Jurnal ELTIKOM :
Jurnal Teknik Elektro, Teknologi Informasi dan Komputer

Vol. 8, No. 2, December 2024, page. 179-191
ISSN 2598-3245 (Print), ISSN 2598-3288 (Online)
DOI: http://doi.org/10.31961/eltikom.v8i2.1340
Available online at http://eltikom.poliban.ac.id

DISEASE DETECTION IN TROPICAL TOMATO LEAVES VIA MACHINE LEARNING MODELS

Benjamin Kommey[1*], Elvis Tamakloe[1], Daniel Opoku[2], Tibilla Crispin[1], Jeffrey Danquah[1]

[1)] Department of Computer Engineering, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana

[2)] Department of Electrical Engineering, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana

E-mail: bkommey.coe@knust.edu.gh, tamakloe.elvis@gmail.com, dopoku.coe@knust.edu.gh, crispintibilla@gmail.com, danquahjeffrey1@gmail.com

Received: 16 October 2024 – Revised: 31 October 2024 – Accepted: 1 November 2024

ABSTRACT

This study addresses the significant threat of tomato diseases to production in Ghana, which has led to substantial yield and quality losses, adversely affecting the livelihoods of local farmers and the availability of this essential dietary staple. Traditional disease identification methods are time-consuming and rely on subjective visual inspections, hindering early detection and control. This study develops a machine learning model capable of accurately identifying tomato plant diseases through image processing. The methodology involves processing a dataset of tomato plant images displaying healthy and diseased symptoms. The proposed model employs the YOLOv5 architecture and is deployed on a mobile platform for accessible disease identification. The model achieved a validation mAP@.5 of 0.715, demonstrating strong performance during live, on-site testing. This system provides a swift, accurate, and automated solution for detecting tomato diseases, supporting the sustainability of tomato production in Ghana.

Keywords*: CNN, disease detection, image processing, leaf, machine learning, tomato*.

I. INTRODUCTION

TOMATOES are a staple ingredient in the daily diet of Ghanaians, with approximately 90% of the 300,000 metric tons produced annually consumed locally. Tomatoes account for about 38% of total vegetable expenditure in the country [1]. However, plant diseases significantly threaten tomato production, causing severe losses in yield and quality. These diseases, caused by various fungi or water molds that infect leaves, stems, and fruits, spread rapidly under favorable environmental conditions such as warm and wet weather. Common tomato diseases include bacterial spots, leaf mosaic, leaf curls, septoria, blight, and fusarium.

A survey conducted in three districts within Ghana's forest and forest-savannah agro-ecological zones identified blight as the predominant fungal disease, with a mean incidence of 63.9% in the Asante Akim North District [2]. In contrast, fusarium was most prevalent in the Offinso North District. These diseases not only reduce the marketability and shelf life of tomatoes but also increase production costs due to the heavy reliance on fungicides. In some cases, tomato diseases have caused annual economic yield losses of up to 79%, underscoring their detrimental impact on Ghana's tomato production.

Early detection and diagnosis of tomato diseases are crucial for effective management and prevention of further damage. Conventional methods, which rely on expert visual inspections, are time-consuming and labor-intensive. This necessitates the development of fast, accurate, and automated disease identification methods using machine learning.

Advancements in image processing and machine learning have recently provided promising solutions for the early and precise detection of plant diseases, enabling timely intervention and mitigation. Among the crops affected by diseases, tomatoes hold particular economic importance in Ghana. The detection

of tomato leaf diseases has garnered considerable attention due to their profound impact on yield and quality.

II. RELATED WORKS

[3] proposed a lightweight CNN model for classifying tomato leaf blight, utilizing a transfer learning approach to evaluate the performance of pre-trained MobileNetV2, NASNetMobile, and EfficientNetV2B0 models. Among these, the EfficientNetV2B0 model achieved the highest training accuracy of 99.95%, followed by MobileNetV2 with 99.65%, and NASNetMobile with 97.98%. The final model size was 30.6 MB, making it suitable for deployment on mobile applications. This model was implemented on a React Native mobile application, enabling farmers to capture images of tomato leaves for disease prediction. [4] introduced a custom CNN model for identifying tomato leaf diseases, aiming to deploy the trained model on a web platform for accessible disease identification. The model, trained on a custom dataset over more than 50 epochs, achieved an accuracy of 94.17% across eight classes of tomato leaf diseases and one healthy class. The web platform allowed users to upload images of plant leaves, predict the disease, and receive suggested remedies. [5] investigated diseases affecting corn, potato, and tomato plants, including bacterial spots, septoria, and target rust on tomatoes. The study evaluated the performance of VGG16, VGG19, and a traditional CNN model using a training dataset of 25,272 images sourced from Kaggle. Results indicated that VGG19 performed best with an accuracy of 95%, while both VGG16 and the traditional CNN achieved 86%.

[6] modeled ResNet and Xception architectures to classify healthy leaves and detect early blight in tomato plants, aiming to train models suitable for mobile application development. The study utilized self-collected images to train the two models. YOLO object detectors, including its variants (YOLOv3, YOLOv3-tiny, and YOLOv3-SPP), were used as feature extractors to identify diseased regions of tomato leaves. After training, ResNet achieved an accuracy of 99.735%, while Xception achieved 99.952%. [7] employed a standard CNN model for detecting tomato leaf diseases, leveraging multiple layers to process the dataset more efficiently. The images were sourced from Kaggle's PlantVillage dataset and augmented. Seven layers, including Input, Convolutional, Pooling, Nonlinear, Dropout, Softmax, and Normalizing layers, were integrated to generate the final feature map. The model achieved an accuracy of 97.35%. [8] utilized a computer vision algorithm powered by artificial intelligence to detect tomato plant diseases. The model focused on three common diseases: Early Blight, Late Blight, and Septoria Leaf Spot. A Region Proposal Network (RPN) was used to segment the leaves, followed by the Chan-Vese (CV) algorithm to extract symptomatic features from the segmented images. The Kaggle PlantVillage dataset was used for training and testing. The final model, a CNN with nine layers, achieved an accuracy of 91.66%. [9] developed a custom CNN model to detect tomato leaf diseases across nine disease classes and one healthy class. The model was trained on Kaggle's PlantVillage dataset and utilized a sequential architecture comprising three convolutional layers, three max-pooling layers, a ReLU activation function, and a learning rate of 0.001. The final model achieved an accuracy of 91.2%.

