Virtual Crop Development and Automated Weed Detection for Precision Agriculture

Arun Soma[1], Eduardo Colmenares[2]
Department of Computer Science
Midwestern State univeristy
Wichita Falls, TX, USA
arunedusoma@gmail.com[1]
eduardo.colmenares@msutexas.edu[2]

Heng Wu[3]
Department of Computer Science
University of La Verne
La Verne, CA, USA
hwu@laverne.edu

Abstract

Artificial Intelligence (AI) constitutes one of the most recent major advancements in technology, transforming research, teaching, and traditional practices across various fields. One area where AI is already making a significant impact and is projected to continue growing is advanced precision agriculture. This research employs an AI-assisted workflow to accelerate the implementation of algorithms that utilize convolutional neural networks (CNN) and real-time object detection models to accurately differentiate between useful crops and harmful weeds. The study is divided into two phases. The first phase, covered in this paper, focuses on creating a virtual crop environment that serves as the foundation for training and validating the model. The second phase, planned for future work, aims to integrate the trained model into a drone system for autonomous, targeted spraying, enhancing the efficiency and sustainability of agricultural practices.

Keywords: Agriculture, CNN, Artificial Intelligence, Adam, Weed, Crops.

Regular Research Paper
Track: Research in AI -AI Applications (Agriculture)

1 Introduction

Precision agriculture is transforming modern farming by leveraging advanced technologies to optimize crop management, reduce environmental impact, and enhance yield efficiency. Traditional farming methods, including manual weed control, are labor-intensive and often result in excessive herbicide use, environmental degradation, and inefficiencies in resource allocation. AI-driven solutions offer transformative alternatives by employing machine learning techniques to identify and target weeds with remarkable precision. The integration of UAV imagery, as demonstrated by Haq [1] and Singh et al. [2], significantly enhances the identification of small and scattered weed patches, addressing a critical challenge in traditional farming. By automating weed detection and herbicide spraying,

AI-powered drones allow farmers to focus on other essential tasks, ultimately boosting productivity and improving their overall well-being. This study introduces a CNN-based weed detection framework, highlighting its potential for integration into drone-based systems to further optimize precision agriculture.

Companies like John Deere are at the forefront of incorporating AI-driven precision agriculture into large-scale farming machinery. Their autonomous tractors and sprayers utilize advanced computer vision and machine learning to optimize field operations. The See & Spray technology, for instance, significantly reduces herbicide usage by targeting only weeds instead of blanketing entire fields, cutting chemical use by up to 90 percent. This innovation not only enhances efficiency but also lowers operational costs and reduces environmental impact. While traditional farming methods rely heavily on manual labor, less precise irrigation, and limited technological intervention, AI-driven solutions bridge the gap between sustainability and efficiency. By incorporating advanced sensor technologies and real-time monitoring systems, farmers can optimize resource allocation, improve crop health management, and reduce labor dependency. The automation of these processes allows farmers to reallocate their time to strategic activities such as attending educational workshops, researching improved agricultural techniques, and refining overall farm management practices.

2 Understanding Convolutional Neural Networks

Convolutional Neural Networks (CNNs) emulate the way the human brain processes visual information, identifying and classifying objects based on features [3, 4]. In simple words, a human brain classifies any image it sees based on the features it detects first. CNNs work similarly by extracting hierarchical features through successive layers [5].

At the core of CNNs is the convolution operation, which applies filters (or kernels) to input images, extracting key features such as edges and textures [3]. As shown in Figure 1 from [6], a smiley face image can be represented as a matrix where darkly lit pixels are denoted as ones and the remaining pixels as zeros.

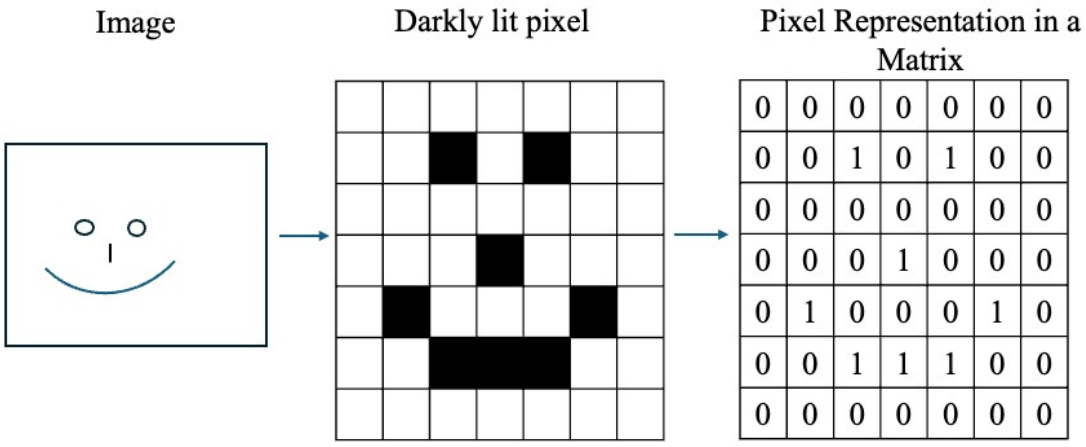


Figure 1: Transformation of an Image into Pixel Representation: From Visual Illustration to a Matrix.

