INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Processing Plant Diseases Using Transformer Model

Hong-Zheng Marcus Lye [a], Kok-Why Ng [a,*]

[a] *Faculty of Computing and Informatics, Multimedia University, Selangor, 63100, Malaysia*

*Corresponding author: *kwng@mmu.edu.my*

***Abstract*—Agriculture faces challenges in achieving high-yield production while minimizing the use of chemicals. The excessive use of chemicals in agriculture poses many problems. Accurate disease diagnosis is crucial for effective plant disease detection and treatment. Automatic identification of plant diseases using computer vision techniques offers new and efficient approaches compared to traditional methods. Transformers, a type of deep learning model, have shown great promise in computer vision, but as the technology is still new, many vision transformer models struggle to identify diseases by examining the entire leaf. This paper aims to utilize the vision transformer model in analyzing and identifying common diseases that hinder the growth and development of plants through the plant leave images. Besides, it aims to improve the model's stability by focusing more on the entire leaf than individual parts and generalizing better results on leaves not in the image center. Added features such as Shift Patch Tokenization, Locality Self Attention, and Positional Encoding help focus on the whole leaf. The final test accuracy obtained is 89.58%, with relatively slight variances in precision, accuracy, and F1 score across classes, as well as satisfactory model robustness towards changes in leaf orientation and position within the image. The model's effectiveness shows the vision transformer's potential for automated plant disease diagnosis, which can help farmers take timely measures to prevent losses and ensure food security.**

***Keywords*—Vision transformer; deep learning; plant disease; computer vision.**

Manuscript received 7 Dec. 2022; revised 12 Jul. 2023; accepted 25 Sep. 2023. Date of publication 31 Dec. 2023.

I. Introduction

Modern agriculture faces challenges like excessive chemicals to be used for high yield and pest control, which necessitates prompt and accurate disease diagnosis [1], [2]. Due to the industry's growth, traditional methods of diagnosing diseases based on farming expertise or professional advice are no longer sufficient. A novel approach involves automatic plant disease identification using machine vision techniques, where disease symptoms like changes in leaf shape, color, and texture provide crucial data [3], [4]. Differences in these characteristics within images can help classify and determine the disease afflicting the plant. Recently, transformer models in deep learning have shown remarkable results, surpassing convolutional neural networks (CNN) in many complex tasks, including computer vision. [5], [6]. They are more accurate, finely tuned, robust, and better at handling imperfections [7]. This paper uses the transformer model to analyze plant diseases, identifying common diseases that stress plants via leaf images [8].

Studies by [9], [10] and [11] employed visual transformers (ViT) to weed and crop classification using Unmanned Aerial Vehicles (UAV) images. Their application used the self-attention mechanism on the transformer model. It showed promising performance, particularly in situations with fewer training samples. The Swin Transformer by [12] demonstrated reduced computational complexity and increased accuracy but required large-scale datasets. Hybrid models, combining CNNs with ViTs, proposed by [6] and [13], improved the model speed and reduced the complexity while maintaining the model's accuracy.

Convolutional Neural Networks (CNN) models are the most widely used due to their suitability for real-time detection, albeit with lower accuracy than other models. Innovative CNN studies like [14] and [15] proposed different techniques to improve the performance of CNNs, including optimizing parameter count and computational efficiency and incorporating other methodologies like transformers. Another study by [16] focused on individual lesion examination for leaf disease identification, boosting accuracy but potentially losing some contextual information from the original images. Meanwhile, [17] and [18] used CNN to address false positives and class imbalances in tomato disease identification.

You Only Look Once (YOLOv4) algorithm proposed by [19] and the multi-granularity feature extraction model based on ViT by [20] represent unique approaches that do not fit neatly into the previous categories, offering innovations in feature learning ability, detection procedure accuracy, and disease classification accuracy [21], [22]. In summary, ongoing research in plant disease identification using computer vision is dynamic, with different methodologies having their unique strengths and weaknesses, often influenced by the trade-offs between accuracy, complexity, and speed.

II. Material and Method

A. Data Preprocessing

Before training begins, the Keras Sequential model is used to preprocess and augment the training dataset. The purpose of this is to increase the diversity and size of the training dataset as well as to standardize the input data for the model. The Normalization layer scales input values to the range [-1, 1], with a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean and dividing by the standard deviation of the dataset. The train dataset's mean and standard deviation are calculated using Keras's adapt () function. Normalization helps the model converge faster during training and prevents training optimization from getting stuck due to uneven distribution.

The Resizing layer scales the images to have a fixed size.[23]. This is done because deep learning models like Vision Transformers require input size to be consistent. [24], [25]. The size chosen is typically based on what works best for the model architecture, computational efficiency, and the task at hand. Here, the resized image size was set to 72x72 for quicker computation.

The following three layers introduce slight edits to the input images into the training dataset. This helps to increase the diversity of the data and reduces the model's dependency on the orientation of the objects in the images, making the model more robust to orientation changes in unseen data. The RandomFlip layer randomly flips the input images horizontally, the RandomRotation layer applies a small random rotation to the input images, whereas the RandomZoom layer applies a small zooming-in or zooming-out effect to the input images.

