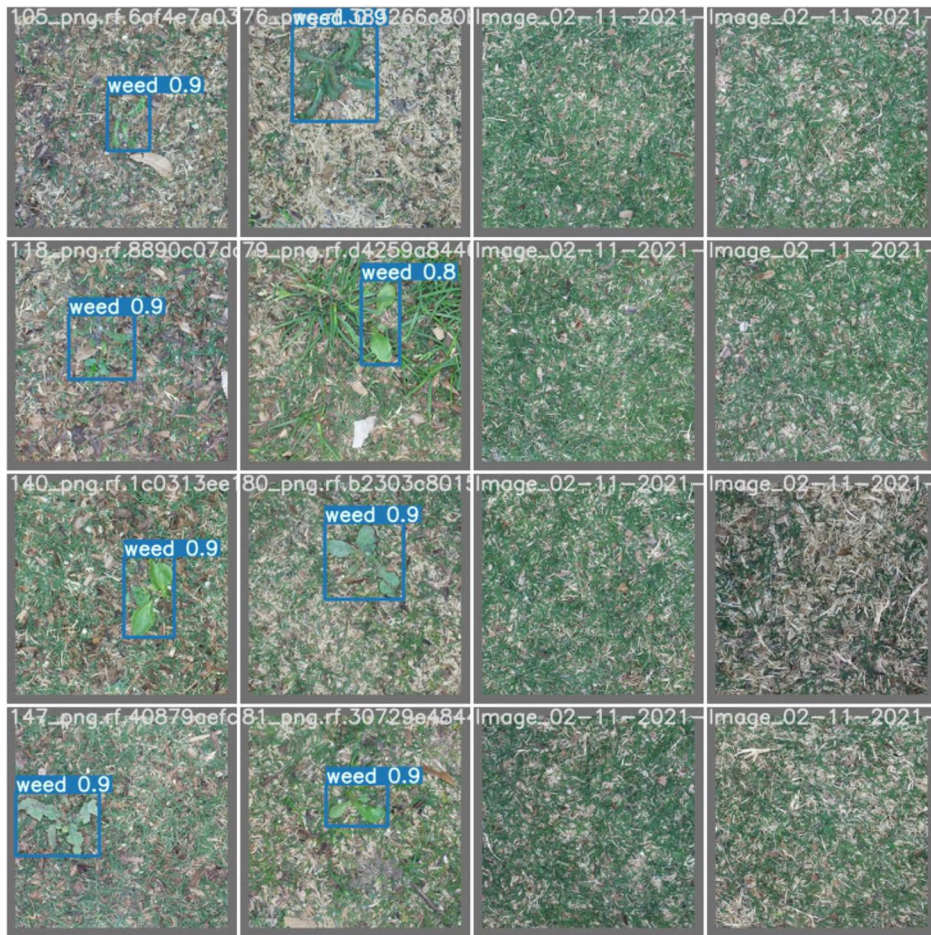


Figure 17: Predictions made by 'Yolov5s' at 320 resolution on 8 images containing dandelions and 8 background images.

5 Discussion

Weeds such as dandelions are considered a serious issue in turfgrass management. They compete for nutrients, water, sunlight, and cause undesirable aesthetics. Traditional methods of weed management in turfgrass have many issues, such as increased costs and potential environmental and health hazards. Deep Learning and other smart solutions are abundant in Agriculture production and research but lacks in the Groundskeeping industry such as athletic fields, golf courses, commercial and residential lawns, and institutional landscapes. The aim of this thesis was to test the feasibility of real-time weed detection in turfgrass using an edge device. This would provide the Computer Vision module to an affordable autonomous robot utilizing precision sprayers.

The first published study on weed detection in turfgrass systems using Deep Learning was by [2]. They showed the feasibility of using CNNs to identify broadleaf weeds in actively growing bermudagrass with Image Classification and weedy Bluegrass in dormant bermudagrass using Object Detection. The same authors followed up that study by demonstrating the feasibility of detecting weedy grasses in actively growing bermudagrass using Object Detection [57]. In addition, Object Detection of broadleaf weeds was performed in [56], including dandelion, in perennial ryegrass. Ryegrass has a different texture and growth habit than bermudagrass. My thesis work sits within these studies as none had shown the feasibility of dandelion detection in actively growing bermudagrass using Object Detection. Further, my thesis extends all their work by testing the feasibility of using an object detector in the field on an edge device, in the hopes of achieving real-time performance.

