CHAPTER – IV

METHODOLOGY

4.1 Introduction: An Overview of the Chapter

The methodological approach for creating and validating the Convolutional Neural Network (CNN)-based system for classifying weeds and crops is described in this chapter. This chapter offers a thorough explanation of the model architecture, the research design, and the several methods used to accomplish the study's goals.

We begin by detailing the research design, explaining the rationale behind the chosen methodology and how it aligns with the research objectives. Following this, we delve into the specifics of the CNN architecture, particularly focusing on the YOLOv8 model, which is central to our approach. The chapter also covers the training process, including the selection of hyperparameters, optimization techniques, and the strategies used to prevent overfitting.

Additionally, this chapter has a section on the validation methods and assessment metrics used to evaluate the model's performance. We go over how crucial these indicators are to giving us a thorough grasp of the model's recall, accuracy, precision, and overall efficacy.

Lastly, we explore the comparative analysis conducted between our proposed model and existing models, such as AlexNetOWTBn, VGG16, and YOLOv3. This comparison highlights the advantages and potential limitations of our approach, providing a context for interpreting the results presented in the subsequent chapter.

This chapter tries to give a clear and repeatable framework for creating a reliable and efficient CNN-based precision agriculture system by going into great detail on the approach.

4.2 Model Development

With layers like convolutional, pooling, and fully connected layers, CNNs are effective tools for classifying images. For our agricultural classification jobs, we employ pre-trained models through techniques like transfer learning, callbacks to optimize the training process, and ImageGenerators to load and preprocess data efficiently.

A thorough process for developing and assessing deep learning models for precision agricultural picture classification is shown in Figure 4. To improve robustness and generalization, it begins with a dataset of annotated pictures that are subjected to data augmentation techniques including rotation and flipping. Three sets of augmented data are created: training, validation, and testing. The validation set is used to adjust hyperparameters and avoid overfitting, while the training phase teaches the model to recognize patterns and features from the training data. The test data is then used to analyze the final trained model in order to guarantee an objective evaluation of performance.

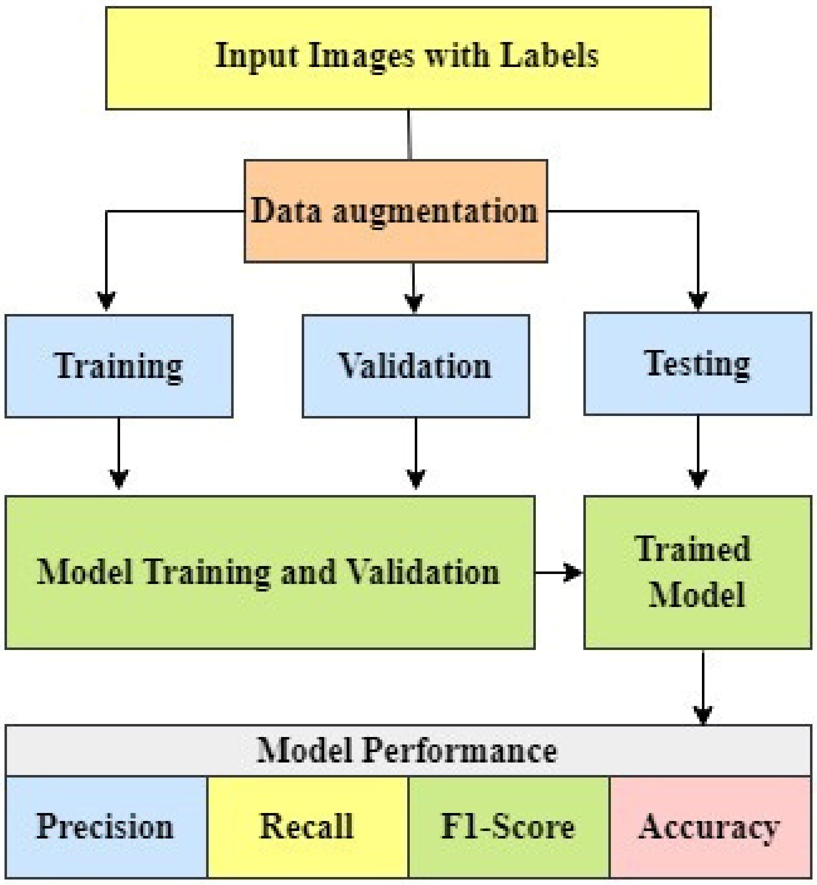

Fig. 4 : Steps in building deep learning models

We create four distinct models using these procedures for building deep learning models, and we compare them using the following performance metrics: accuracy (Equation (4)), recall (Equation (2)), F1-score (Equation (3)), and precision (Equation (1)). The precision of a set of items indicates its relevance, recall the percentage of true positives that are correctly detected, the F1-score strikes a balance between recall and precision, and accuracy gauges the overall accuracy of forecasts. This structured approach ensures that the model not only learns effectively but also performs reliably in real-world applications, enhancing resource management and decision-making in agricultural practices. The most efficient model from this comparative study will be selected for the classification task, optimizing the system's overall accuracy and effectiveness.