As illustrated in Figure 2 from [6], the feature map is generated by the feature detector sliding across the input image in strides. In this example, the stride is set to a

single element hop, where the light blue region represents the first window stride of the detector. The element-wise calculation for this window, demonstrating the multiplication and summing of values, is highlighted in light orange [7]. This process reduces the spatial dimensions of the image while preserving key features. In CNNs, such feature maps are stacked across convolutional layers to progressively learn more complex patterns [8].

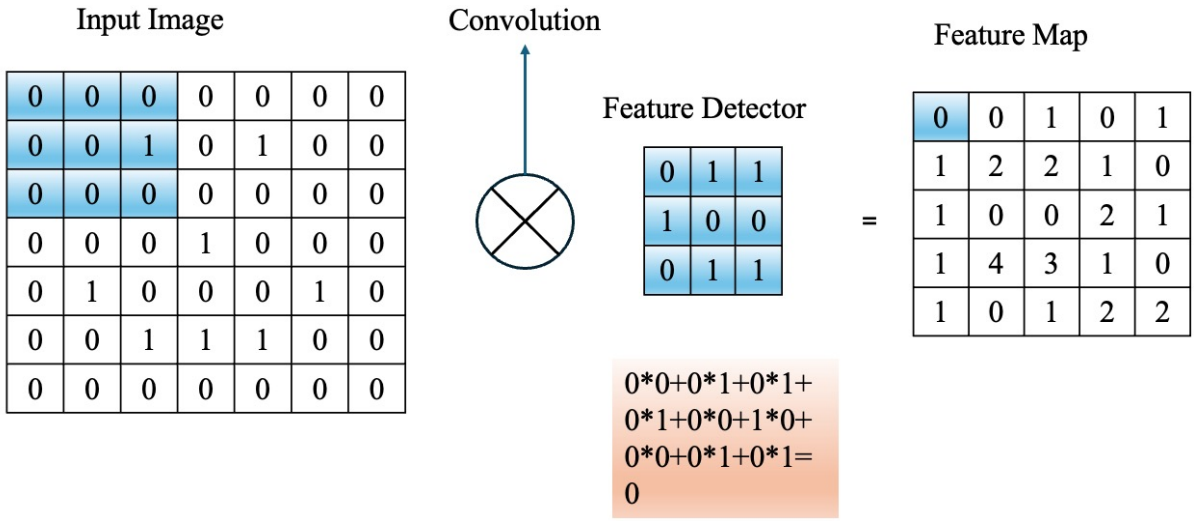


Figure 2: Convolution operation generating a feature map.

Pooling follows convolution to refine the feature maps further. Pooling reduces the spatial dimensions of data while retaining the most critical information. Figure 3 from [6] illustrates max pooling with a 2×2 window and a stride of 2, where each stride is highlighted in a different color [7]. If the final stride cannot fully accommodate the pooling window, it adjusts to the available size. Max pooling selects the highest value in each window, thereby retaining the key features while reducing spatial dimensions and ensuring invariance to minor distortions.

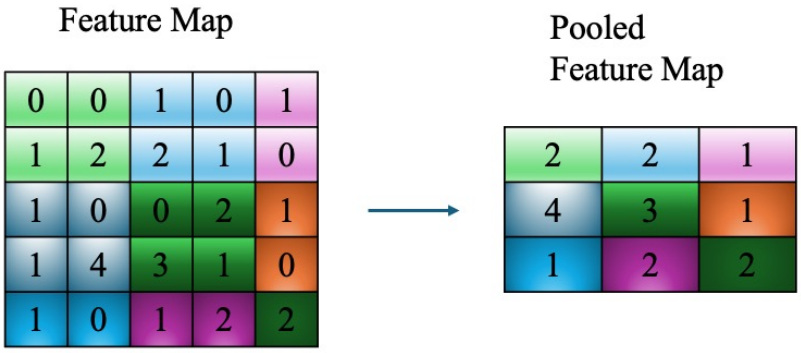


Figure 3: Image explaining Max Pooling.

After pooling, flattening converts the pooled feature maps into a one-dimensional vector, as shown in Figure 4 from [6]. This vector is then fed into fully connected layers for classification [9, 10]. These layers combine features to make the final predictions—identifying attributes like a nose or eyes—and iteratively refining accuracy through backpropagation.