B. Model Construction and Evaluation

The PlantVillage dataset was chosen from Kaggle to demonstrate the implementation of the proposed method. The PlantVillage dataset consisted of Tomato leaf images (16018 images), Potato leaf images (2152 images), and Pepper leaf images (2475 images). The Tomato dataset was chosen for having the most images, though the number of images was reduced to 12239 for a faster training time. In Figure 1, data preprocessing was performed before training began. The training dataset was standardized via normalization and resizing. Then, it was augmented by randomly flipping, rotating, and zooming in or out on the images to promote dataset diversity and to increase model robustness to changes in leaf orientation.

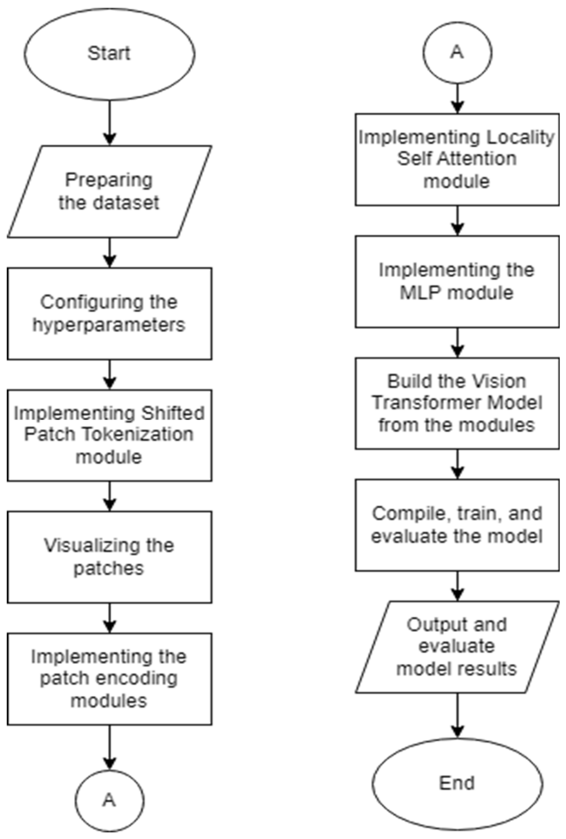
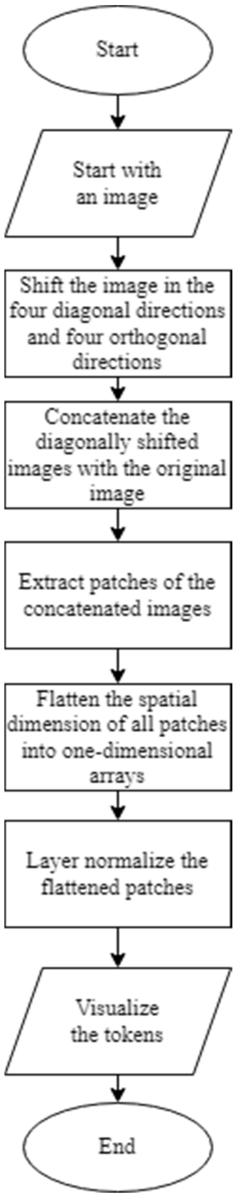

Start
Preparing the dataset
Configuring the hyperparameters
Implementing Shifted Patch Tokenization module
Visualizing the patches
Implementing the patch encoding modules
A
A
Implementing Locality Self Attention module
Implementing the MLP module
Build the Vision Transformer Model from the modules
Compile, train, and evaluate the model
Output and evaluate model results
End

Fig. 1 Flowchart of Vision Transformer Model

Firstly, a set of labeled leaf images is received from the Tomato dataset and then prepared by preprocessing the images to make them more accessible for the model, such as resizing the images to a smaller resolution. After configuring the model's hyperparameters, Shift Patch Tokenization will be implemented (this process will be further explained later), and then the patches will be visualized to ensure the process was implemented successfully.

When it is completed, the patch encoding layer is employed to add positional information to the patches, while the locality self-attention module (the process would be explained later) is applied to add attention weights to local maximums, which helps the model learn the inter-token relations within the patches. Next, the Multi-Layer Perceptron (MLP) neural network, which is the basis for the model's machine learning process, is applied [26]. Finally, all the components are combined to form the Vision Transformer model. After compiling the code, it proceeds to train the model and evaluate its performance for its accuracy and loss across the training process. The results of the model are shown then.

In Figure 2, the Shift Patch Tokenization starts with an image already preprocessed to fit the model's image size requirements. Firstly, the image is shifted in the four diagonal directions as well as the four orthogonal directions (top left, top right, bottom left, bottom right, left, right, top, bottom), leaving empty space where the image was shifted from, before concatenating the four shifted images with the original image into arrays. Then, small image patches are extracted from the array of concatenated images before the spatial dimensions of the extracted patches are flattened into one-dimensional arrays for easier processing during model training, producing the tokens of the image. Finally, layer normalization is performed on the tokens to translate the range of the token values to a range between 0 and 1. This aims to help the model reach convergence faster during training and help the model generalize better on unseen images. The tokens are then visualized to ensure the process was performed without issue.