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

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

$$F1 - Score = \frac{2\times Precision\times Recall}{Precision+Recall} \quad (3)$$

$$Accuracy = \frac{TP+TN+FP+FN}{TP+TN} \quad (4)$$

The detailed discussion of the aforementioned equations will be provided in the subsequent sections of this chapter.

4.2.1 Description of the Models Used

This section describes the many models used to classify crops and weeds and estimate their densities. Five distinct models were utilized, each tailored for specific tasks, and the following subsections provide a comprehensive description of each model.

Model-1: Customized CNN from Scratch

A bespoke Convolutional Neural Network (CNN) created from scratch makes up the initial model. The purpose of this model was to categorize photos of weeds and crops. Multiple convolutional layers make up the architecture, which is then followed by pooling layers that take features out of the input images. The classification is done by fully connected layers at the end of the network. When comparing performance, this model acts as the reference point.

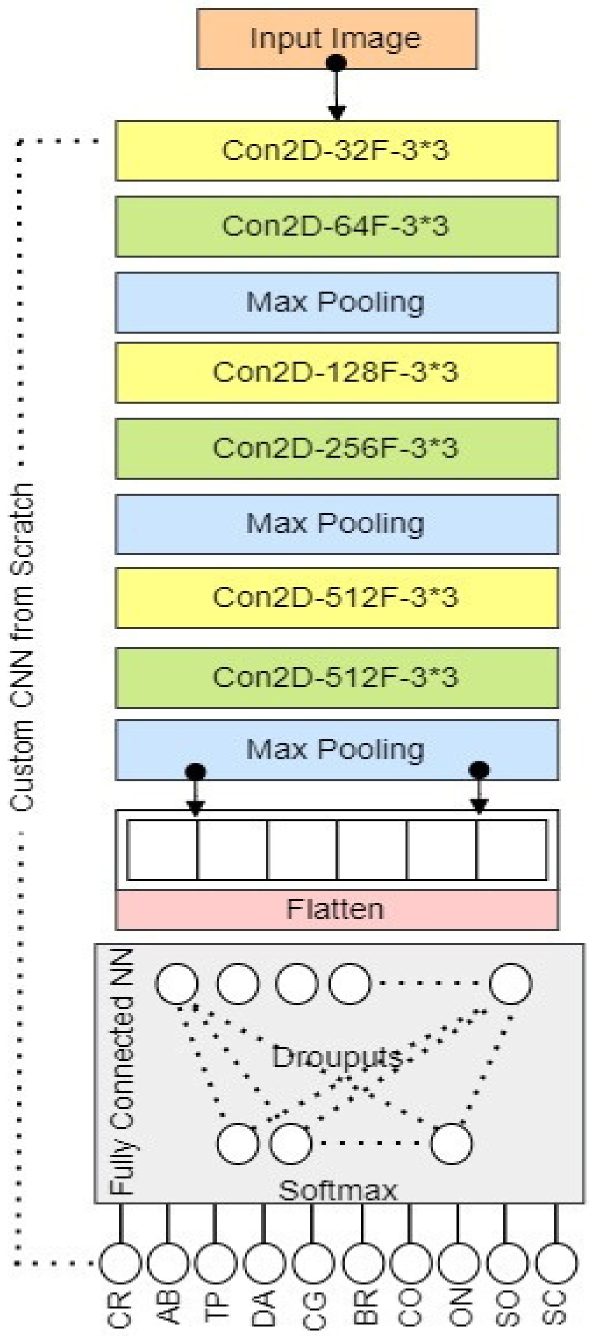

Fig. 4.1 : The architecture of Model-1: Customized CNN from Scratch

Model-2: Customized CNN from Scratch with Image Augmentation

Building upon the first model, the second model incorporates image augmentation techniques to enhance the training data. Image augmentation entails randomly transforming the input images, such as flipping, rotating, and zooming in order to improve the resilience of the model and diversify the training set. This approach helps mitigate overfitting and improves generalization to new, unseen images.