[10] utilized logistic regression, support vector machine (SVM), and random forest algorithms for tomato leaf disease detection. The dataset used was Kaggle's PlantVillage dataset, and features were extracted using the Histogram of Oriented Gradients method. Among the models, SVM achieved the highest accuracy of 73%. [11] proposed a custom CNN model for detecting tomato leaf diseases, trained on Kaggle's PlantVillage dataset. The model was compared to a pre-trained InceptionV3 model, with both models trained for 10 epochs. The custom CNN model outperformed InceptionV3, achieving an accuracy of 98.2% compared to InceptionV3's 81.33%. [12] developed a custom CNN model to detect tomato leaf diseases using images of tomato leaves. The architecture included four convolutional layers, four max pooling layers, and fully connected layers. The model achieved an overall accuracy of 96.26%. [13] implemented a custom CNN model to classify three tomato leaf diseases, utilizing data augmentation techniques. The model was trained on a dataset of 4,400 leaf images from Kaggle. Additionally, a 5-fold cross-validation method was applied, resulting in an accuracy of 97.8%. [14] introduced a custom CNN model called TLNet (Tomato Leaf Net) for tomato leaf disease detection across eight classes. The model was trained using a Sequential architecture with five convolutional layers, seven batch normalization layers, five max pooling layers, three dense layers, and flatten, dropout, and Softmax activation layers. Training was conducted over 50 epochs with the Adam optimizer and a learning rate of 0.000001. The model achieved an accuracy of 98.77%.

[15] applied Deep Neural Networks (DNN) to detect tomato leaf diseases, using the Adaptive Histogram Equalization technique to address inadequate lighting in leaf images. The model achieved an accuracy of 98.29%. [16] implemented two distinct CNN models for identifying tomato illnesses. The first model, built using transfer learning on a pre-trained network, achieved an accuracy of 91.34%, while the nascent CNN model obtained an accuracy of 87%. [17] investigated the identification of tomato leaf diseases using the Inception V3 deep learning model and evaluated additional architectures, including VGG16, EfficientNet B3, and Inception V3. [18] utilized a CNN model based on the pre-trained VGG19 architecture to identify four tomato plant diseases: early blight, late blight, bacterial spot, and leaf mold. The model achieved an accuracy of 97.5% with a detection time of just 4 seconds.

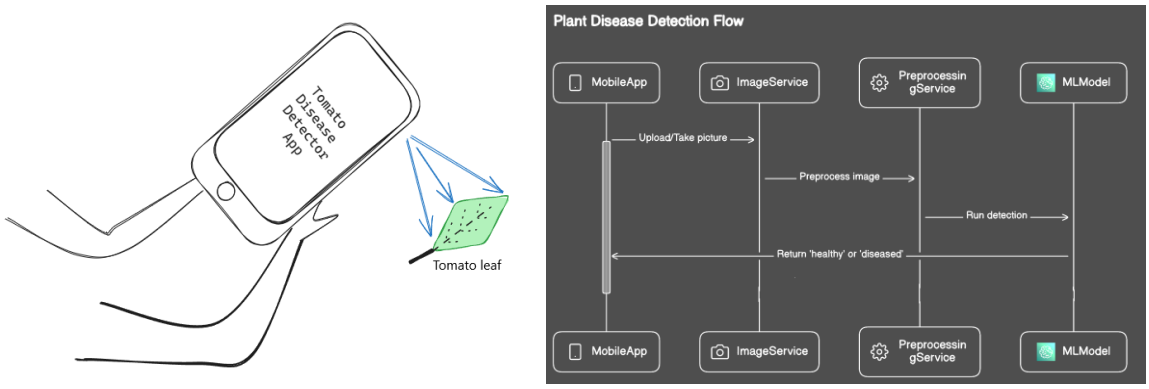
[19] utilized ResNet110 to address the limitation of limited plant leaf images, leveraging the PlantVillage dataset from Kaggle, which contains 16,012 images of tomato leaves across 10 classes. The performance of ResNet110 was compared to VGG16 and VGG19 models from other studies that used the same dataset. The proposed ResNet110 achieved the highest accuracy of 99.7%. [20] investigated transfer learning methods for detecting tomato leaf diseases, addressing the poor performance of nascent CNN models on small datasets. Pre-trained models, including InceptionV3, VGG16, and ResNet50, were employed, and their accuracy was compared to classic CNN models. The dataset contained 5,500 images distributed across 9 disease classes and one healthy class. [21] aimed to detect diseases in tomato leaf images using the ResNet50 architecture, with preprocessing steps involving Hough transform and morphological image processing. The proposed ResNet50 model achieved an accuracy of 97.6%, outperforming AlexNet (95%) and VGG16 (95.2%). [22] focused on identifying tomato leaf diseases using pre-trained CNN models. The study involved nine disease classes and one healthy class, testing nine pre-trained models and evaluating their performance. [23] deployed two CNN-based models, VGG16 and GoogLeNet, for tomato leaf disease classification using transfer learning. Both models were trained on Kaggle's PlantVillage dataset and achieved an accuracy of 99.23%. [24] explored six pre-trained models with fine-tuning techniques by modifying the network architecture layers. The models included MobileNetV1, InceptionV3, VGG16, DenseNet, ResNet, and AlexNet. The study did not apply data augmentation, focusing solely on optimizing accuracy. The final results showed accuracy levels of 99.75% for ResNet, 99.62% for MobileNetV1 and DenseNet, 98.74% for InceptionV3, 97.48% for AlexNet, and 96.73% for VGG16. [25] discussed various approaches to tomato disease detection, including the K-means algorithm, which groups similar pixels based on color or intensity.

[26] implemented a federated CNN model for tomato leaf disease detection across five severity levels, including one for healthy leaves and four representing different disease stages. The federated approach allowed multiple clients to train the model without sharing raw data. Instead, client data was used locally to train and update the model, which achieved an accuracy of 97% across all severity levels. [27] explored enhancing the pre-trained MobileNetV2 model by adding five additional layers and integrating it with the Caffe framework. The Caffe model was loaded for plant disease detection, followed by the modified MobileNetV2 for classification. Results showed that the proposed Caffe-MobileNetV2 methodology outperformed standalone CNN or MobileNetV2 models, achieving an accuracy of 99.28%. [28] integrated the VGG16 model with Faster R-CNN to develop a system for identifying and categorizing tomato leaf diseases. The combination of Faster R-CNN and VGG16 was used for training and testing the final model, and its performance was compared to alternative methods. The results showed varying accuracy scores: CNN with the EM algorithm achieved 94.53%, CNN with Random Forest achieved 95.46%, R-CNN achieved 92.2%, Fast R-CNN achieved 95.75%, and Faster R-CNN achieved 98%.