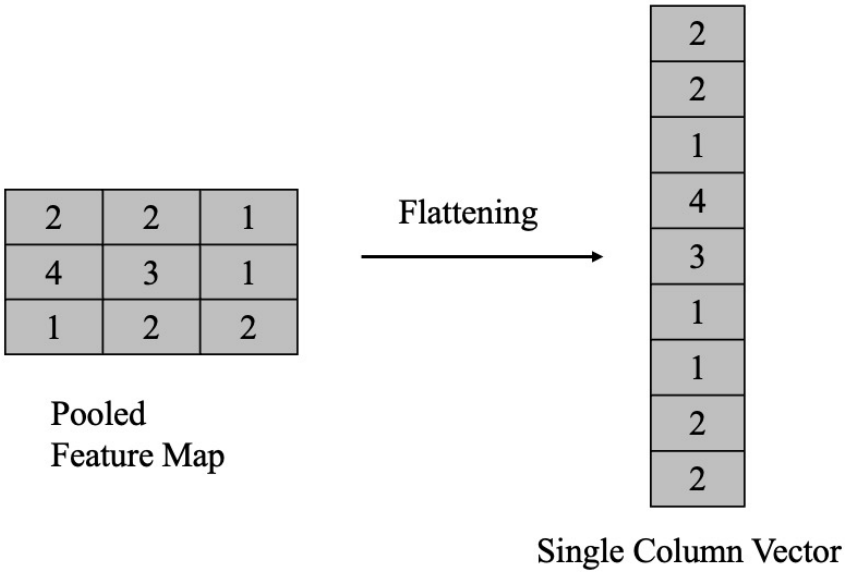


Figure 4: Demonstration of the flattening process before feeding the pooling layer output to the fully connected layer.

3 Methodology

3.1 Data Collection

This research utilized manually collected imagery, capturing over 5,000 images of weeds and non-weeds using a Pentax camera. Access to local fields was facilitated by a collaborating farmer. These images were labeled and organized into different datasets.

(a) Example of a collected weed image.

(b) Example of a collected non-weed image.

Figure 5: Sample images from the manually collected dataset.

3.2 Image Preparation and Dataset Organization

In our research on weed classification using a Convolutional Neural Network (CNN), the `os` [11]and `shutil` [12] libraries were extensively utilized for file and directory management, facilitating the systematic preprocessing and organization of image data. These libraries ensured an efficient workflow for training, validation, and testing by structuring the dataset in a reproducible manner.

The dataset was manually created, with each image carefully examined and placed into the appropriate category—either "weeds" or "non-weeds"—based on visual assessment. This manual sorting process was crucial in ensuring high-quality labels, providing a more reliable foundation for training the model. By directly observing and classifying the images, researchers developed a deeper understanding of the dataset, reinforcing classification accuracy and minimizing potential labeling errors.

To provide a detailed overview of the data preparation process, the complete pseudocode is presented in Algorithm 1. This pseudocode outlines the sequential steps—from defining file paths and dataset directories to copying images into the proper splits. By leveraging this preprocessing pipeline, the CNN model was trained on a well-organized and balanced dataset, ensuring robustness in hyperparameter tuning and model evaluation.

Algorithm 1 Data Preparation for Weed Classification

Define file paths:
$weed\_path \leftarrow$ "/home/username/WeedDetection/weeds"
$non\_weed\_path \leftarrow$ "/home/username/WeedDetection/NonWeeds"
$base\_dir \leftarrow$ "/home/username/WeedDetection/data"
Define dataset directories:
$train\_dir \leftarrow base\_dir +$ "/train"
$val\_dir \leftarrow base\_dir +$ "/validation"
$test\_dir \leftarrow base\_dir +$ "/test"
procedure EnsureDirectories($directories, classes$)
    for all $directory \in directories$ **do**
        for all $class\_name \in classes$ **do**
            $class\_path \leftarrow directory +$ "/" $+ class\_name$
            if $class\_path$ does not exist **then**
                Create directory $class\_path$
            end if
        end for
    end for
end procedure
procedure SplitData($src\_path, class\_name, train\_dir, val\_dir, test\_dir, split\_ratio = (0.7, 0.15, 0.15)$)
    $all\_images \leftarrow []$
    for all $subfolder \in src\_path$ **do**
        $subfolder\_path \leftarrow src\_path +$ "/" $+ subfolder$
        if $subfolder\_path$ is a directory **then**
            $images \leftarrow []$
            for all $img \in subfolder\_path$ **do**
                $full\_path \leftarrow subfolder\_path +$ "/" $+ img$
                if $full\_path$ is a file **then**
                    Add $full\_path$ to $images$
                end if
            end for
            Add $images$ to all_images
        end if
    end for
    $train\_size \leftarrow 0.7 \times \text{len}(all\_images)$
    $val\_size \leftarrow 0.15 \times \text{len}(all\_images)$
    $train\_images \leftarrow all\_images[0 : train\_size]$
    $val\_images \leftarrow all\_images[train\_size : train\_size + val\_size]$
    $test\_images \leftarrow all\_images[train\_size + val\_size :]$
    procedure CopyImages($images, target\_dir$)
        for all $image \in images$ **do**
            $target\_path \leftarrow target\_dir +$ "/" $+ class\_name +$ "/" $+ image\_name$
            if $target\_path$ does not exist **then**
                Copy $image$ to $target\_path$
            end if
        end for
    end procedure
    Call CopyImages($train\_images, train\_dir$)
    Call CopyImages(val_images, val_dir)
    Call CopyImages($test\_images, test\_dir$)
end procedure
Execution: Ensure directories and split data
Call EnsureDirectories([$train\_dir, val\_dir, test\_dir$], ["weeds", "non_weeds"])
Call SplitData($weed\_path$, "$weeds$", $train\_dir, val\_dir, test\_dir$)
Call SplitData(non_weed_path, "non_weeds", $train\_dir, val\_dir, test\_dir$)