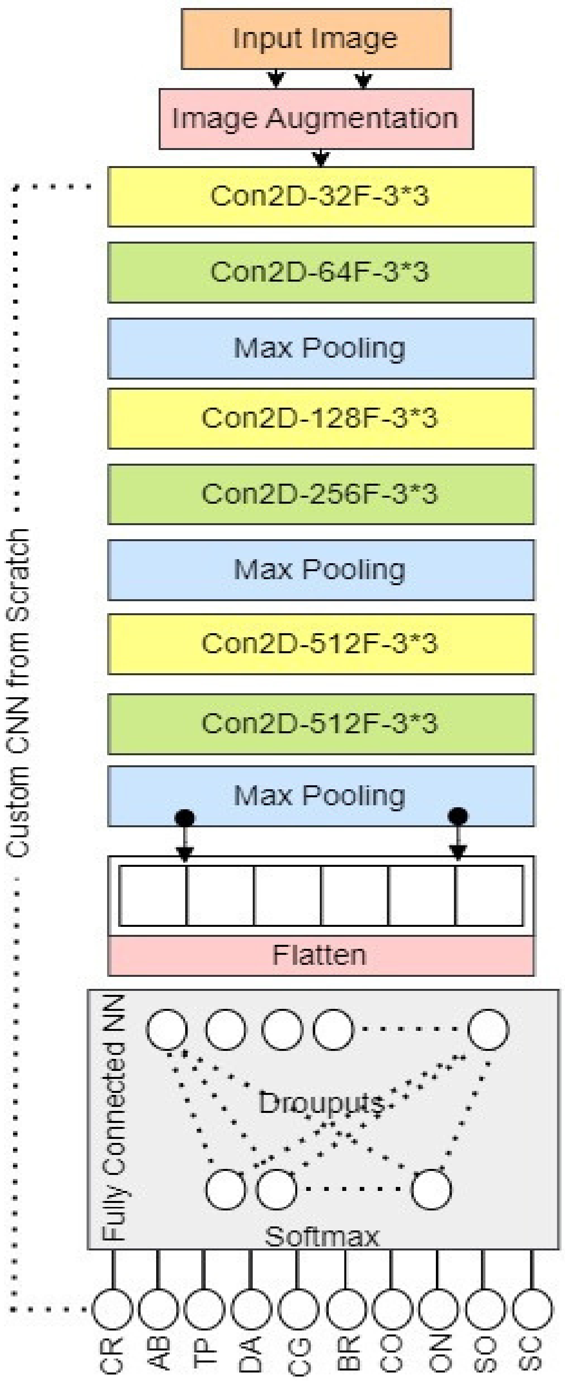

Fig.4.2:The architecture of Model-2:Customized CNN with Image Augmentation

Model-3: Transfer Learning with VGGNet

The third model makes use of VGGNet architecture—more specifically, VGG16—for transfer learning. Transfer learning is the process of optimizing a pre-trained model for a given classification job by employing it on a sizable dataset. The crop and weed dataset was used to refine the VGG16 model, which is renowned for its depth and effectiveness in picture classification tasks. This approach leverages the rich feature representations learned by VGG16, providing a strong baseline for comparison with other models.