III. RESEARCH METHOD

A. Proposed System Architecture

Figure 1a illustrates the tomato leaf disease detection system. The system is designed for practical on-site application, where users can assess the health of tomato leaves using a smartphone. The mobile software application, equipped with a trained model, can be installed on any Android smartphone. Users can either upload an image of the leaf or capture one using the phone's camera. The application then predicts the health status of the leaf, classifying it as healthy or diseased. The primary goal of the mobile application is to enable faster detection by allowing multiple users to access the app on their



(a) (b)

Figure 1. System and mobile software application architecture

smartphones. These users are not required to have prior knowledge of tomato farming or plant health. By facilitating widespread usage, the system aims to reduce yield losses through early detection of diseases such as blight.

Figure 1b outlines the functionality of the mobile app. The *MobileApp* stage serves as the user interface, providing the frontend of the application. The *ImageService* stage manages the image acquisition process, including uploading an existing image or capturing one through the camera. The *PreprocessingService* stage converts the uploaded or captured image into the appropriate dimensions and format required by the trained model. The *MLModel* stage contains the trained machine learning model, which predicts the leaf's health status as either healthy or diseased. The result is then presented in a user-friendly format through the mobile app.

The application aims to streamline disease detection, offering a user-friendly interface with essential features. The camera button allows users to launch the camera app for on-site image capture, while the upload button enables users to select and upload images from their phone's gallery. Once an image is captured or uploaded, it is processed by the trained model to classify the health status of the leaf. The results are displayed to the user as either healthy or infected.

The system's object detection capability extends beyond classification by recognizing and localizing multiple objects within an image, which is particularly useful when a single tomato plant may exhibit multiple diseases. employs bounding boxes to precisely locate detected objects, combining classification and localization. In contrast, image classification assigns a single label to the entire image, identifying only the dominant class without localization. Although similar, object detection and image classification have distinct differences and use cases where each excels. Object detection identifies multiple objects within an image and provides bounding boxes that specify their positions. Its output includes multiple labels and bounding boxes, making it more complex due to the added localization task. The primary evaluation metrics for object detection are Intersection over Union (IoU) and Mean Average Precision (mAP). Image classification, on the other hand, predicts the class of a single object by analyzing patterns and features of the entire image. Its output is a single label representing the dominant class in the image. This approach is less complex and straightforward, with accuracy, precision, and recall being the primary evaluation metrics. Object detection was chosen as the preferred approach for this system because it enables the identification of multiple diseases in a single image, making it more practical for real-world applications.

Transfer learning is a technique that leverages a pre-trained neural network as a foundation for a new training task. A pre-trained model is a deep learning model trained on a large dataset with pre-computed weights. To implement transfer learning, the pre-trained model is fine-tuned on a smaller dataset, making it particularly effective when data is limited. Unlike traditional CNN methods, which involve training the model from scratch, transfer learning starts with pre-existing weights rather than random initialization. Traditional CNNs train the entire network on a specific task, which can produce good results but often requires larger datasets and significant computational resources. Transfer learning, by contrast, is more efficient for tasks with smaller datasets, such as this project, which used only 4,280 images and did not require extensive custom features. The pre-trained model utilized for this project was YOLOv5 by Ultralytics [31], renowned for its speed and accuracy in object detection tasks for images and videos. YOLOv5 features a 3-tier architecture comprising the Backbone, Neck, and Head, each serving specific

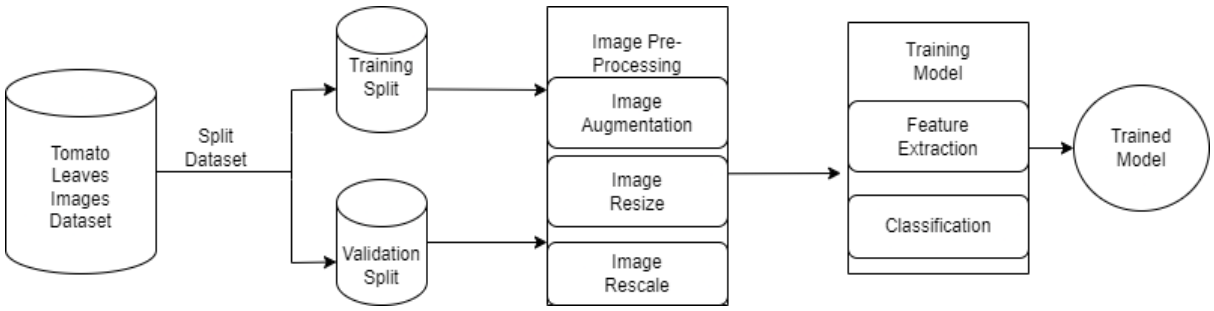


Figure 2. System architecture training

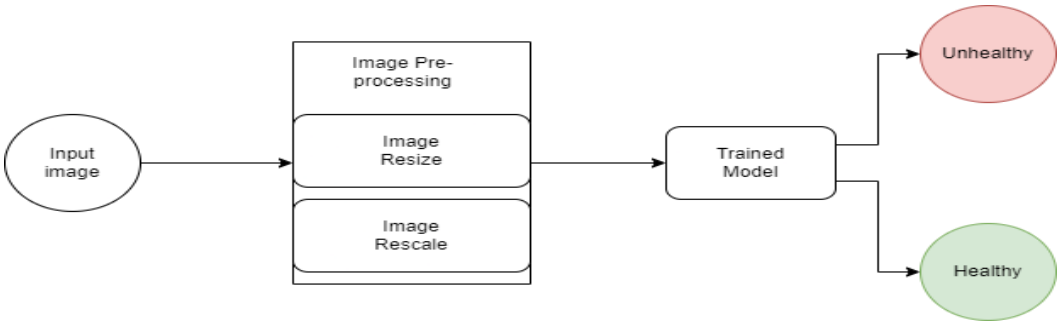


Figure 3. System architecture testing

roles to optimize performance. YOLOv5 is available in different versions—YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x—corresponding to small, medium, large, and extra-large sizes. While larger models offer higher accuracy, they also require longer training times. The YOLOv5 model was selected for this project due to its strong balance of accuracy and generalization capabilities, making it well-suited for the task.

B. System neural network architecture training and testing

The neural network architecture comprises two phases: the training phase and the testing phase. In the training phase, as illustrated in Figure 2, the dataset is divided into training and validation sets. The images are pre-processed through resizing, rescaling, and augmentation techniques (e.g., rotation, horizontal flipping, and zooming). The pre-processed training and validation images are then used to train and validate the model, with the training process involving feature extraction and classification.

In the testing phase, as illustrated in Figure 3, the trained model processes images captured or uploaded by users. These user-provided images are pre-processed by resizing and rescaling to meet the model's input requirements. The pre-processed image is then passed through the trained model, which classifies it as either healthy or diseased.