Work in this thesis involved gathering images in the field. This data was cleaned, labeled, and then augmented for training. Overall, 15 CNN object detectors were trained and evaluated for accuracy. Finally, they were all moved to an edge device, optimized, and performed inference.

This thesis's key finding is that it is feasible to achieve real-time detection of dandelions in bermudagrass using an edge device such as the Jetson Nano. I used a fair evaluation method of CNNs by keeping most parameters and variables constant across comparisons. Four object detectors were able to achieve real-time performance: 'Yolov4-tiny-3l', 'Yolov5s', 'Yolov4CSPDarknet19', and 'SSDMobileNetV2'. Of these, only 'Yolov4-tiny-3l' achieved this with 416 resolution, while the rest were at 320. The max FPS achieved with 640 resolution was 16.1 FPS by 'Yolov4-tiny-3l'.

I attribute attaining real-time detection from these CNNs to a couple of factors. First, they all have lightweight architecture. Selecting a CNN with reduced layers and kernels is essential when inference speed is most important. Second, quantization is equally important by reducing their precision from FP32 to FP16. Third, choosing an edge device with a GPU made achieving real-time inference possible. This is because the training and inference platforms used are optimized in CUDA, and selecting an edge device without a CUDA-enabled GPU means missing out on extra performance. Lastly, the choice of inference platform showed differences in speed. NVIDIA’s Deepstream demonstrated the highest inference speed. This makes sense as NVIDIA also makes the Jetson, CUDA, and TensorRT.

Of the 4 CNNs attaining real-time inference, it was clear from Figure 16 that ‘Yolov5s’ at 320 input resolution stood out as the optimal choice. It achieved 97% precision, 91% recall and 41.2 FPS using Deepstream. I believe it would be very suitable serving as an object detector on an autonomous weeding robot. Its high precision means it will have a high success rate of detection when it makes a prediction and its high recall means it will have a high detection rate of target weeds. As mentioned in Chapter 3, recall is not as important as precision in dandelion detection and spraying. Therefore, even though ‘Yolov4-tiny-3l’ has lower recall, it would also be a suitable CNN to implement in a weeding robot as well.

What cannot be determined by this thesis is how lightweight and quantized CNNs can perform on edge devices that are not CUDA-capable. The TensorRT library and Deepstream inference platform are what made the most contribution to speeding up inference in the models evaluated in this thesis. One should not infer how these models would perform on edge devices like the Raspberry Pi or Google Coral, both which lack CUDA cores.

5.1 Limitations

Bermudagrass is either actively growing or dormant. In Southern California, it can be actively growing year-round, while in other years and locations it can enter a dormant state if temperatures dip below freezing. One shortcoming of this thesis is that no training images contained dormant bermudagrass. I hypothesize the trained models would be more robust against color changes in bermudagrass if dormant bermudagrass images were used in training.

Another issue faced was that my dataset seemed to lack complexity. It could have used more images of grass that had been scalped by a mower, of

sprinkler heads and irrigation boxes, high traffic areas where grass is compacted, and when grass has morning dew on it. All these would have improved the complexity and robustness of the dataset and could have helped with accuracy issues when training at different input resolutions.

Another limitation of this thesis was the capture method of data. While I used the same type of camera that would be used in production, the frames were captured while the robot was stationary. In production, the robot would be moving. The lack of motors on the robot prototype meant I could not capture video or frames in motion.

5.2 Future Work

The inference time measured in this thesis is end-to-end, which includes pre-processing, passing through the network, and NMS with bounding box prediction. If this work is to be implemented into a robot, additional processing time will be required to translate bounding box detection into an algorithm that handles weed removal, such as precision spraying or mechanical removal. How fast this algorithm is will determine the final Computer Vision pipeline of the robot. Therefore, it is advised to pick and train a model that is achieving more than 30FPS to account for this extra post-processing speed.