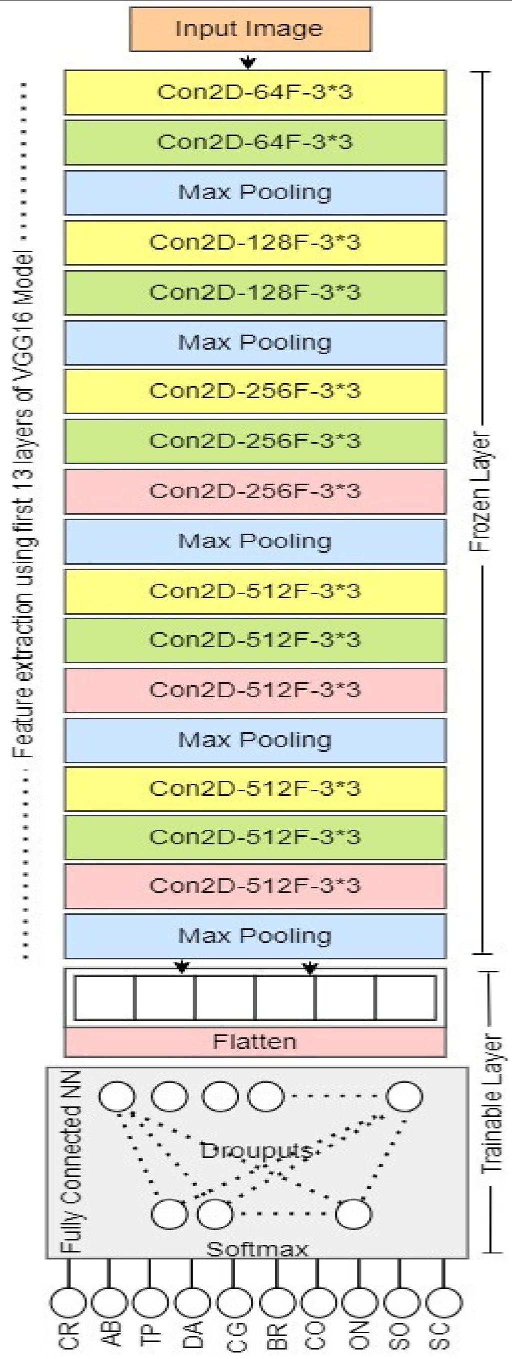

Fig. 4.3 : Fine-Tuned and Customized Architecture of Model-3

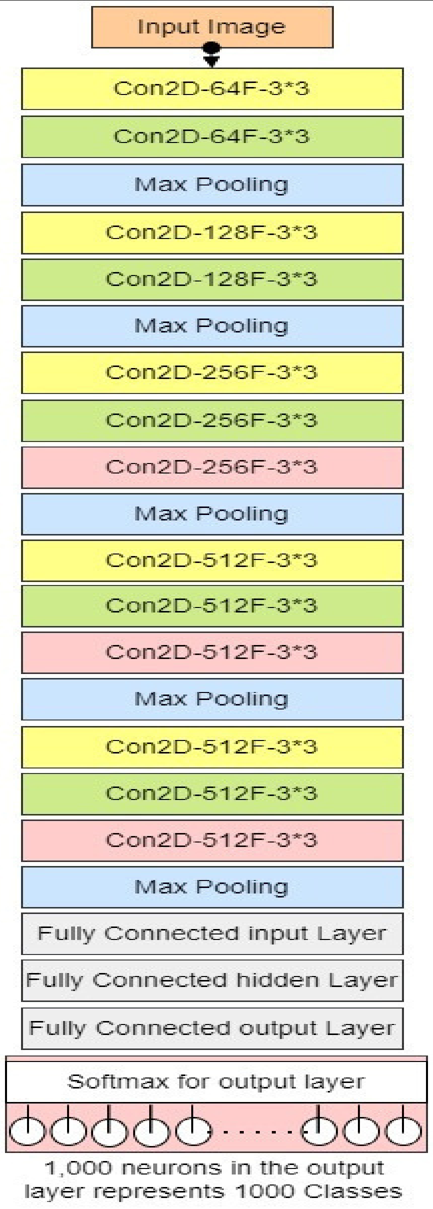

Fig. 4.4 : The architecture of actual VGGNet Model

Model-4: Transfer Learning with ResNet50

The fourth model, which uses the ResNet50 architecture, also makes use of transfer learning, just like Model 3. ResNet50, also known as the Residual Network with 50 layers, is well known for its ability to use residual connections to train extremely deep networks. By reducing the impact of the vanishing gradient issue, these connections enhance the network's capacity for learning. Using the crop and weed dataset, the ResNet50 model was adjusted to enhance classification performance.