Other models could potentially serve a similar purpose for this project, such as VGG16, VGG19, and MobileNetV2. However, VGG16 and VGG19 were excluded due to their unsuitability for mobile deployment. While MobileNetV2 is better suited for mobile applications, it is a general-purpose model, unlike YOLOv5, which is specifically optimized for object detection. YOLOv5 offers superior accuracy, speed, and scalability, with multiple model sizes (e.g., YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x) to suit different use cases. For these reasons, YOLOv5 was chosen over MobileNetV2, as it better aligns with the project's objectives.

The training processor offers two options: CPU and GPU. In machine learning, GPUs are typically preferred due to their advantages over CPUs, including significantly shorter training times and, in some cases, better results. However, GPUs require more computational resources than CPUs. For this work, after multiple trials comparing CPU and GPU training, the GPU demonstrated superior performance with much shorter training times. The number of training cycles, referred to as epochs, is a critical parameter in model training. An epoch defines how many times the model processes a batch of image data. In this case, the initial target was 1,000 epochs. However, higher epoch counts can increase training

TABLE 1
DATA STRUCTURE FOR IMAGE TRAINING

Class	Number of instances
Bacterial spot	1251
Early blight	4938
Late blight	10869
Septoria	18165
Leaf curl	1371
Mosaic	603
Fusarium	2643

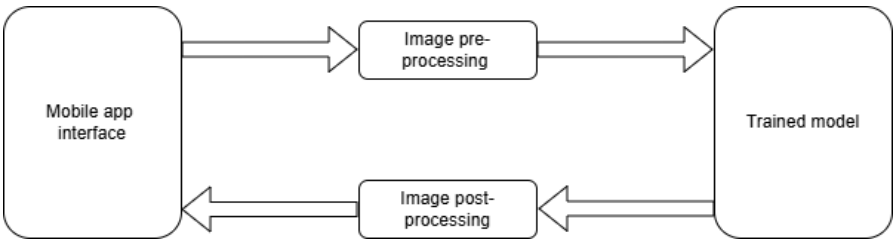


Figure 4. System prototype block diagram

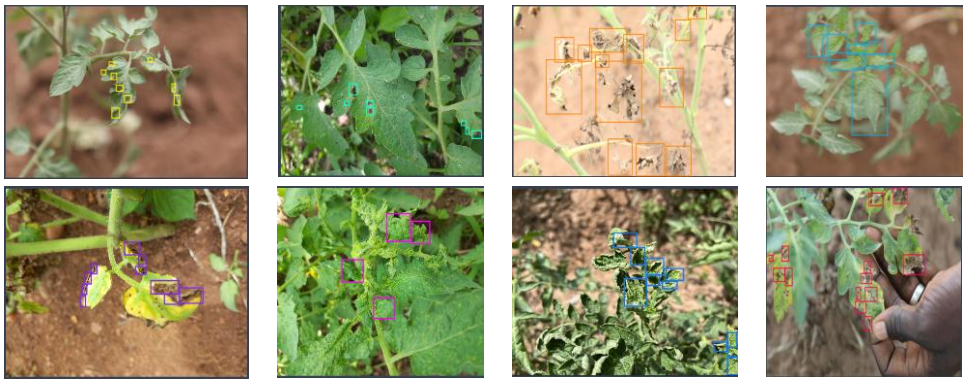


Figure 5. System training phase images

time and the risk of overfitting, whereas lower epoch counts reduce training time but may impact performance metrics such as accuracy, precision, and recall. To balance these factors, EarlyStopping was implemented, resulting in training stopping at 721 epochs. EarlyStopping prevents overfitting and reduces unnecessary training time while achieving optimal results.

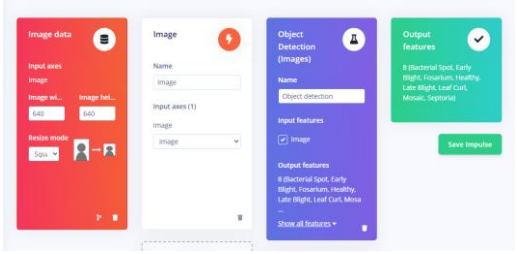
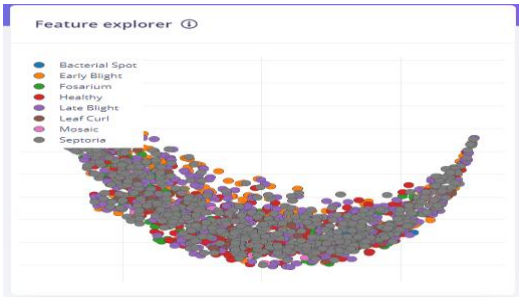
The model size also influences training outcomes, with different sizes (small, medium, large, and extra-large) yielding varying results and training durations. After extensive testing, the "Large" model was selected as it provided the best balance between accuracy and training time. Furthermore, the final trained model's size was optimized for mobile deployment. Batch size, another important parameter, determines the number of image samples processed before recalculating weights and moving to the next batch. Batch size impacts both training time and model performance. After multiple tests, a batch size of 16 was found to be optimal for this work. The validation set size, representing the percentage of training data allocated for validation, was set at 20%, a commonly recommended value. Adjustments to the validation set size did not significantly affect training time or model performance.

The neural network architecture parameters are crucial for configuring and selecting the neural network for training. The input layer, which is the first stage of the training computation, holds the features generated during the feature generation stage of the platform. These features are passed to the chosen model, YOLOv5, for training. YOLOv5 processes these features through its various layers to produce output. The output stage classifies the data into one or more classes or determines if no class is present.

C. Proposed system prototype

The system prototype, as illustrated in Figure 4, comprises components including the mobile software application, the trained model, and the tools and technologies used to develop the project. The block diagram highlights two primary components: the mobile software application interface and the trained model, along with tools for image pre-processing and post-processing that work together to achieve the desired results.