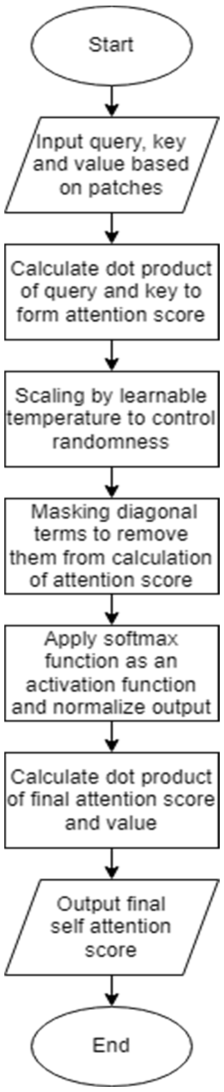


Start
Start with an image
Shift the image in the four diagonal directions and four orthogonal directions
Concatenate the diagonally shifted images with the original image
Extract patches of the concatenated images
Flatten the spatial dimension of all patches into one-dimensional arrays
Layer normalize the flattened patches
Visualize the tokens
End

Fig. 2 Flowchart of Shift Patch Tokenization

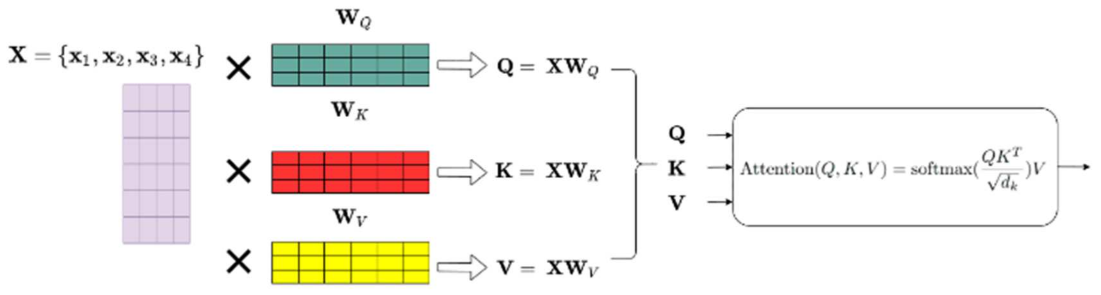
In Figure 3, the Locality Self Attention (LSA) module is a modified version of the regular self-attention module that is typically used in transformers. First, the module obtains the queries, keys, and values from the input tokens, which are the tokens of the patches that were extracted earlier during the patch encoding process. To explain what queries, keys, and values are: in general, for attention modules, a query represents the token of data currently being focused on during the calculation process, while keys represent all the tokens of data, and values represent the information held within the tokens of data.

In Figure 4, each query is multiplied by each key to obtain an attention score consisting of an array of values representing relationships between tokens (a.k.a. inter-token relations). Multiplication is performed via dot product, which is the multiplication of two vector matrices to obtain a final product of a single number. Then, the attention score is scaled by hyperparameters known as temperature for controlling the randomness of the predictions and for sharper differences in attention score values, which helps in finding inter-token relations. The temperature can be adjusted by backpropagation during the learning process. However, when using self-attention, the query, key, and value come from one input source, which can cause the attention score to focus on intra-token relations (relationships between the values within the same token) rather than inter-token relations. To prevent this from affecting the attention score, a mask that multiplies zero on diagonal terms is used, thus removing them from consideration in calculating the attention score, as a diagonal term is created when a token is multiplied by itself.



Start
Input query, key and value based on patches
Calculate dot product of query and key to form attention score
Scaling by learnable temperature to control randomness
Masking diagonal terms to remove them from calculation of attention score
Apply softmax function as an activation function and normalize output
Calculate dot product of final attention score and value
Output final self attention score
End

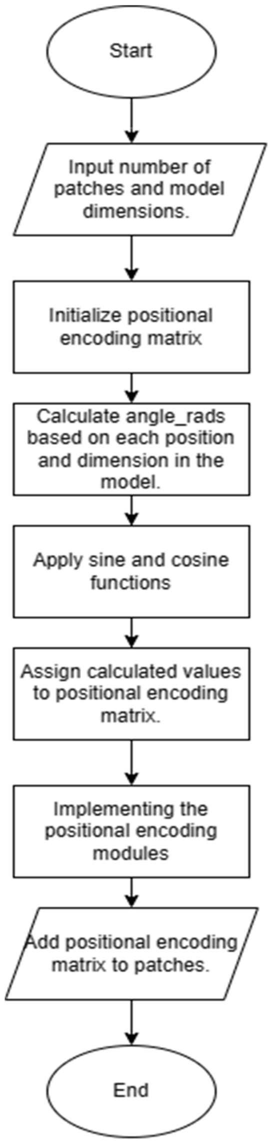
Fig. 3 Flowchart of Locality Self Attention Module