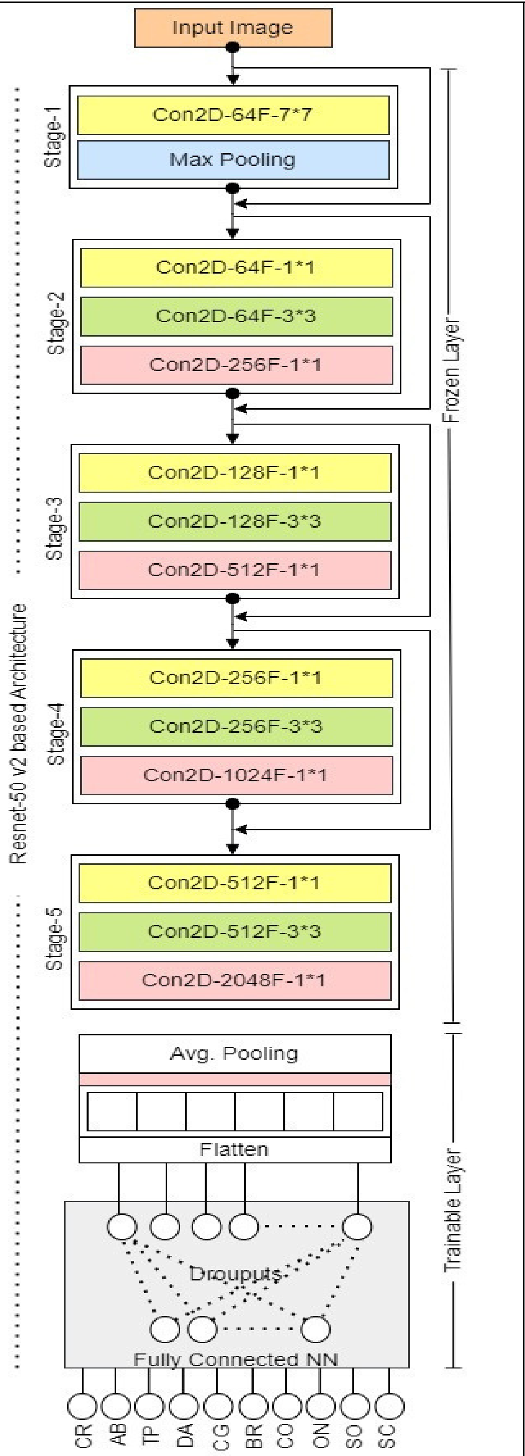

Fig. 4.5 : Fine-Tuned and Customized Architecture of Model-4:

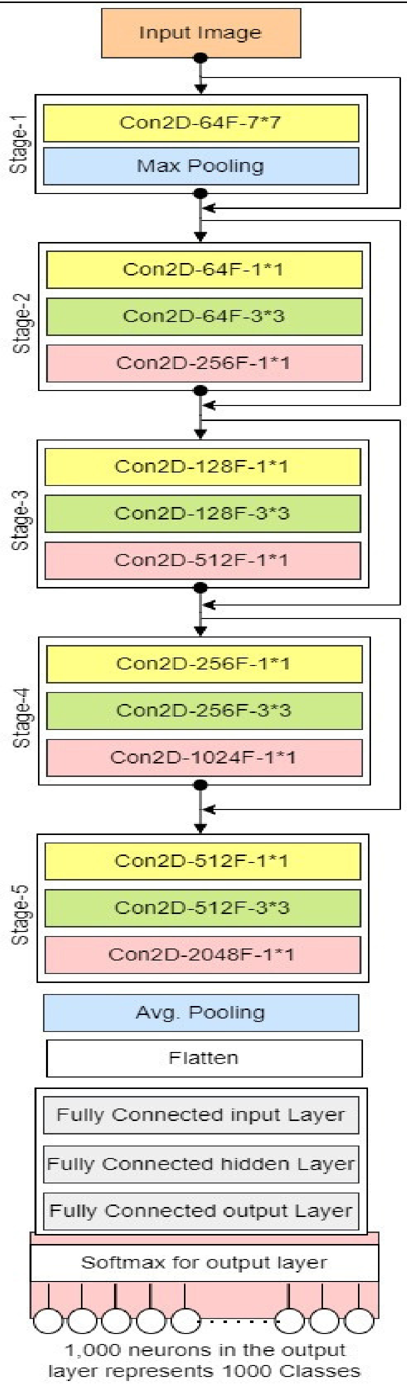

Fig. 4.6 : The architecture of the actual ResNet50 Model

4.2.2 Estimating Crop and Weed Density: Transfer Learning with YOLOv8

We used the YOLOv8 (You Only Look Once) object identification technique to assess the population density of weeds and crops. This process begins with segmenting high-resolution images of agricultural fields into smaller sections, known as quadrats. Each quadrat is then analyzed using the YOLOv8 model to detect and count the occurrences of weeds and crops. By leveraging the pre-trained YOLOv8 model and applying transfer learning techniques, we adapted the model to our specific dataset, ensuring high accuracy in detection. This technique is ideal for real-time applications in large-scale farming operations because it makes precise and effective monitoring of plant populations across vast agricultural areas possible.

The use of YOLOv8 in this context offers significant advantages in speed and accuracy, facilitating rapid analysis and decision-making. The data gathered from the detection process is aggregated to estimate the population density of weeds and crops across the entire field. In addition to increasing crop optimization and weed control effectiveness, this strategy enhances precision agricultural methods by offering in-depth understanding of plant population dynamics. The integration of advanced object detection algorithms like YOLOv8 enhances the overall capability to monitor and manage agricultural fields effectively.