The `os` module was primarily responsible for defining and managing file paths. The dataset was systematically structured using `os.path.join()` [11], ensuring compatibility across different operating systems. To guarantee the proper organization of training, validation, and testing datasets, `os.makedirs()` was employed to create directories if

they did not already exist. Additionally, `os.listdir()` enabled recursive folder traversal, allowing the script to identify subdirectories and retrieve corresponding images while filtering out non-directory elements.

The `shutil` module played a crucial role in handling file operations such as copying images into their respective dataset splits. After retrieving the images, the dataset was divided based on predefined ratios of 70% for training, 15% for validation, and 15% for testing. The `shutil.copy()` function facilitated the efficient transfer of images to their designated directories while preserving the original dataset [12]. This structured approach ensured consistency across different experimental runs, enhancing the reliability of model training.

To balance detail retention with computational efficiency, images were resized to 150×150 pixels, ensuring faster processing while maintaining necessary visual features. A Python script automated dataset preparation, iterating through the manually created folders for weeds and non-weeds and systematically allocating images into the predefined dataset splits. The integration of `os` and `shutil` enabled seamless scalability, allowing researchers to expand or modify the dataset structure without manual intervention. By leveraging this preprocessing pipeline, the CNN model training was conducted on a well-organized and balanced dataset, ensuring robustness in hyperparameter tuning and model evaluation.

3.3 Model Training

The Convolutional Neural Network (CNN) developed in this study was designed to classify manually captured images of weeds and non-weeds. Python 3.8, TensorFlow 2.0, and the Keras API were employed for model definition and training, with the dataset split into training (70%), validation (15%), and testing (15%) subsets, as shown in algorithm 2. The preprocessing steps included resizing images to 150 × 150 pixels and Normalizing pixel values in the range from zero to one, ensuring consistency and facilitating the training process.

The CNN architecture comprised three convolutional layers, each followed by a max-pooling layer. The first convolutional layer used 32 filters, the second 64, and the third 128, all with 3 × 3 kernels. This configuration enabled the network to extract features of increasing complexity, starting with basic edges in initial layers and advancing to intricate patterns in deeper layers. Max-pooling layers with a 2 × 2 window were employed after each convolutional layer to downsample feature maps, reducing computational cost while retaining essential information. The extracted feature maps were flattened and passed into a fully connected layer with 128 neurons to learn high-level patterns.

To mitigate overfitting, a dropout layer with a 50% dropout rate was added before the output layer, which consisted of a single neuron with a sigmoid activation function, generating probabilities for binary classification. The ReLU activation function was used throughout the convolutional and fully connected layers to introduce non-linearity, enabling the network to capture complex relationships in the data.

The model was compiled with the Adam optimizer for adaptive learning, a binary cross-entropy loss function for binary classification, and accuracy as the evaluation metric. Early stopping monitored validation loss and halted training after five epochs of no improvement, preventing overfitting and conserving computational resources. A model checkpoint saved the best-performing version of the model during training. The training process spanned up to 30 epochs with a batch size of 32, balancing computational

efficiency with model performance. The use of rigorous hyperparameter tuning, as highlighted by Haq [1], ensured high classification accuracy, demonstrating the effectiveness of this approach for weed detection.