$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
$\mathbf{W}_Q$
$\mathbf{W}_K$
$\mathbf{W}_V$
$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$
$\mathbf{K} = \mathbf{X}\mathbf{W}_K$
$\mathbf{V} = \mathbf{X}\mathbf{W}_V$
Q
K
V
$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Fig. 4 Description for Query, Key and Value in Calculated in Attention Score

After that, a SoftMax function is applied as an activation function to normalize the values within the attention score into probabilities ranging from 0 to 1. This final attention score is then multiplied with the input values via dot product to obtain the output of the self-attention module, which is then used to help the model learn the inter-token relations. The PositionalEncoding class starts by receiving the model dimensions (d_model) and the maximum sequence length (maximum_position_encoding) as input, as stated in Figure 5.



Start
Input number of patches and model dimensions.
Initialize positional encoding matrix
Calculate angle_rads based on each position and dimension in the model.
Apply sine and cosine functions
Assign calculated values to positional encoding matrix.
Implementing the positional encoding modules
Add positional encoding matrix to patches.
End

Fig. 5 Flowchart of Positional Encoding Module

It initializes a matrix of shape (1, maximum_position_encoding, d_model) with zeros, which will store the positional encoding information. The class calculates the angle (i / 10000^(2*d/d_model)) for each position (i) in the sequence and each dimension (d) in the model and stores these angles in a matrix called angle_rads. The sine function is applied to the even indices (2i) of angle_rads, and the cosine function is applied to the odd indices (2i+1) of angle_rads. The resulting matrix contains the positional encoding values, which are then assigned to the positional encoding matrix initialized earlier. This matrix is now the pos_encoding attribute of the PositionalEncoding class.

When the PositionalEncoding layer is called, it takes the input tokens as input and adds the positional encoding matrix (pos_encoding) element-wise to these input tokens. The resulting tensor contains both the original input tokens and the positional information, which is then returned as the output of the PositionalEncoding layer.

C. *Dataset Training*

The dataset used for training the proposed model is the PlantVillage dataset (sources from Kaggle). Initially, it consisted of a total of 16012 images from the Tomato part of the dataset, with ten classes in total. All the images within the dataset have a size of 256x256 pixels. After deleting some images from the dataset to cut down on training time, the Tomato dataset was chosen, with 12239 images and ten classes, with 9791 training images and 2448 testing images.

The dataset was first read in to initiate the training, and some preprocessing was used to prepare the training images. Only training data was preprocessed, whereas testing data remained untouched because the preprocessing methods are used to help the model learn better and more efficiently, thus reaching convergence faster. Having the testing data undergo the preprocessing is unnecessary as it would only slow down the testing process. The images were read in an 64x64-pixel RGB format, with a corresponding label to each image. The images and their labels were then inserted into their respective arrays.

Then, the image array and the label array were each split into training and testing data: x_train and y_train being the training images and their respective labels, as well as x_test and y_test being the testing images and their respective labels as stated in Figure 6. The images were saved in 4-dimensional arrays, whereas the labels were saved in 1-dimensional arrays.

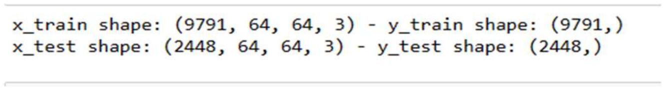

```
x_train shape: (9791, 64, 64, 3) - y_train shape: (9791,)
x_test shape: (2448, 64, 64, 3) - y_test shape: (2448,)
```

Fig. 6 x_train shape, y_train shape and x_test shape, y_test shape

After that, the Shifted Patch Tokenization, Patch Encoder, and Positional Encoding modules were constructed. The Shifted Patch Tokenization module shifts the images by the size of half a patch in each of the four directions: left-up, left-down, right-up, and right-down. (The Shift Patch Tokenization module does not create the patches but merely gives information on the size of the patches.) It does this by first cropping the images to the section opposing the intended shift direction, then creating black padding for the images to replace the cropped section, but towards the shift direction instead of opposing it. It then adds the information from the additional images to the original patches.

On the other hand, the Patch Encoder creates the patches and then encodes the positional information to the patches relative to each other. Then, it calculates the positional data and adds it to the encoded patches, also known as the input tokens. Positional Encoding also adds spatial information in the form of sine and cosine angles to the input tokens created by the Patch Encoder module.

Next, the transformer block was built. The Multi-Head Attention with LSA, and MLP modules were first built separately. Then, the modules were put together to create the transformer. The transformer block consists of 8 layers, with each layer being a preprocessing normalization layer, a Multi Head Attention with LSA module to calculate attention scores, a function to add the attention score to the encoded patches, another preprocessing normalization layer, an MLP module, then finally another function to add the classification tokens from the MLP to the encoded patches [27].