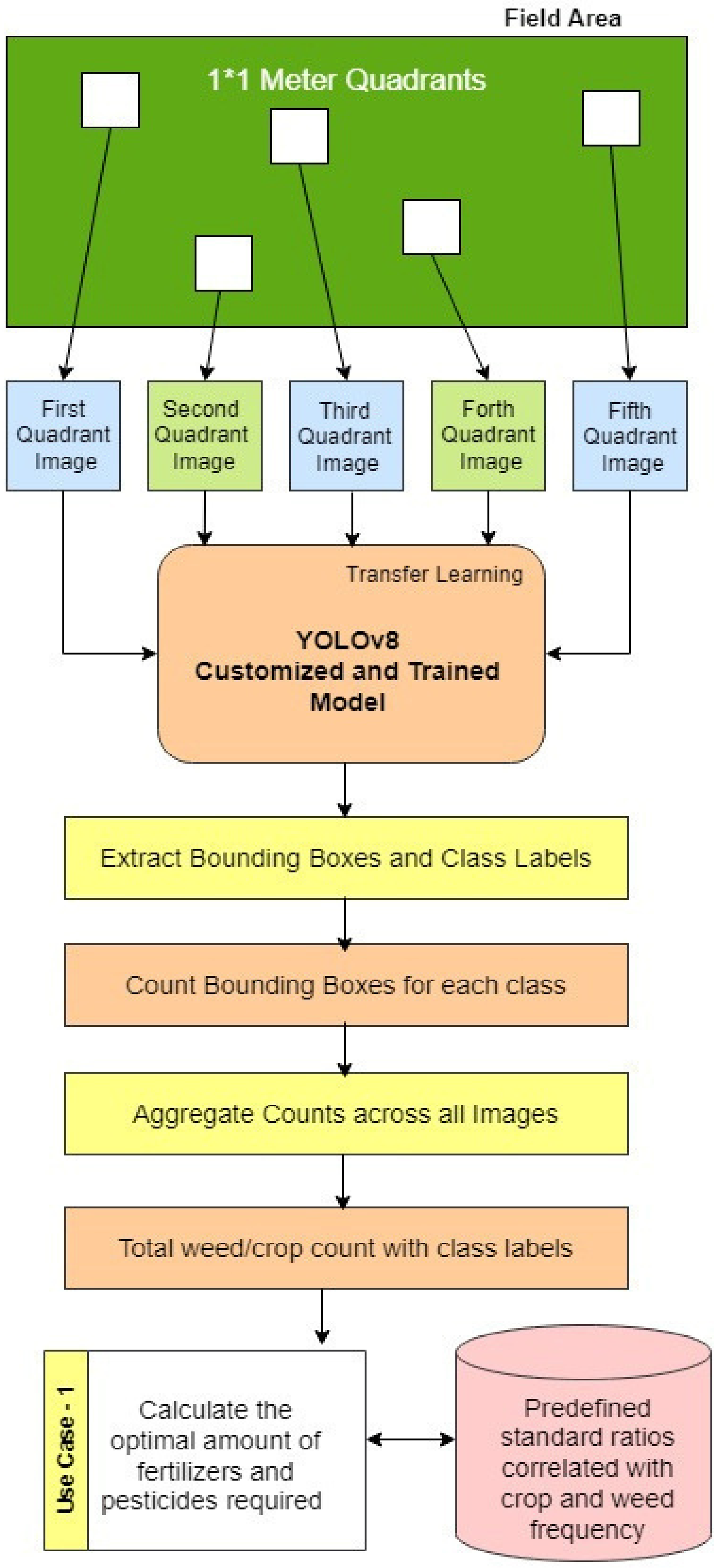
Field Area
1*1 Meter Quadrants
First Quadrant Image
Second Quadrant Image
Third Quadrant Image
Forth Quadrant Image
Fifth Quadrant Image
Transfer Learning
YOLOv8 Customized and Trained Model
Extract Bounding Boxes and Class Labels
Count Bounding Boxes for each class
Aggregate Counts across all Images
Total weed/crop count with class labels
Use Case - 1
Calculate the optimal amount of fertilizers and pesticides required
Predefined standard ratios correlated with crop and weed frequency

Fig. 4.7 : Process Flow for Population Density Analysis of Weeds and Crops Using YOLOv8

4.3 Experimental Setup

This section offers a thorough rundown of the experimental configuration that was utilized to train, adjust, and assess the CNN models for classifying plant species and weeds. The experiments were conducted on Google Colab, leveraging the capabilities of the Keras and TensorFlow libraries for deep learning tasks. The experimental pipeline encompassed data preprocessing, model training, evaluation metrics computation, and model deployment using Gradio for real-time predictions.