TABLE 2
EDGE IMPULSE IMAGE TRAINING

Edge impulse	Image training	Output image
Impulse creation	Process is a custom step peculiar to the Edge Impulse platform. This stage handles the pre-processing of the images. It resizes the images to stated dimensions and prepares the images for training The Edge Impulse platform describes an impulse as; takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data. The impulse has four blocks which perform different functions for the overall pre-processing of the image. The Image data block allows you to specify the dimensions for resizing all the input images. This is specified within the 'Image width' and 'Image height' fields. The 'Resize mode' field specifies how the images should be resized (Squash, Fit longest axis, Fit shortest axis). Different settings have different advantages, disadvantages and use cases. After multiple trials, the settings in the image above produced the best results. The processing block (Image block) performs and simple task of specifying the type of data. The block in the above image specifies the data as images this prevents any errors during pre-processing. The learning block (Object Detection' block) specifies the type of training (object detection or traditional CNN) for the model. Objected detection is selected over the traditional CNN method for reasons stated in a previous chapter. The last block, 'Output features' show the different classes of the input image data. The block makes clear the classes to train on and makes clear the results expected by the user	
Feature creation	The feature generation stage plays a very important role for object detection by extracting relevant features from the raw image data. The features serve as input for machine learning models. The process allows the use of pre-defined extraction methods (RGB or Grayscale) or create custom methods using processing blocks. Feature generation groups the image data to show how similar the images are to one another of the same label. Feature generation describes the structure of the data and can be a predictor of the training results. Below is an image of the feature map of the training images The feature map shows that the images do not group very well. This may be due to the small size of the training data set. With a larger and better-labelled dataset, the images may generate a better feature map.	
Object detection	This is one of the most important stages in the project. This stage involves passing the pre-processed images as input to the pre-trained model (YOLOv5) for training. Different parameters produce different results for post training. The main task at this stage to find the best parameters that produce the best results. The Edge Impulse platform provides a lot of assistance at this stage by recommending settings and parameters for the training of the model. The recommended parameters do not always produce the best results but serve as a starting point for the training of the model. Below shows the parameters used for training the model	

The dataset used for the project is Crop Disease Ghana [29] from Kaggle, containing images and labels for three crops: tomato, corn, and pepper. The tomato classes applied for model training include healthy, early blight, late blight, fusarium, septoria, bacterial spot, leaf curl, and mosaic. The images were annotated using the PASCAL VOC format, which supports object detection. Each image has an accompanying .xml file containing the leaf's coordinates, bounding box dimensions, and corresponding labels. Given this dataset structure, an object detection approach was adopted.

The model was trained using Edge Impulse [30], a leading-edge AI platform designed to simplify building, deploying, and scaling embedded ML applications. The platform offers a range of tools, including data annotation tools, pre-trained models, and charting tools, which streamline machine learning projects. Edge Impulse was chosen for its user-friendly interface and comprehensive toolkit. Before training, the dataset was cleaned to ensure accuracy. Some images were removed, bounding boxes were redrawn, and incorrect labels were updated or eliminated to guarantee that the model was trained on properly labeled data, resulting in more accurate predictions.

The initial step in any machine learning training process involves verifying the data to ensure quality and accuracy. In this project, the data consisted of images with their corresponding bounding box labels. The cleaning process ensured that the model was trained on correctly labeled images to produce accurate results. Additionally, data augmentation was applied to artificially expand the training dataset by generating modified versions of existing images, which improves model performance and generalization. Augmentation techniques included horizontal flips, 90-degree rotations, random blurs, random saturation, and random brightness adjustments. After cleaning and augmentation, a total of 4,280 images were prepared. These images were automatically split by the Edge Impulse platform into 70% for training (3,741 images), 20% for validation (361 images), and 10% for testing (178 images).

The training process followed the cleaning and augmentation steps and involved image pre-processing, running training cycles, and validating results by assessing performance metrics. The Edge Impulse platform facilitated the training and validation process by recommending optimal parameters. The training process on Edge Impulse included the creation of impulses, feature generation, and object detection. Detailed explanation of the steps is provided in Table 2, and corresponding images are depicted in Figure 5.

The mobile software application platform serves as the interface between the user and the trained model, providing information about the disease detected by the model. The operational process of the mobile application is outlined below.

1) The user either captures an image of the tomato leaf using the in-built camera on-site or uploads an image from the phone's gallery.
2) The image undergoes pre-processing to remove potential interferences.
3) The inference output is processed.
4) The processed output, including bounding box coordinates for the detected sections, is overlaid on the image.

The mobile app's frontend and backend were developed using the Flutter framework [32]. Flutter was selected for its suitability in mobile application development and its packages that support machine learning functionalities. The inferencing stage utilizes the 'tflite_flutter' package [35], developed by TensorFlow specifically for tflite models. This version of the trained model is optimized for devices with low computational resources, such as mobile devices. During the mobile image processing stage, the input image is converted to the required format for the model. The output results are formatted by class confidence and class score, the best bounding boxes are selected, and the bounding boxes and labels are adjusted to fit the original image.

To convert the input image to the required format, it is essential to understand the image size specifications of the model. These specifications can be determined using tools such as Netron [36]. Since the model was trained with dimensions of 640x640, the input image must be resized accordingly. Inference cannot proceed without resizing the image to the correct dimensions. Additionally, the image must be rescaled and transformed into a suitable matrix format.

During inference, the model generates multiple results, which include bounding boxes and class labels. For the tflite model, it produces 25,200 results, each containing bounding box coordinates and class confidence scores. The class confidence score indicates whether a detected bounding box corresponds to any of the output classes. To ensure accuracy, only bounding boxes with a class confidence score above a defined threshold are selected. A threshold of 0.8 was chosen to include only detections with 80% or greater certainty.

The output data also contains class scores for all output classes. These scores indicate the class to which a bounding box belongs. A class score threshold of 0.5 is applied, and bounding boxes are discarded if all class scores fall below this threshold. The highest-class score is then used to assign a label to the bounding box.

To adapt the bounding box data to the original image, mathematical computations are required. The output includes normalized x, y, width, and height coordinates (ranging from 0 to 1) for each bounding box. These normalized coordinates are converted to actual coordinates on the original image. Additional computations are performed to ensure the bounding boxes fit appropriately within the mobile device display.

The mobile software application interface serves as the frontend of the application, facilitating interactions between the user and the backend. It enables users to capture images using the camera or select images from the gallery. Additionally, it displays the bounding boxes generated after detection. The red