One shortcoming of the Jetson Nano is that it does not support INT8 operations. Testing inference on a device that supports it like the Google Coral or Jetson NX could provide higher FPS. Further, exploring other edge devices is important as newer technologies are developed that help improve throughput.

Two other routes that can be explored to help speed up throughput of the detection system. First, model pruning can be performed, which eliminates some weights and thus shrinking model size. Second, throughput can potentially be improved by using OpenMP parallelization on the CPU of an edge device to decrease latency in pre-processing and the spraying algorithm which takes the bounding box coordinates from inference.

Finally, this thesis provides the Computer Vision framework for creating an autonomous weeding robot. It shows real-time dandelion detection using an affordable Jetson Nano is possible. Next steps would be to build the robot and implement an algorithm that uses the object detectors trained with this thesis to spray dandelions using precision microsprayers.

5.3 Conclusions

Computer Vision plays an important role in the perception suite in precision weed management. To avoid misclassifications and misses all together, the model used must be robust and accurate. Robustness and high accuracy can be achieved through the use of Deep Learning. However, Deep Learning is very computationally intensive and can be slow on edge devices. Fortunately, it benefits from the parallel processing and matrix computation of a GPU. This improvement, along with quantization, lightweight CNN architecture, and platform optimization, means tasks such as Object Detection is possible on resource-constrained edge devices, thus taking Deep Learning to the edge. These edge devices can then be used outside the lab on robotic platforms. In the Groundskeeping industry, utilizing an autonomous weeding robot can save on resource costs, divert manpower to more complex projects, and eliminate human error in the weeding process. This AI-powered software and hardware for this new intelligent weeding approach in turfgrass is now realized with this thesis.

References

- [1] Sparsh Mittal. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *Journal of Systems Architecture*, 97:428–442, August 2019.
- [2] Jialin Yu, Shaun M. Sharpe, Arnold W. Schumann, and Nathan S. Boyd. Deep learning for image-based weed detection in turfgrass. *European Journal of Agronomy*, 104:78–84, March 2019.
- [3] T. C. Wei, U. U. Sheikh, and A. A. A. Rahman. Improved optical character recognition with deep neural network. In *Proceedings of the 2018 IEEE 14th International Colloquium on Signal Processing Its Applications (CSPA)*, pages 245–249, March 2018.
- [4] J. Vreća, K. J. X. Sturm, E. Gunzl, F. Merchant, P. Bientinesi, R. Leupers, and Z. Brezočnik. Accelerating Deep Learning Inference in Constrained Embedded Devices Using Hardware Loops and a Dot Product Unit. *IEEE Access*, 8:165913–165926, 2020.
- [5] K. A. Althelaya, S. A. Mohammed, and E.-S. M. El-Alfy. Combining Deep Learning and Multiresolution Analysis for Stock Market Forecasting. *IEEE Access*, 9:13099–13111, 2021.
- [6] D. Batra, G. Singhal, and S. Chaudhury. Gabor filter based fingerprint classification using support vector machines. In *Proceedings of the IEEE INDICON 2004. First India Annual Conference, 2004.*, pages 256–261, December 2004.
- [7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, pages 1097–1105, Red Hook, NY, USA, December 2012. Curran Associates Inc.
- [8] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages 515–518, December 2001. ISSN: 1063-6919.

- [9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, June 2005. ISSN: 1063-6919.
- [10] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1627–1645, September 2010.
- [11] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv:1312.6229 [cs]*, February 2014.
- [12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv:1311.2524 [cs]*, abs/1311.2524, October 2014.
- [13] Ross Girshick. Fast R-CNN. *arXiv:1504.08083 [cs]*, abs/1504.08083, September 2015.
- [14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*, abs/1506.01497, January 2016.
- [15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot Multi-Box Detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 21–37, Cham, 2016. Springer International Publishing.
- [16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]*, May 2016.
- [17] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]*, December 2016.
- [18] Hei Law and Jia Deng. CornerNet: Detecting Objects as Paired Keypoints. *arXiv:1808.01244 [cs]*, March 2019.