Finally, the rest of the model was built, and all of the components were compiled together for training. A WarmUpCosine module and an Adam optimizer module were created to help accelerate the learning process of the model. The model was then compiled together, with the Shift Patch Tokenization module being first, followed by the Patch Encoder, and then finally the transformer block with the WarmUpCosine and Adam optimizer modules incorporated into the block. The model was then trained to fit the training data, using 30 epochs to train the model, with a batch size of 256 images per batch and a 90:10 validation split, with the 10% split of the data used to validate the model after each training epoch. Two callback functions are used: one to record the time used for each epoch and another for early stopping to prevent overfitting. This model was trained for 19 epochs before the early stopping callback was activated to stop the training process.

III. Result and Discussion

In Figure 7, the final training accuracy of the model was 95.03%, with the final validation accuracy being 74.77%. The training accuracy quickly raised from 38.74% to 85.15% within the first ten epochs before slowing down, whereas the validation accuracy raised from 63.36% to 85.40% within eight epochs before vacillating somewhat between 85% and 90% for the rest of the training.

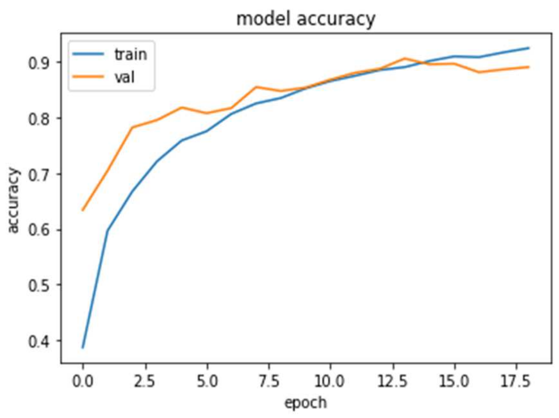


Fig. 7 Model Accuracy

In Figure 8, the final training loss of the model was 0.2236, with the final validation loss being 0.3500. The loss quickly dropped from 2.8832 to 0.7212 within the first 5 epochs before slowing down. On the other hand, the validation loss only dropped from 1.1317 to 0.5264 at around 5 epochs, then leveled off around 0.35 starting from the 10th epoch for the remainder of the training.

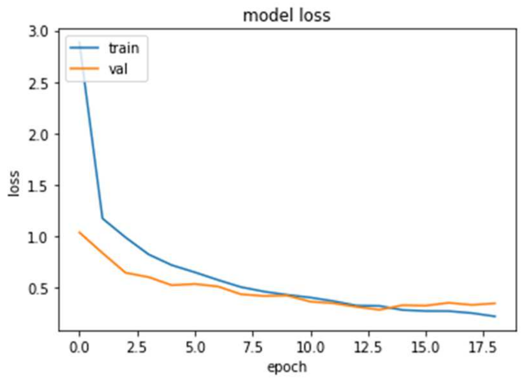


Fig. 8 Model Loss

In Figure 9, the final top-5 training accuracy of the model was 99.89%, with the final top-5 validation accuracy being 99.79%. The top-5 training accuracy rose from 81.09% to 98.47% within the first 5 epochs, then slowed down and leveled off around 99.80% at the 10th epoch until the end of the training. Meanwhile, the top-5 validation accuracy rose from 96.63% to 99.18% within 5 epochs, then slowed down and leveled off around 99.70% at the 8th epoch until the end of the training.

After the training, a final test was done to evaluate the model on unseen data, as stated in Figure 10. The testing accuracy was 89.58%, which is fairly lower when compared to the training accuracy but still within an acceptable range, as it is close to the validation accuracy. Moreover, the testing top-5 accuracy was 98.61%, which means that the model is still reasonably accurate overall, with slight imperfections sometimes causing the correct classification to fall somewhat below the most likely class within the model.

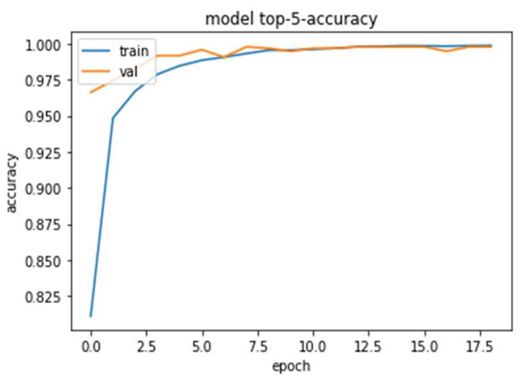
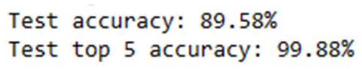


Fig. 9 Top-5-Accuracy


```
Test accuracy: 89.58%
Test top 5 accuracy: 99.88%
```
Fig. 10 Model Accuracy

A. Accuracy Comparison with Other Models in Prior Research

Table 1 shows that the accuracy score of the proposed vision transformer was significantly improved over the traditional vision transformer, as well as various models in prior research, such as the ones by [28] and [29].