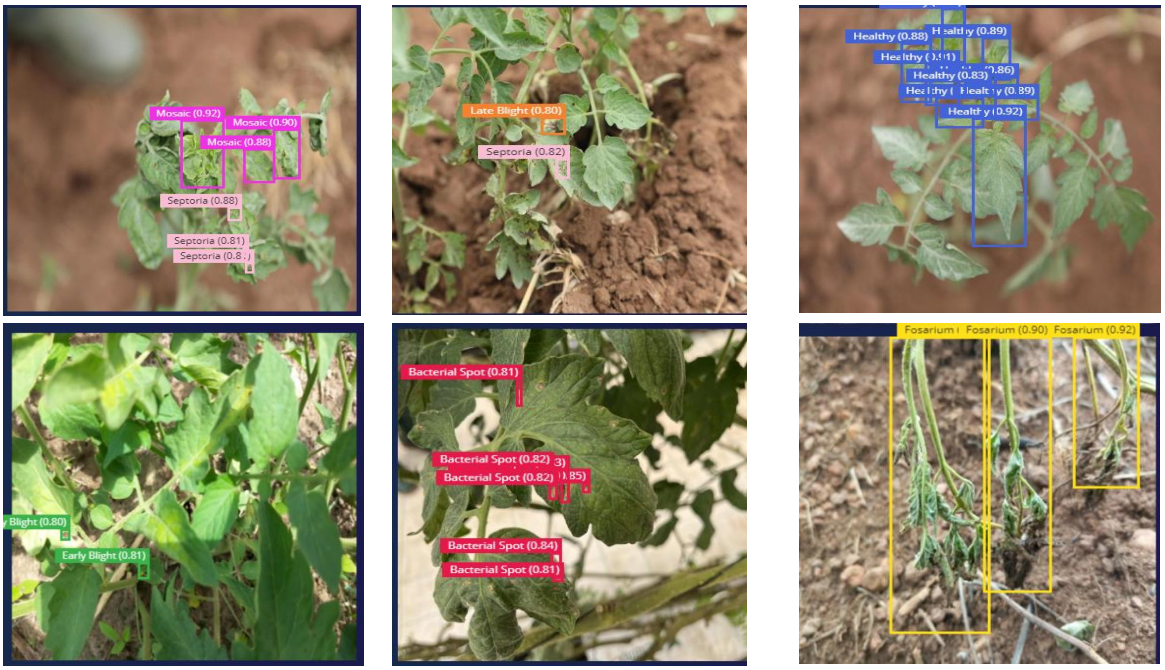


Figure 6. Screenshots from the training phase

Figure 7. Tomato farm used for testing

circle highlights the name of the software application. The blue arrow at the center indicates the ‘NO PHOTO’ image, signaling that no image has been uploaded for detection. The brown arrow points to the camera button, which allows the user to capture images of the tomato leaf using the inbuilt camera. The black arrow identifies the gallery button, which enables users to upload images from the device gallery. Finally, the red arrow points to the detect button, which initiates inference on the uploaded image.

When an image is captured or uploaded, the 'NO PHOTO' placeholder is replaced with the tomato leaf image. The user can then press the detect button to perform inference on the uploaded image.

IV. Results and Discussion

The tomato disease model was successfully trained and tested on 539 test images, which included samples from each disease class to ensure unbiased test results. Figure 6 displays some images from the testing phase. The results indicate that most detections were accurate. After exporting and integrating the model into the mobile app, the application was tested using images from the test split of the dataset. Additional testing was conducted on a small tomato farm with approximately 25 tomato plants, all at the same growth stage (pre-fruit). The farm is shown in Figure 7.

Using the camera feature of the mobile app, tomato leaves were tested on-site for diseases. Testing was performed during the afternoon and early evening to assess the impact of time of day. Two devices, a Samsung Galaxy S9 and a Samsung A02, were used to evaluate performance across different hardware. Three scenarios were considered: the effect of time of day, device type, and camera angles.

TABLE 3
INFERENCE TIMES FOR DIFFERENT DEVICES

Crop (#)	Galaxy S9 interface (ms)	Galaxy A02 interface (ms)
1	4521	7048
2	4041	7443
3	3966	5963
4	4323	6542
5	4111	7010
Average	4192	6801

Figure 8. Camera angle testing

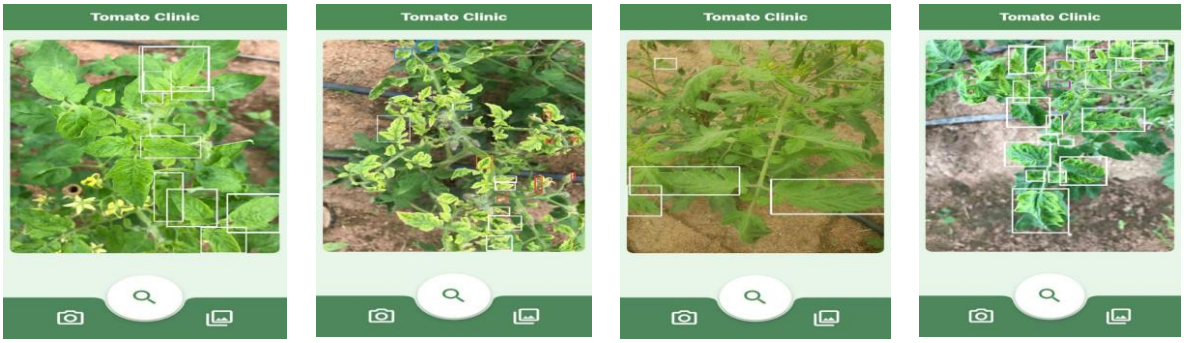


Figure 9. On-site testing results

For the time-of-day scenario, one set of crops was tested around noon, and another set around 5 p.m., when lighting conditions were moderate (not too dark or too bright). The results showed consistent model performance in both scenarios, with no significant impact from the time of day. This consistency is likely due to the dataset containing augmented images, which make the model robust under various lighting conditions, except in very dark settings.

For the device comparison, the same five crops were tested using both the Galaxy S9 and the A02 to assess the impact of device type on results. The detected outcomes remained consistent across devices, with differences observed only in inference time. The Galaxy S9 had an average inference time of 4,192 ms per image, while the A02 recorded an average of 6,801 ms per image.

Observing inference results across both devices revealed that camera angles significantly influenced the outcomes. Images where the leaves were directly facing the camera produced good results. Conversely, in images where the leaves were not directly facing the camera, the number of detected results decreased, and in some cases, no detections were made. Figure 8 illustrates examples of testing with different camera angles.

After testing multiple images under various conditions, the results showed that most crops were classified as healthy, with only one instance of leaf mosaic detected. Figure 9 provides screenshots of detections conducted on-site.

TABLE 4
SUMMARY OF TRAINING RESULTS

Class	Images	Instances	Precision	Recall	mAP5@.5	mAP@.5:.95
all	749	9697	0.746	0.711	0.715	0.407
Bacterial spot	749	239	0.683	0.615	0.627	0.294
Early blight	749	963	0.776	0.75	0.778	0.412
fusarium	749	491	0.762	0.766	0.753	0.438
healthy	749	1667	0.791	0.792	0.823	0.568
Late blight	749	2185	0.711	0.681	0.681	0.343
Leaf curl	749	278	0.772	0.731	0.674	0.443
mosaic	749	121	0.82	0.744	0.802	0.526
septoria	749	3753	0.677	0.62	0.613	0.268

Following parameter configuration, the training and validation processes began with the definition of hyperparameters. The code snippet below specifies the hyperparameters used during training: *hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0*.