- [19] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. CenterNet: Keypoint Triplets for Object Detection. *arXiv:1904.08189 [cs]*, April 2019.
- [20] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully Convolutional One-Stage Object Detection. *arXiv:1904.01355 [cs]*, August 2019.
- [21] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. Bottom-up Object Detection by Grouping Extreme and Center Points. *arXiv:1901.08043 [cs]*, April 2019.
- [22] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object Detection in 20 Years: A Survey. *arXiv:1905.05055 [cs]*, May 2019.
- [23] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv:1804.02767 [cs]*, April 2018.
- [24] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling Cross Stage Partial Network. *arXiv:2011.08036 [cs]*, November 2020.
- [25] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. *arXiv:1911.11929 [cs]*, November 2019.
- [26] Glenn Jocher. ultralytics/yolov5, April 2021. <https://github.com/ultralytics/yolov5>, accessed on 2021-01-15.
- [27] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of Tricks for Image Classification with Convolutional Neural Networks. *arXiv:1812.01187 [cs]*, December 2018.
- [28] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*, February 2015.
- [29] Byung-Gil Han, Joon-Goo Lee, Kil-Taek Lim, and Doo-Hyun Choi. Design of a Scalable and Fast YOLO for Edge-Computing Devices. *Sensors (Basel, Switzerland)*, 20(23), November 2020.

- [30] Micaela Verucchi, Gianluca Brilli, Davide Sapienza, Mattia Verasani, Marco Arena, Francesco Gatti, Alessandro Capotondi, Roberto Cavicchioli, Marko Bertogna, and Marco Solieri. A Systematic Assessment of Embedded Neural Networks for Object Detection. In *25th IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, October 2020.
- [31] Alexey Bochkovskiy. AlexeyAB/darknet, December 2020. <https://github.com/AlexeyAB/darknet>, accessed on 2021-01-15.
- [32] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]*, April 2020.
- [33] NVIDIA. Jetson Nano 4GB Developer Kit, October 2020. <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>, accessed on 2020-11-15.
- [34] John Roncoroni. UC IPM Pest Notes: Dandelion. *UC ANR*, 7469, 2018.
- [35] A. W. Hooper, G. O. Harries, and B. Ambler. A photoelectric sensor for distinguishing between plant material and soil. *Journal of Agricultural Engineering Research*, 21(2):145–155, June 1976.
- [36] W. S. Lee, D. C. Slaughter, and D. K. Giles. Robotic Weed Control System for Tomatoes. *Precision Agriculture*, 1(1):95–113, January 1999.
- [37] Hong Fu, Zheru Chi, Dagan Feng, and Jiatao Song. Machine learning techniques for ontology-based leaf classification. In *ICARCV 2004 8th Control, Automation, Robotics and Vision Conference, 2004.*, volume 1, pages 681–686 Vol. 1, December 2004.
- [38] Andres Milioto, Philipp Lottes, and C. Stachniss. Real-time Blob-wise Sugar Beers vs Weeds Classification For Monitoring Fields Using Convolutional Neural Networks. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2W3:41–48, 2017.
- [39] Abdel-Aziz Binguitcha-Fare and Prince Sharma. Crops and weeds classification using Convolutional Neural Networks via optimization of transfer learning parameters. *International Journal of Engineering and Advanced Technology*, 8(5), 2019.