TABLE I
ACCURACY COMPARISON BETWEEN DIFFERENT CLASSIFICATION MODELS

Study	Classification Model	Accuracy Score (%)
This study	Traditional Vision Transformer	81.86
This study	Proposed Vision Transformer	90.00
[12]	Inception-V3	77.50
[13]	Mobilenet	88.40
[12]	Mobilenet	82.60

Even so, there was limited computing power, which restricted the input size of the vision transformers during the training process. As a result, the 64x64-pixel RGB input format was chosen instead of the typical 128x128-pixel RGB or 256x256-pixel RGB formats, which would have improved the quality and accuracy of the model, as the model would be able to consider a larger area of the image per token, and thus can better describe the image in its trained weights.[30] Despite that, the fact that the proposed vision transformer still managed to obtain an accuracy of nearly 90% under these harsh conditions shows that the proposed model has much potential for improvement when given better computing power and, thus, better-quality training.

B. Experiment Setting

There are a total of 10 classes within the dataset. The labels of the classes are denoted with integers ranging from 0 to 9. Listed in order of the integers, the classes are named as: Tomato__Target_Spot, Tomato__Tomato_mosaic_virus, Tomato__Tomato_YellowLeaf__Curl_Virus, Tomato_Bacterial_spot, Tomato_Early_blight, Tomato_healthy, Tomato_Late_blight, Tomato_Leaf_Mold, Tomato_Septoria_leaf_spot, Tomato_Spider_mites_Two_spotted_spider_mite.

Two test images from different dataset classes were randomly selected for a small test run: one from the Tomato__Tomato_YellowLeaf__Curl_Virus set and one from the Tomato_healthy set. These images were labeled as yellowleaf1 and healthy1, respectively. Copies of the two images were then given a random rotation (labeled as yellowleaf2, healthy2) and slightly cropped (labeled as yellowleaf3, healthy3), respectively, giving a total of 6 test images (Figure 11 referred). This was to test the algorithm's robustness to the leaf's orientation and position within the image.

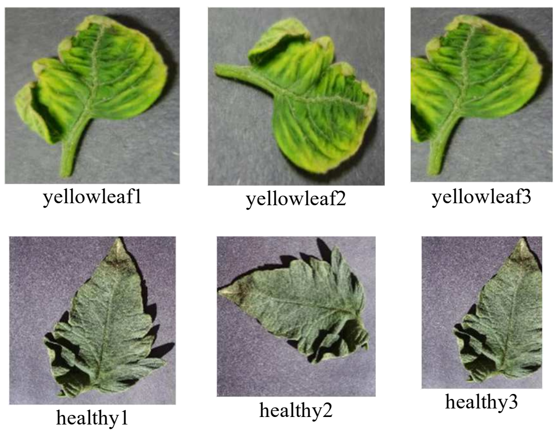


Fig. 11 The 5 test images, manually chosen at random

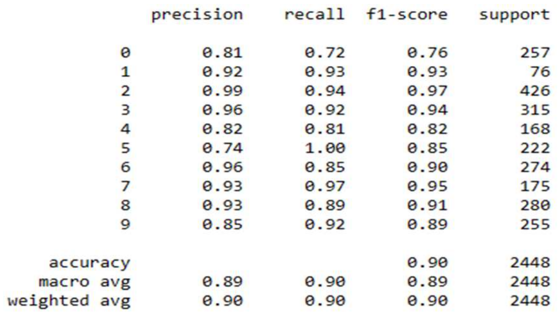
After loading the test images, the model was used to attempt a classification of the images, and the output was compared to the actual classes of the images. The model correctly classified all the test images. Thus, this small-scale experiment proves that this model can identify plant diseases regardless of leaf position and orientation. A classification report is used to analyze the test data results more finely. The overall precision, recall, F1 score and support, and the separate statistics for each test data class are illustrated in Figure 12 below.

In precision, Tomato_healthy (class 5) has relatively low precision (74%), meaning that this class has a lot of false positives during classification, with Tomato__Target_Spot (class 0), Tomato_Early_blight (class 4) and Tomato_Spider_mites_Two_spotted_spider_mite (class 9) having mediocre precision (81%, 82% and 85% respectively). On the other hand, Tomato__Tomato_YellowLeaf__Curl_Virus (class 2) has particularly high precision (99%), with Tomato_Bacterial_spot (class 3) and Tomato_Late_blight (class 6) also having relatively high precision (both 96%).

As for the recall, Tomato__Target_Spot (class 0) has the worst recall (72%), with Tomato_Early_blight (class 4) and Tomato_Late_blight (class 6) having mediocre precision (81% and 85%, respectively). Meanwhile, Tomato_healthy (class 5) has the highest recall (100%), with the next highest being Tomato_Leaf_Mold (class 7, at 97%).