Algorithm 2 CNN Model Training

```
Load and preprocess dataset
Split dataset into training (70%), validation (15%), and testing (15%) subsets
Define CNN architecture with three convolutional layers and max-pooling
Add fully connected layer and dropout for regularization
Compile model using Adam optimizer and binary cross-entropy loss
Initialize early stopping and model checkpointing
for epoch = 1 to 30 do
    Train model on training data
    Validate model on validation data
    if validation loss does not improve for 5 epochs then
        Stop training
    end if
    Save best-performing model checkpoint
end for
Save final trained model
```

Checkpointing was an essential part of the training process. The model's best weights, based on validation performance, were saved during training, preventing the need to retrain the model and ensuring that the most optimal model is kept for future use. This mechanism ensured that the model at the epoch where validation loss was lowest was preserved, even if later epochs led to overfitting or degradation in performance.

After training, the final model was saved to disk, ensuring reproducibility and future use for real-world weed detection applications.

3.3.1 ReLU Activation Function

The acronym ReLU stands by Rectified Linear Unit and is one of the most popular and widely used activation functions. Activation functions plays a critical role in deciding if a neuron will be active or not, this occurs by introducing non linearity to the model. If non-linearity is not present, then the neuronal network would perform linear regression independent of the number of existing layers. In simple words, the network would behave like a single linear function, meaning, that it could only model simple relationships. However, a vast majority of the data is in the form of text and images present non-linear complex patterns such as edges and curves which require more complex modeling in order to properly capture its features [10, 13].

ReLU achieves non-linearity by introducing a strong shape change at x = 0, for x-values >0 the functions returns x (linear behavior), and for x-values <=0 the function returns 0. This relative minimum introduces a strong non-linearity behavior which favors the stacking of layers and the learning of complex features [10, 13]. This activation function is crucial for Convolutional Neuronal Networks since it allows the transformation of simple linear models into non-linear models capable of recognizing data with intricate patterns such as images and text. Despite its positive characteristics, ReLU can present some problems, such as "dying ReLU" and "unbounded outputs" among several others [10, 13].

3.3.2 Binary Cross Entropy

Binary Cross Entropy (BCE) is a loss function used for binary classification problems. A binary classification problem involves making a decision between two classes, for example,

(weed vs no-weed), (cat vs dog), (apple vs orange) [14]. BCE is defined as:

$$\text{BCE Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{1}$$

where y_i is the actual expected value (0 or 1), $\hat{y}_i$ is the predicted probability (between 0 and 1), and N is the total number of samples. For detailed information, please refer to [14].

3.3.3 Adam

Adaptive Moment Estimation (Adam) is a widely utilized optimization algorithm for gradient descent, designed to enhance convergence efficiency and stability. Adam integrates two key techniques to improve upon traditional gradient descent. First, it employs momentum, which retains a moving average of past gradients, thereby smoothing updates and mitigating erratic fluctuations. Second, it utilizes an adaptive learning rate, wherein step sizes are adjusted individually for each parameter in the model, optimizing learning dynamics [13]. The synergy of these techniques enables Adam to achieve greater reliability, efficiency, and robustness, particularly in the presence of noisy data, making it a superior alternative to conventional gradient descent methods. For more information about gradient descent optimization algorithms, refer to [13].

3.4 Device Configuration

3.4.1 CPU Configuration

The CPU training part of the model was conducted on a MacBook Air M1 (2020 Model) running macOS Sonoma 14.7. The system is powered by an Apple M1 Chip, which features an 8-core CPU (arranged as 4 high-performance cores and 4 efficiency cores) and 8 GB of unified memory. This information is included to ensure reproducibility. For this training, TensorFlow was forced to use the CPU by explicitly specifying the computation device with `tf.device('/CPU:0')` [15, 16]. Under this configuration, training the CNN for 30 epochs required approximately 1 hour and 53 minutes.

3.4.2 GPU Configuration

For accelerated training, the integrated GPU on the Apple M1 was utilized in conjunction with the CPU. The MacBook Air GPU comprises 7 cores and supports Metal 3, Apple's advanced graphics and compute API designed to provide efficient, low-overhead access to the GPU for both rendering and data processing tasks. Metal 3 enhances performance by reducing latency and optimizing resource usage, which is particularly beneficial for computationally intensive tasks such as CNN training. By specifying `tf.device('/GPU:0')` [15, 16], TensorFlow was forced to compute selected sections of the code with the GPU instead of CPU, reducing the training time to 37 minutes for 10 epochs (with early stopping enabled). These performance gains highlight the advantages of GPU acceleration for complex tasks, enabling faster iterations and making large-scale model training more feasible. These enhancements align with findings reported in [1].

4 Results

4.1 Model Performance

The evaluation metric of precision was chosen for this study due to its emphasis on minimizing false positives, which is particularly critical in weed detection tasks. Misclassifying non-weeds as weeds can result in unnecessary herbicide application, leading to resource wastage and potential harm to crops. Precision provides a reliable measure of the proportion of true positive predictions among all positive predictions, ensuring the model prioritizes accurate identification of weeds. The CNN model was trained under two computational configurations to assess the impact of device usage on training efficiency and performance. Both configurations used the same project structure and dataset. The only difference was the allocation of computational resources during the training process. Across all configurations, the model consistently achieved a precision of 98%, evaluated over 50 test runs.The first configuration used only the CPU, which required 1 hour and 57 minutes to complete 30 epochs. The second configuration, which employed the CPU for data preprocessing and the GPU for model training, was the most efficient. This setup completed training in 37 minutes, running only 10 epochs due to early stopping when the validation loss stabilized.