- [40] Vi Nguyen Thanh Le, Selam Ahderom, and Kamal Alameh. Performances of the LBP Based Algorithm over CNN Models for Detecting Crops and Weeds with Similar Morphologies. *Sensors*, 20(8):2193, January 2020. Multidisciplinary Digital Publishing Institute.
- [41] Rekha Raja, Thuy T. Nguyen, David C. Slaughter, and Steven A. Fennimore. Real-time weed-crop classification and localisation technique for robotic weed control in lettuce. *Biosystems Engineering*, 192:257–274, April 2020.
- [42] B Chen, S Tojo, and K Watanabe. Machine Vision for a Micro Weeding Robot in a Paddy Field. *Biosystems Engineering*, 85(4):393–404, August 2003.
- [43] Ulrich Weiss, Peter Biber, Stefan Laible, Karsten Bohlmann, and Andreas Zell. Plant Species Classification Using a 3D LIDAR Sensor and Machine Learning. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 339–345, December 2010.
- [44] Ya Xiong, Yuanyue Ge, Yunlin Liang, and Simon Blackmore. Development of a prototype robot and fast path-planning algorithm for static laser weeding. *Computers and Electronics in Agriculture*, 142:494–503, November 2017.
- [45] Alexander Selberg. *Generic Object Tracking with NVIDIA Jetson Nano Using Siamese Convolutional Neural Networks*. PhD thesis, Lund University, 2020.
- [46] Mathieu Carpentier, Philippe Giguère, and Jonathan Gaudreault. Tree Species Identification from Bark Images Using Convolutional Neural Networks. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1075–1081, October 2018. ISSN: 2153-0866.
- [47] Saedi Seyed and Khosravi Hossein. A deep neural network approach towards real-time on-branch fruit recognition for precision horticulture. *Expert Systems with Applications*, 159:113594, November 2020. Pergamon.

- [48] R. Goring. *Feasibility of Neural Networks for Maritime Visual Detection on a Mobile Platform*. PhD thesis, Embry-Riddle Aeronautical University, 2017.
- [49] Elias Stein, Siyu Liu, and John Sun. Real-Time Object Detection on an Edge Device. Final Report, Stanford University, 2019.
- [50] Yunong Tian, Guodong Yang, Zhe Wang, Hao Wang, En Li, and Zize Liang. Apple detection during different growth stages in orchards using the improved YOLO-V3 model. *Computers and Electronics in Agriculture*, 157:417–426, February 2019.
- [51] A. Grebo, T. Konsa, G. Gašparović, and B. Klarin. Application of YOLO algorithm on student UAV. In *2020 5th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–6, September 2020.
- [52] Shaun M. Sharpe, Arnold W. Schumann, and Nathan S. Boyd. Goosegrass Detection in Strawberry and Tomato Using a Convolutional Neural Network. *Scientific Reports*, 10(1):9548, June 2020. Nature Publishing Group.
- [53] Bo Liu and Ryan Bruch. Weed Detection for Selective Spraying: a Review. *Current Robotics Reports*, 1(1):19–26, March 2020.
- [54] Ibrahim Babiker, Wen-Fang Xie, and Guangyi Chen. Recognition of Dandelion Weed via Computer Vision for a Weed Removal Robot. In *2019 1st International Conference on Industrial Artificial Intelligence (IAI)*, pages 1–6, July 2019.
- [55] Lorena Parra, Jose Marin, Salima Yousfi, Gregorio Rincón, Pedro Vicente Mauri, and Jaime Lloret. Edge detection for weed recognition in lawns. *Computers and Electronics in Agriculture*, 176:105684, September 2020.
- [56] Jialin Yu, Arnold W. Schumann, Zhe Cao, Shaun M. Sharpe, and Nathan S. Boyd. Weed Detection in Perennial Ryegrass With Deep Learning Convolutional Neural Network. *Frontiers in Plant Science*, 10, 2019. Frontiers.

- [57] Jialin Yu, Arnold W. Schumann, Shaun M. Sharpe, Xuehan Li, and Nathan S. Boyd. Detection of grassy weeds in bermudagrass with deep convolutional neural networks. *Weed Science*, 68(5):545–552, September 2020. Cambridge University Press.
- [58] Bill Williams. PiCameraApp, 2018. <https://github.com/Billwilliams1952/PiCameraApp>, accessed on 2021-11-15.
- [59] Joseph Nelson. Roboflow: Everything you need to start building computer vision into your applications, January 2021. Version 1.0, <https://roboflow.ai>, accessed on 2021-01-15.
- [60] Alexey Bochkovskiy. YOLOv4-tiny released: 40.2% AP50, 371 FPS (GTX 1080 Ti), 1770 FPS tkDNN/TensorRT, October 2020. <https://github.com/AlexeyAB/darknet/issues/6067>, accessed on 2020-12-15.
- [61] Lukas Biewald. Experiment Tracking with Weights and Biases, 2020. <https://www.wandb.com>, accessed on 2021-02-15.