The F1 score of the class represents its overall score, combining the scores of precision and recall into a single category. Thus, using this metric, the class with the overall worst performance is Tomato__Target_Spot (class 0), whereas the class with the overall best performance is Tomato__Tomato_YellowLeaf__Curl_Virus (class 2). Other notable classes are Tomato_healthy (class 5), with low precision but high recall; Tomato_Early_blight (class 4), which was mediocre in both precision and recall; as well as

Tomato_Late_blight (class 6) for having high precision but mediocre recall.



```
              precision    recall  f1-score   support

           0       0.81      0.72      0.76       257
           1       0.92      0.93      0.93        76
           2       0.99      0.94      0.97       426
           3       0.96      0.92      0.94       315
           4       0.82      0.81      0.82       168
           5       0.74      1.00      0.85       222
           6       0.96      0.85      0.90       274
           7       0.93      0.97      0.95       175
           8       0.93      0.89      0.91       280
           9       0.85      0.92      0.89       255

    accuracy                           0.90      2448
   macro avg       0.89      0.90      0.89      2448
weighted avg       0.90      0.90      0.90      2448
```

Fig. 12 Model precision, recall, F1 Score, and support, evaluated for each class

Finally, for support, Tomato__Tomato_mosaic_virus (class 1) has the lowest support, whereas Tomato__Tomato_YellowLeaf__Curl_Virus (class 2) has the highest support. This is explained by the difference in number of images within the classes, with 373 images for Tomato__Tomato_mosaic_virus (class 1) as compared to 2081 images for Tomato__Tomato_YellowLeaf__Curl_Virus (class 2). This roughly explains the high F1 score of Tomato__Tomato_YellowLeaf__Curl_Virus (class 2). However, despite this, Tomato__Tomato_mosaic_virus (class 1) and Tomato_Leaf_Mold (class 7) still have F1 scores of over 90%. Thus, low support is not a major factor in determining low F1 scores, though it still has a noticeable effect. However, interestingly, Tomato__Target_Spot (class 0) has a low F1 score despite having relatively high support. This is presumably due to difficulties in differentiating class 0 from the other classes, as shown by its low precision and recall.

Figure 13 is the model's confusion matrix, which specifies each class's precision and recall. First, a fair number of images were falsely labeled as Tomato_healthy (class 5), and to a lesser extent Tomato__Target_Spot (class 0), Tomato__Early_blight (class 4) as well as Tomato_Spider_mites_Two_spotted_spider_mite (class 9). Classes 0, 4 and 9 were visually similar to healthy leaves. However, only class 5 falls under 80% precision, which means 9 out of 10 classes are still decently trustworthy.

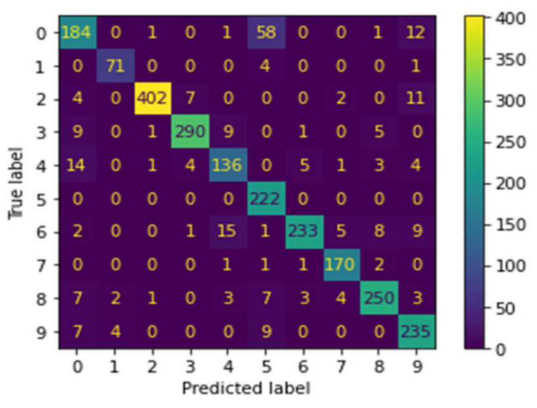


Fig. 13 Confusion Matrix in Multiclass Label

As for recall, despite the relatively low precision of Tomato_healthy (class 5), all images from this class were predicted correctly. Thus, it can be deduced from the results that the model has learned what a healthy leaf looks like, but it has defined it too broadly. Conversely, the images from Tomato__Target_Spot (class 0), Tomato_Early_blight (class 4), and Tomato_Late_blight (class 6) had more false predictions, with most of the false predictions being from class 0. Interestingly, most of the false predictions in class 6 fell under class 4. This is an acceptable outcome because both classes predict the same disease at different stages. Only class 0 falls below a recall of 80%, meaning that for 9 out of 10 classes, the model is decently sensitive to the respective predictions.

IV. Conclusion

Vision Transformer is selected to be the classification model for detecting plant disease. It is effective in image recognition and can learn the detailed features in the input on its own with minimal manual processing. The model can identify plant diseases regardless of leaf position within the image. An application was developed using the transformer model. It is evaluated using accuracy, precision, recall, and F1 score to comprehend its effectiveness more effectively. The transformer model is enhanced by adding the Shift Patch Tokenization, Positional Encoding, Multi-head Attention, and Locality Self Attention modules. It achieved a final test accuracy of 90%, with a significant reduction in variance for accuracy, precision, recall, and F1 score across different dataset classes, thus resulting in more accurate and consistent identification.

However, it is limited by dataset availability, as some of the classes have lower accuracy or F1 scores because of having a small number of images and the limited computing power for model training. While data augmentation can alleviate some of the effects of the small dataset size, it is undeniable that expanding the size and quality of the available datasets would be immensely helpful in improving model performance. Improvements in computing power would also result in better-quality model training. Moreover, given that Vision Transformer research is still relatively new, the features and variables within the additional modules used in the proposed model could still be further improved in future research, giving opportunities to refine further and enhance methods of plant disease identification, thus giving the agricultural industry better tools for effective plant disease treatment.