Despite the differences in training time and computational setups, both models achieved the same precision, demonstrating the robustness of the CNN architecture and the reliability of the dataset. The use of GPUs significantly reduced training time, particularly in the hybrid configuration where the CPU handled data preprocessing and the GPU was used for training. This setup proved to be the most efficient without compromising performance. The results are summarized in Table 1.

Configuration	**Training Time**	**Epochs**	**Precision**
CPU Only	1h 57mins	30	98%
CPU for Data Prep, GPU for Training	37 mins (early stopping)	10	98%

Table 1: Training Configurations and Performance Metrics.

These findings highlights the importance of computational resource optimization in training machine learning models, where hybrid configurations can significantly enhance efficiency without sacrificing accuracy.

4.2 Virtual Crop Configuration

The virtual crop configuration provides a dynamic and adaptable framework for visualizing the model's predictions. Each time the model runs, the data is reshuffled, presenting a new spatial arrangement of weeds and non-weeds. This setup ensures variability in testing and allows the model to see a different crop scenario during every run.

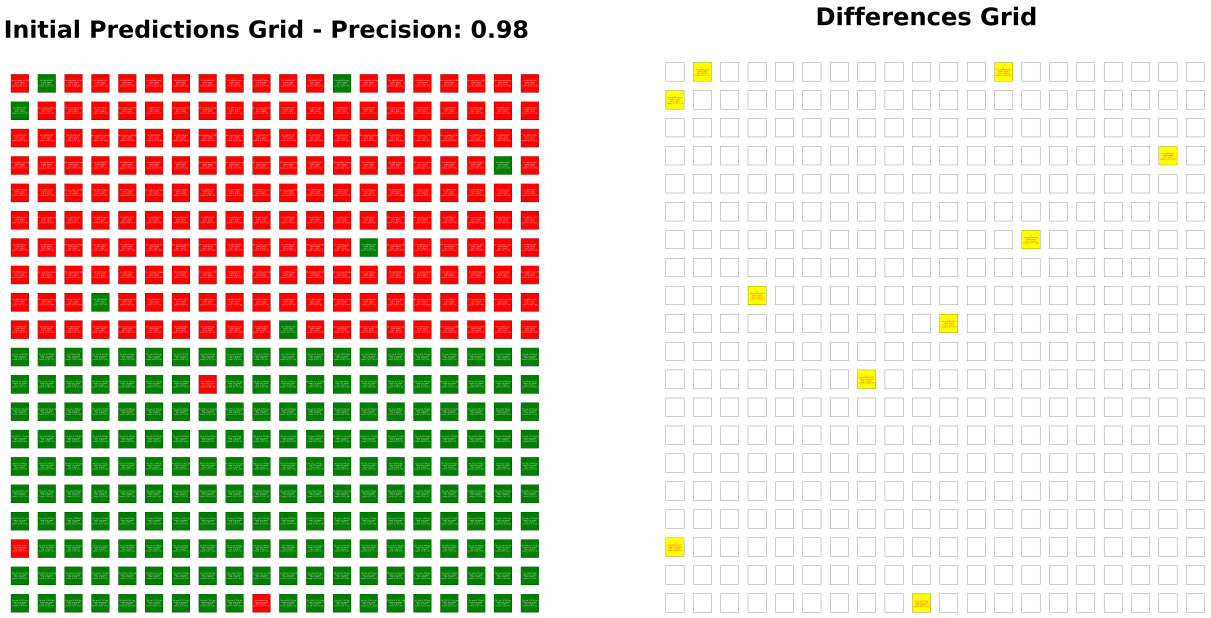
The spatial coordinates, acting as spoofed latitudes and longitudes, add a layer of realism and pave the way for future drone integration. These coordinates will eventually assist in planning the shortest path for field coverage during herbicide spraying.

The significance of the virtual crop lies in its ability to replicate real-world conditions and variability. Unlike static datasets, this approach allows the model to encounter dynamic scenarios, which enhances its ability to generalize to unseen data. By simulating

the randomness of crop and weed distributions, the virtual crop helps build a more robust and adaptable weed detection system. Additionally, it bridges the gap between controlled testing environments and practical applications in agricultural fields, where factors like spatial arrangement and randomness play a critical role in determining model effectiveness.