4.3.1 Implementation Environment

Google Colab, a cloud-based platform that offers free access to GPU resources for deep learning model training, was used for the trials. Using Google Colab made it possible to effectively employ GPU acceleration, which sped up training and made it easier to experiment with various model architectures and hyperparameters.

Libraries and Frameworks

The implementation of the research project relied on a combination of popular libraries and frameworks in the Python ecosystem, primarily focused on deep learning, image processing, and data visualization. Below is a detailed overview of the libraries and frameworks utilized:

NumPy: A set of mathematical methods to manipulate massive, multi-dimensional arrays and matrices, as well as support for these arrays, make NumPy an essential package for scientific computing with Python.

Pandas: Data structures and procedures for working with numerical tables and time series data are provided by this robust Python data manipulation and analysis package.

Matplotlib is a feature-rich Python visualization toolkit that can be used to create static, interactive, and animated graphics. It offers a MATLAB-like interface for customizing and plotting different kinds of charts and graphs.

OpenCV (cv2): Known for its broad support for image processing tasks like feature extraction, object detection, and picture segmentation, OpenCV is a popular computer vision library. Real-time computer vision applications make extensive use of it.

TensorFlow: Created by Google Brain, TensorFlow is an open-source deep learning framework for creating and refining neural network models. It provides an adaptable architecture that may be used to implement machine learning models on CPUs, GPUs, and TPUs, among other platforms.

Keras: Written in Python, Keras is an API for high-level neural networks that may be used with TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). It offers a simple-to-use interface with less coding complexity for creating and refining deep learning models.

Seaborn: Seaborn is a Matplotlib-based statistical data visualization library that offers an aesthetically beautiful and educational depiction of intricate datasets. It makes the process of creating aesthetically pleasing graphs for statistical modeling and exploratory data analysis easier.

scikit-learn: Supporting both supervised and unsupervised learning techniques, scikit-learn is a flexible Python machine learning library. It offers tools for selecting, evaluating, and preparing data as well as for calculating performance indicators.

Gradio: Gradio is a straightforward yet effective framework for building user interfaces for machine learning models. Through web-based interfaces, it makes it easier for ML models to be deployed and interacted with, allowing users to input data, see predictions, and investigate model behavior in real-time.

4.3.2 Hardware and Software Specifications

For the experimental setup, the following hardware and software specifications were utilized:

Hardware:

GPU: The experiments were conducted using a GPU-accelerated environment provided by Google Colab. Specifically, a Tesla P100 GPU was allocated for training the deep learning models. The GPU acceleration significantly reduced the training time compared to CPU-only execution, enabling faster experimentation and model iteration.

Software:

Operating System: The experiments were performed on Google Colab, which provides a cloud-based Jupyter notebook environment.

Deep Learning Frameworks: The primary deep learning frameworks used for model development and training were Keras and TensorFlow. The high-level API for creating and training neural networks was given by Keras, and the backend for effective computation and optimization was supplied by TensorFlow.

Python Libraries: Various Python libraries were employed for data preprocessing, model evaluation, and deployment. These include NumPy, Pandas, Matplotlib, and Gradio.

Development Environment: The experiments were conducted using Python 3.x within the Google Colab environment, leveraging its integration with Jupyter notebooks for interactive development and experimentation.

4.3.3 Training Parameters

The following training parameters were applied in the experimental setup:

1. Batch Size: During training, the batch size is the quantity of training examples processed in a single iteration. The CNN models were trained with a batch size of 32 to balance memory usage and computational effectiveness.
2. Number of Epochs: One full run through the training dataset is represented by one epoch. In order to avoid overfitting, the models were trained for a predetermined amount of epochs and then stopped early. For every model, there were between fifty and one hundred epochs.
3. Optimizer: During model training, gradient descent optimization was performed using the Adam optimizer. Adam is an adaptive learning rate optimization algorithm that produces better performance and faster convergence by computing individual adaptive learning rates for various parameters.
4. Learning Rate: During optimization, the step size is decided by the learning rate at each iteration. The CNN models were trained at a learning rate of 0.001, which ensured effective and steady convergence without generating oscillation or divergence.