Among these, the most critical hyperparameters are the learning rate, momentum, and weight decay. The learning rate determines the step size during optimization and controls how quickly the model minimizes the loss function. In this case, a learning rate of 0.01 was selected as it ensures effective convergence. Momentum accelerates gradient descent in the correct direction while reducing oscillations, promoting consistent convergence; this parameter was set at 0.937. Weight decay mitigates overfitting by adding a penalty to the loss function, encouraging smaller model weights; its value was set at 0.0005. These values were key to achieving successful training outcomes.

The hyperparameter values were optimized through multiple tests and trials to achieve the best results. The optimization algorithm employed was SGD (Stochastic Gradient Descent), which updates the model parameters during training. After setting the hyperparameters, the next step involved freezing certain layers of the model. Freezing layers is a technique used to control how weights are updated during training. The weights of frozen layers remain unchanged throughout subsequent training iterations. This method is particularly beneficial for small datasets paired with large models, as it reduces training time without significantly affecting training outcomes. In this process, 183 layers were frozen. Below is a code snippet showing the freezing of the first 10 layers:

freezing model.0.conv.weight
freezing model.0.bn.weight
freezing model.0.bn.bias
freezing model.1.conv.weight
freezing model.1.bn.weight
freezing model.1.bn.bias
freezing model.2.cv1.conv.weight
freezing model.2.cv1.bn.weight
freezing model.2.cv1.bn.bias
freezing model.2.cv2.conv.weight

Following the freezing of layers, the training and validation cycles commenced. During each cycle, metrics such as precision, mean Average Precision (mAP), and loss were calculated. Weights were continually readjusted to optimize the model's performance. At the end of the training process, the best-performing version of the model was selected as the final model.

The training stage lasted 14.5 hours and yielded a training precision score of 80.2%. Other metrics assessed during training included accuracy, recall, and mAP. Precision, a key metric, measures the proportion of correctly predicted positives. It is defined by (1).

$$P = \frac{Truepositves}{Truepositives + falsenegatives} \quad (1)$$

Accuracy is the ratio of correct predictions to the total number of predictions, providing an overall measure of the model's performance across all classes. Recall, which reflects the model's ability to identify all relevant instances, is a combination of precision and accuracy. Mean Average Precision (mAP) compares ground-truth bounding boxes to detected boxes and provides a performance score. Average Precision (AP) represents the number of bounding boxes that meet an Intersection over Union (IoU) threshold. The IoU measures how well a predicted bounding box aligns with the ground truth. At an IoU threshold of 0.5, a detection is considered correct if the overlap between the predicted box and the ground truth box is at least 50%. The overall mAP score is the mean of the AP scores for all classes.

Table 4 summarizes the training results, showing that the model achieved a mAP@0.5 score of 0.715. This indicates that the trained model performed well in detecting tomato diseases during validation. It is expected that performance could improve with a larger dataset.

To enable deployment in a mobile application, the final model was converted to a format suitable for on-device inference. The tflite format is recommended for such applications due to its low resource requirements. The conversion from .pt to .lite format was performed automatically using the Edge Impulse platform. The resulting .lite model is 88 MB in size, making it appropriate for integration into a mobile app.

V. CONCLUSION

This project successfully developed and tested machine learning models capable of detecting tomato diseases from images of tomato leaves. The YOLOv5 architecture used achieved a validation mAP@0.5 of 0.715 and demonstrated strong performance during live on-site testing. Results indicate that the model performs reliably across different devices. Despite a few limitations, the findings have significant implications for disease detection. The developed mobile software application enables users, regardless of their expertise in tomato diseases, to assist in disease identification, potentially reducing yield losses. Future work should focus on extending the models to detect a broader range of plant diseases, such as those affecting corn and pepper. Additionally, a secure database system should be integrated into the application to store detection images for future reference and learning. Expanding the models to support live video feeds for real-time plant disease detection would further enhance their utility.

REFERENCES

[1] Goodman AMC, "Ghana's Tomato Processing Industry: An Attractive Investment Option in 2016." https://goodmanamc.blogspot.com/2015/12/ghanas-tomato-processing-industry.html. (Accessed Oct. 13, 2024).

[2] B.A. Opoku, C. Kwoseh, E. Gyasi, and E. Moses, "Incidence and severity of major fungal diseases on tomato in three districts within the forest and forest-savannah agro-ecological zones of Ghana." *Ghana Journal of Agricultural Science*, vol. 56, no. 2, pp: 46-60. 2021. doi: 10.4314/gjas.v56i2.5.

[3] O. Chougule, D. Katheria, K. Jain, and S. Shinde, "Tomato Blight Classification Using Transfer Learning and Fine Tuning." *2022 2nd Asian Conference on Innovation in Technology (ASAINCON)*, pp: 1-11. 2022. doi: 10.1109/ASIANCON55314.2022.9908820.

[4] K.S. Rekha, H.D. Phaneendra, B.C.S. Gandha, H. Rohan, B.N.S. Niranjan, and C. Badrinat, "Disease Detection in Tomato Plants Using CNN." *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, pp: 1-6. 2022. doi: 10.1109/GCAT55367.2022.9972186.

[5] M. Al-Shalout, M. Elleuch, and A. Douik, "Detection Plant Diseases Using Deep Learning Algorithms." *2023 International Conference on Cyberworlds (CW)*, pp: 341-344. 2023. doi: 10.1109/CW58918.2023.00060.

[6] A.S. Chakravarthy, and S. Raman, "Early Blight Identification in Tomato Leaves using Deep Learning." *2020 International Conference on Contemporary Computing and Applications (IC3A)*, pp: 154-158. 2020. doi: 10.1109/IC3A48958.2020.233288.

[7] A. Gupta, M.S. Kumar, M.R. Kumar, and D.H. Kumar, "Deep Learning Technique Used for Tomato and Potato Plant Leaf Disease Classification and Detection." *2023 International Conference on Smart Systems for applications in Electrical Sciences (ICSSES)*, pp: 1-6. 2023. doi: 10.1109/ICSSES58299.2023.10199327.

[8] Sheenam and A. Kumar, "Advanced CNN-Based Approach for Accurate Tomato Plant Diseases Recognition." *2023 3rd International Conference on Intelligent Technologies (CONIT)*, pp: 1-5. 2023. doi: 10.1109/CONIT59222.2023.10205755.

[9] V.S.K. Chaitanya, D. Rakesh, S. Dash, B.K. Sahoo, S. Padhy, and M. Nayak, "Tomato Leaf Disease Detection using Neural Networks." *2022 International Conference on Machine Learning, Computer Systems and Security (MLCSS)*, pp: 53-58. 2022. doi: 10.1109/MLCSS57186.2022.00018.