Figure 6 presents the results of an analysis performed on an initial virtual crop where the top 50% (10x20 matrix)of the plants are weeds , while the bottom 50% (10x20 matrix) of the plants are valid crops. This initial distribution was purposely design that way to provide a basic, easy but conclusive example.

Figure 6a, represents the predicted classifications, with red for predicted weeds and green for predicted non-weeds. Each grid cell includes details such as the predicted class, true label, and spatial coordinates. This provides a clear visualization of where weeds and non- weeds are detected within the grid, simulating their distribution in a virtual field.



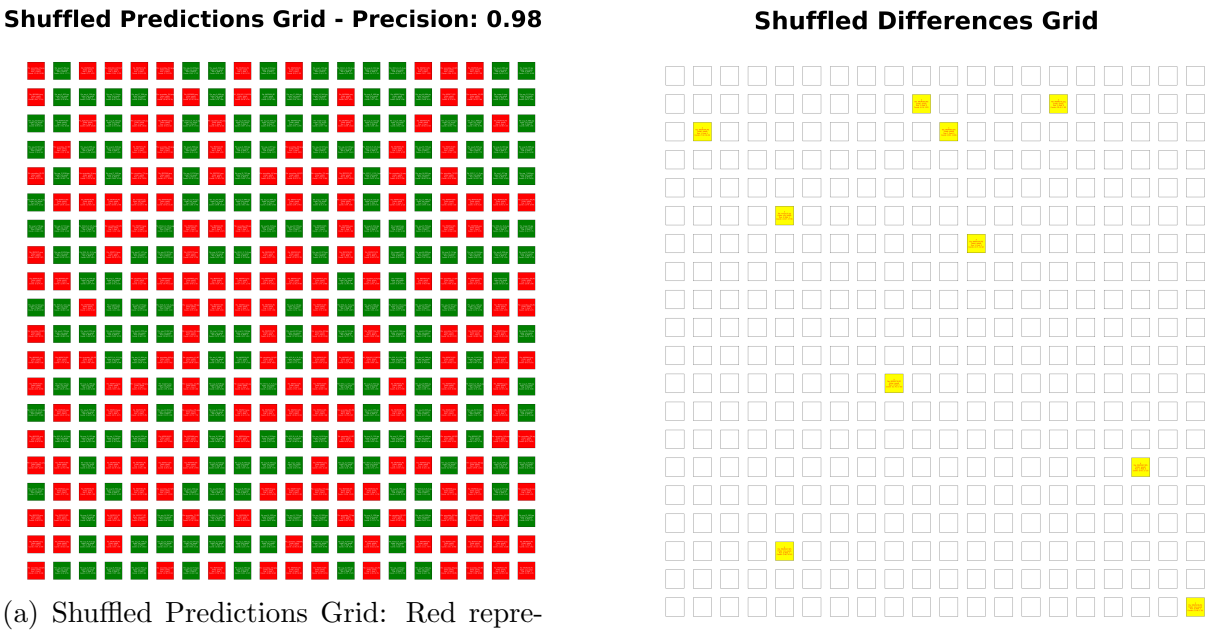
(a) Initial Predictions Grid: Red indicates predicted weeds, green indicates predicted non-weeds, with spatial details in each cell.

(b) Differences Grid: Yellow cells highlight misclassified predictions for error evaluation.

Figure 6: Visualizations of Initial Model Predictions and Error Analysis.

Additionally, the differences grid, as shown in Figure 6b, highlights only the false predictions. These mismatches between the true and predicted labels are marked in the grid, making it easy to identify mis- classified plants. This error visualization allows for straightforward evaluation of the model's performance and areas for improvement, contributing to the refinement of the weed detection system.

To further validate the model's robustness, a shuffled test dataset was used to reassess performance under different spatial distributions. This reshuffling ensures that the model does not learn a specific fixed order of images but instead generalizes well across different field arrangements. The shuffled prediction grid (Figure 7a) demonstrates how the model adapts to a randomly reorganized dataset, while the differences grid (Figure 7b) highlights misclassified predictions under this altered setup.



(a) Shuffled Predictions Grid: Red represents predicted weeds, green represents predicted non-weeds, including updated spatial details.

(b) Shuffled Differences Grid: Yellow cells highlight misclassified predictions in the reshuffled dataset.

Figure 7: Visualizations of Shuffled Model Predictions and Error Analysis.

5 Conclusion and Future Work

This research demonstrates the transformative potential of AI in agriculture, specifically through the development of a CNN-based weed detection system. By integrating a virtual crop framework, the model achieved consistent precision of 98% across various computational configurations, validating its robustness and adaptability for real-world applications. The virtual crop environment proved effective in simulating field variability, reshuffling datasets, and incorporating spoofed spatial coordinates to mimic realistic conditions. These features enable better generalization and lay a foundation for future drone-based weed detection systems.