4.3.4 Evaluation Metrics

The performance of the trained CNN models was evaluated using the evaluation metrics listed below:

1. Confusion Matrix: When comparing the model's predictions to the ground truth labels, a confusion matrix offers a thorough synopsis. True positives, false positives, true negatives, and false negatives can all be shown, providing insights into the classification accuracy and error kinds of the model.
    a. A tabular representation of the actual vs. expected classes generated by a classification model is called a confusion matrix.
    b. True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) are its four constituent parts.
    c. The confusion matrix for an N-class multi-class classification issue is a N×N matrix.
    d. In the matrix, each column (i, j) denotes the number of class I occurrences that were incorrectly projected to be class J.
    e. Misclassifications are represented by off-diagonal matrix elements, whereas correctly categorized instances are represented by diagonal matrix elements (from top-left to bottom-right).
2. Accuracy: The percentage of correctly identified cases relative to the total number of instances is called accuracy. Although it offers a broad evaluation of the model's overall effectiveness, it might not be enough for datasets that are unbalanced.
    a. The ratio of successfully predicted instances to all instances is how accuracy is measured. Accuracy assesses the overall correctness of the model's predictions across all classes.
    b. Although accuracy by itself might not be appropriate for unbalanced datasets, it does offer a broad picture of the performance of the model. Accuracy formula

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}{TP + TN}$$

3. Precision: The percentage of true positive predictions among all the model's positive predictions is measured by precision. It shows how well the model can prevent false positive mistakes.

a. Precision gauges how well the model predicts favorable outcomes.
b. The ratio of real positives to the total of false positives and true positives is used to compute it.
c. Precision is the percentage of cases that are actually positive that are anticipated to be positive.
d. Precision calculation formula

$$Precision = \frac{\mathrm{TP}}{TP + FP}$$

4. Recall (Sensitivity): The percentage of accurate positive predictions among all real positive examples in the dataset is measured by recall. It evaluates how well the model captures all positive events, leaving none out.
   a. Recall quantifies the model's accuracy in identifying positive examples.
   b. The ratio of true positives to the total of false negatives and true positives is used to compute it.
   c. The percentage of real positive cases that the model correctly identified is known as recall.Formula to calculate Recall

$$Recall = \frac{\mathrm{TP}}{TP + FN}$$

5. F1-Score: This balanced indicator of the model's performance across precision and recall is the harmonic mean of precision and recall. It is especially helpful for datasets that are unbalanced.
   a. The F1-Score, which is the harmonic mean of recall and accuracy, is helpful in situations when there is an unequal class distribution because it strikes a balance between memory and precision.
   b. The F1-Score has a maximum value of 1 and a minimum value of 0.
   c. Formula to calculate F1-Score

$$F1 - Score = \frac{2 \times \mathrm{Precision} \times \mathrm{Recall}}{Precision + Recall}$$

6. Support: The number of real instances of each class in the dataset is represented by support. It aids in interpreting the importance of evaluation measures and offers insights into the distribution of classes.

In classification reports, support is frequently shown in addition to precision, recall, and F1-score to give a comprehensive view of the model's performance across many classes. It assists in determining whether the assessment metrics are impacted by dataset imbalances or are based on an adequate number of cases for each class.

7. AUC ROC Plot: A graphical depiction of the model's performance across various threshold values is the Area Under the Receiver Operating Characteristic (ROC) curve. It offers a comprehensive evaluation of the model's class discrimination capabilities, particularly in binary classification problems.

7.1 The Curve of Receiver Operating Characteristic (ROC):

- For various threshold values, the relationship between the true positive rate (Sensitivity) and the false positive rate (1 - Specificity) is shown graphically by the ROC curve.
- Sensitivity is a metric that quantifies the percentage of actual positive cases among all true positive predictions that the model accurately recognized.
- Specificity quantifies the percentage of real negative occurrences across all actual negative cases that the model properly identified as true negative predictions.
- Sensitivity is plotted against 1-Specificity at different threshold values, often between 0 and 1, to form the ROC curve.

7.2 Interpretation of ROC Curve:

- A perfect classification model would have a ROC curve that hugged the upper-left corner of the plot, achieving high sensitivity and specificity at the same time.
- The ROC curve of a random guessing model is represented by a diagonal line that runs from bottom-left to top-right, and for all threshold values, the true positive rate equals the false positive rate. It is believed that a model that falls within this range has no discriminating power.The model performs better the farther the

ROC curve is from the random guessing line and the closer it is to the top-left corner.

7.3 Area Under the Curve (AUC):

- An binary classification model's total performance is measured by the AUC. It stands for the ROC curve's area under the curve.
- AUC values vary from 0 to 1, with a perfect classifier being indicated by an AUC of 1.
- AUC = 0.5 indicates that a model has no ability to discriminate—the same as guesswork.
- AUC less than 0.5 suggests that the model is not as good as random guessing.

The stronger the model's capacity to discriminate between positive and negative instances, the higher its AUC score.