[10] R. Mohanty, P. Wankhede, D. Singh, and P. Vakhare, "Tomato Plant Leaves Disease Detection using Machine Learning." *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pp: 544-549. 2022. doi: 10.1109/ICAAIC53929.2022.9793302.

[11] R.K. Kodali, and P. Gudala, "Tomato Plant Leaf Disease Detection using CNN." *2021 IEEE 9th Region 10 Humanitarian Technology Conference (R10-HTC)*, pp: 1-5. 2021. doi: 10.1109/R10-HTC53172.2021.9641655.

[12] P. Pradhan, and B. Kumar, "Tomato leaf disease detection and classification based on deep convolutional neural networks." *2021 First International Conference on Advances in Computing and Future Communication Technologies (ICACFCT)*, pp: 13-17. 2021. doi: 10.1109/ICACFCT53978.2021.9837379.

[13] A.I. Yoren, and S. Suyanto, "Tomato Plant Disease Identification through Leaf Image using Convolutional Neural Network." *2021 9th International Conference on Information and Communication Technology (ICoICT)*, pp: 320-325. 2021. doi: 10.1109/ICoICT52021.2021.9527425.

[14] M.A.A. Mamun, D.Z. Karim, S.N. Pinku, and T.A. Bushra, "TLNet: A Deep CNN model for Prediction of tomato Leaf Diseases." *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, pp: 1-6. 2020. doi: 10.1109/ICCIT51783.2020.9392664.

[15] T.A. Kumar, S. Oviya, S.A. Ajagbe, C. Ananth, O. Aiyeniko, and M.O. Adigun, "A Novel Deep Neural Network-Based Prediction Model for Identifying Diseases in Tomato Leaves." *2023 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pp: 1-6. 2023. doi: 10.1109/ICECET58911.2023.10389207.

[16] Y. Mori, B. Limbasia, R.K. Gupta, and S.K. Bharti, "AI Based Potato and Tomato Leaf Disease Detection Using CNN and Pre-Trained Model." *2023 7th International Conference on Computing, Communication, Control And Automation (ICCUBEA)*, pp: 1-6. 2023. doi: 10.1109/ICCUBEA58933.2023.10392028.

[17] S. Samala, N. Bhavith, R. Bang, D. Kondal Rao, C.R. Prasad, and S. Yalabaka, "Disease Identification in Tomato Leaves Using Inception V3 Convolutional Neural Networks." *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, pp: 865-870. 2023. doi: 10.1109/ICOEI56765.2023.10125758.

[18] H.P. Ananya, S.K. Magadum, S. Swathi, and A.S. Prasad, "Real Time Tomato Plant Leaf Disease Detection Using Convolutional Neural Network." *2023 International Conference on Recent Trends in Electronics and Communication (ICRTEC)*, pp: 1-6. 2023. doi: 10.1109/ICRTEC56977.2023.10111895.

[19] P. Harinadha, and C.K. Mohan, "Tomato Plant Leaf Disease Detection Using Transfer Learning-based ResNet110." *2023 International Conference on Data Science and Network Security (ICDSNS)*, pp: 1-8. 2023. doi: 10.1109/ICDSNS58469.2023.10244907.

[20] H. Singh, U. Tewari, and S. Ushasukhanya, "Tomato Crop Disease Classification using Convolutional Neural Network and Transfer Learning." *2023 International Conference on Networking and Communications (ICNWC)*, pp: 1-6. 2023. doi: 10.1109/ICNWC57852.2023.10127284.

[21] N. Kumari, V. Kumar, N.K. Pandey, A.K. Mishra, and D. Kholiya, "Tomato Leaf Disease Detection and Identification using Machine Learning." *2023 3rd International Conference on Innovative Sustainable Computational Technologies (CISCT)*, pp: 1-5. 2023. doi: 10.1109/CISCT57197.2023.10351321.

[22] P. Pradhan, and B. Kumar, "Automatic Detection of Tomato Diseases using Fine-tuned pre-trained Deep Learning Models." *2022 3rd International Conference for Emerging Technology (INCET)*, pp: 1-5. 2022. doi: 10.1109/INCET54531.2022.9825376.

[23] H. Kibriya, R. Rafique, W. Ahmad, and S.M. Adnan, "Tomato Leaf Disease Detection Using Convolution Neural Network." *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)*, pp: 346-351. 2021. doi: 10.1109/IBCAST51254.2021.9393311.

[24] L.A. Chamli Deshan, M.K. Hans Thisanke, and D. Herath, "Transfer Learning for Accurate and Efficient Tomato Plant Disease Classification Using Leaf Images." *2021 IEEE 16th International Conference on Industrial and Information Systems (ICIIS)*, pp: 168-173. 2021. doi: 10.1109/ICIIS53135.2021.9660681.

[25] S. Shrivastav, V. Jindal, R. Eswarawaka, and D. Ray, "A Comprehensive Review and Discussion of Segmentation based Tomato Plant Disease Detection Approaches." *2023 International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE)*, pp: 1-7. 2023. doi: 10.1109/RMKMATE59243.2023.10369911.

[26] S. Mehta, V. Kukreja, and R. Yadav, "A Federated Learning CNN Approach for Tomato Leaf Disease with Severity Analysis." *2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, pp: 309-314. 2023. doi: 10.1109/ICAISS58487.2023.10250571.

[27] B.A. Kumar, M. Bansal, and R. Sharma, "Caffe-MobileNetV2 based Tomato Leaf Disease Detection." *2023 3rd International conference on Artificial Intelligence and Signal Processing (AISP)*, pp: 1-6. 2023. doi: 10.1109/AISP57993.2023.10134929.

[28] G. Priyadharshini, and D.R. Judie Dolly, "Comparative Investigations on Tomato Leaf Disease Detection and Classification Using CNN, R-CNN, Fast R-CNN, and Faster R-CNN." *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp: 1540-1545. doi: 10.1109/ICACCS57279.2023.10112860.

[29] Responsible AI Lab, "Crop Disease (Ghana): Afrocentric (African) Crop Dataset." https://www.kaggle.com/datasets/responsibleailab/crop-disease-ghana/data. (Accessed Oct. 13, 2024).

[30] Edge Impulse, "AI for Any Edge Device." https://edgeimpulse.com/. (Accessed Oct. 13, 2024).

[31] G. Jocher et.al., "ultralytics/yolov5:v5.0-YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integration." https://github.com/ultralytics/yolov5. (Accessed Oct. 13, 2024).