Table 1 illustrates the improved training efficiency achieved by utilizing a combined approach, where the CPU handles data preparation while the GPU performs model training. This hybrid strategy results in a faster training speed compared to the CPU-only approach. The results indicate that training converges more rapidly with GPU acceleration, suggesting that the model reaches its optimal learning capacity in a shorter duration. This reduction in training time is reflected in the fewer required epochs. Consequently, additional training is unlikely to yield significant improvements in performance.

At the time of submission, the researchers were actively investigating whether adopting the Leaky ReLU activation function could improve performance and accuracy. It is well established that, in certain scenarios, the conventional ReLU activation function is prone to the 'dying ReLU' problem, wherein some neurons consistently output zero, rendering them inactive and adversely affecting the learning process. A common mitigation strategy for this issue is Leaky ReLU, which introduces small, nonzero outputs for negative inputs, thereby maintaining neuron activity throughout training.

The development and implementation of a virtual crop proved to be an effective strategy for advancing the research while overcoming the physical constraints imposed

by the necessary drone hardware. The use of the virtual crop enabled both the training and validation of the model across multiple crop variants, thus enhancing its robustness and adaptability. Furthermore, this approach facilitated the creation of a software artifact that aligns with Phase 2 of future work, which involves the deployment of an unmanned aerial vehicle (UAV) for targeted spraying operations.

Future advancements will focus on integrating the weed detection model into an autonomous drone platform equipped with high-resolution cameras and GPS. This integration will optimize herbicide application paths and ensure efficient resource usage. Additionally, efforts will include field testing and expanding the system's capabilities to handle diverse crops and weed types. By addressing real-world challenges through field testing and expanding the system's capabilities, this research aims to contribute to sustainable and efficient farming solutions. The combination of AI, drone technology, and precision agriculture offers a scalable pathway toward addressing critical challenges in modern agriculture, ultimately improving crop yield and reducing environmental impact.

References

[1] M. A. Haq, "Cnn based automated weed detection system using uav imagery," *Computer Systems Science & Engineering*, 2021.

[2] V. Singh *et al.*, "Efficient application of deep neural networks for identifying small and multiple weed patches using drone images," *IEEE Access*, 2021.

[3] "Convolutional neural networks," CS231n: Convolutional Neural Networks for Visual Recognition. [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[4] S. Raschka, "What is a neural network?" *Towards Data Science*, [Online]. Available: https://medium.com/towards-data-science/what-is-a-neural-network-6010edabde2b.

[5] A. Sharma, "A comprehensive guide to convolutional neural networks—the eli5 way," *Analytics Vidhya*, Dec 2018, [Online]. Available: https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/.

[6] Educative, "Introduction to convolutional neural networks," building Advanced Deep Learning and NLP Projects. Available at: https://www.educative.io.

[7] R. Jaiswal, "Convolutional neural network: A step-by-step guide," *Towards Data Science*, [Online]. Available: https://medium.com/towards-data-science/convolutional-neural-network-a-step-by-step-guide-a8b4c88d6943.

[8] H. S. Chatterjee, "A basic introduction to convolutional neural network," *Medium*, [Online]. Available: https://medium.com/@himadrisankarchatterjee/a-basic-introduction-to-convolutional-neural-network-8e39019b27c4.

[9] J. Wu, "Introduction to convolutional neural networks," LAMDA Group, National Key Lab for Novel Software Technology, Nanjing University, China, Tech. Rep., [Online]. Available: https://cs.nju.edu.cn/wujx/paper/CNN.pdf.

[10] A. Gulli, A. Kapoor, and S. Pal, *Deep Learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and More with TensorFlow 2 and the Keras API*, 2nd ed. Birmingham, UK: Packt Publishing, 2019.

[11] "os — Miscellaneous Operating System Interfaces,," Python Software Foundation. [Online]. Available: https://docs.python.org/3/library/os.html.

[12] "shutil — High-Level File Operations,," Python Software Foundation. [Online]. Available: https://docs.python.org/3/library/shutil.html.

[13] S. Ruder, ""An Overview of Gradient Descent Optimization Algorithms," arXiv preprint, arXiv:1609.04747, 2016. [Online]. Available: https://arxiv.org/abs/1609.04747."

[14] R. Sarwar, "Understanding binary cross-entropy/log loss: A visual explanation," *Towards Data Science*, [Online]. Available: https://medium.com/towards-data-science/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[15] "Using Gpus with Tensorflow," TensorFlow. [Online]. Available: https://www.tensorflow.org/guide/gpu.

[16] "tf.device,," TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/device.

[17] "ReLU-Function and Derived Function Review," SHS Web of Conferences, vol. 131, 2022. [Online]. Available: https://www.shs-conferences.org/articles/shsconf/pdf/2022/14/shsconf_stehf2022_02006.pdf.