References

[1] B. S. Bari et al., "A real-time approach of diagnosing rice leaf disease using deep learning-based faster R-CNN framework," PeerJ Computer Science, vol. 7, p. e432, Apr. 2021, doi: 10.7717/peerj-cs.432.

[2] M. Y. Xin, L. W. Ang, and S. Palaniappan, "A Data Augmented Method for Plant Disease Leaf Image Recognition based on Enhanced GAN Model Network," Journal of Informatics and Web Engineering, vol. 2, no. 1, pp. 1–12, Mar. 2023, doi: 10.33093/jiwe.2023.2.1.1.

[3] G. B.V. and U. D. G., "Identifying and classifying plant disease using resilient LF-CNN," Ecological Informatics, vol. 63, p. 101283, Jul. 2021, doi: 10.1016/j.ecoinf.2021.101283.

[4] J. Zhang, Y. Rao, C. Man, Z. Jiang, and S. Li, "Identification of cucumber leaf diseases using deep learning and small sample size for agricultural Internet of Things," International Journal of Distributed Sensor Networks, vol. 17, no. 4, p. 155014772110074, Apr. 2021,

doi:10.1177/15501477211007407.

[5] M. H. Saleem, S. Khanchi, J. Potgieter, and K. M. Arif, “Image-Based Plant Disease Identification by Deep Learning Meta-Architectures,” Plants, vol. 9, no. 11, p. 1451, Oct. 2020, doi: 10.3390/plants9111451.

[6] X. Li and S. Li, “Transformer Help CNN See Better: A Lightweight Hybrid Apple Disease Identification Model Based on Transformers,” Agriculture, vol. 12, no. 6, p. 884, Jun. 2022, doi:10.3390/agriculture12060884.

[7] M. A. Genaev, E. S. Skolotneva, E. I. Gultyaeva, E. A. Orlova, N. P. Bechtold, and D. A. Afonnikov, “Image-Based Wheat Fungi Diseases Identification by Deep Learning,” Plants, vol. 10, no. 8, p. 1500, Jul. 2021, doi: 10.3390/plants10081500.

[8] J. Chen, D. Zhang, A. Zeb, and Y. A. Nanehkaran, “Identification of rice plant diseases using lightweight attention networks,” Expert Systems with Applications, vol. 169, p. 114514, May 2021, doi:10.1016/j.eswa.2020.114514.

[9] R. Reedha, E. Dericquebourg, R. Canals, and A. Hafiane, “Transformer Neural Network for Weed and Crop Classification of High Resolution UAV Images,” Remote Sensing, vol. 14, no. 3, p. 592, Jan. 2022, doi: 10.3390/rs14030592.

[10] H.-T. Thai, N.-Y. Tran-Van, and K.-H. Le, “Artificial Cognition for Early Leaf Disease Detection using Vision Transformers,” 2021 International Conference on Advanced Technologies for Communications (ATC), Oct. 2021, doi:10.1109/atc52653.2021.9598303.

[11] Y. Xiong, L. Liang, L. Wang, J. She, and M. Wu, “Identification of cash crop diseases using automatic image segmentation algorithm and deep learning with expanded dataset,” Computers and Electronics in Agriculture, vol. 177, p. 105712, Oct. 2020, doi:10.1016/j.compag.2020.105712.

[12] Z. Zhang, Z. Gong, Q. Hong, and L. Jiang, “Swin-Transformer Based Classification for Rice Diseases Recognition,” 2021 International Conference on Computer Information Science and Artificial Intelligence (CISAI), Sep. 2021, doi: 10.1109/cisai54367.2021.00036.

[13] Y. Borhani, J. Khoramdel, and E. Najafi, “A deep learning based approach for automated plant disease classification using vision transformer,” Scientific Reports, vol. 12, no. 1, Jul. 2022, doi:10.1038/s41598-022-15163-0.

[14] W. Zhu, J. Sun, S. Wang, J. Shen, K. Yang, and X. Zhou, “Identifying Field Crop Diseases Using Transformer-Embedded Convolutional Neural Network,” Agriculture, vol. 12, no. 8, p. 1083, Jul. 2022, doi:10.3390/agriculture12081083.

[15] S. Zhang, S. Zhang, C. Zhang, X. Wang, and Y. Shi, “Cucumber leaf disease identification with global pooling dilated convolutional neural network,” Computers and Electronics in Agriculture, vol. 162, pp. 422–430, Jul. 2019, doi: 10.1016/j.compag.2019.03.012.

[16] J. G. Arnal Barbedo, “Plant disease identification from individual lesions and spots using deep learning,” Biosystems Engineering, vol. 180, pp. 96–107, Apr. 2019, doi:10.1016/j.biosystemseng.2019.02.002.

[17] A. F. Fuentes, S. Yoon, J. Lee, and D. S. Park, “High-Performance Deep Neural Network-Based Tomato Plant Diseases and Pests Diagnosis System With Refinement Filter Bank,” Frontiers in Plant Science, vol. 9, Aug. 2018, doi: 10.3389/fpls.2018.01162.

[18] P. Sharma, Y. P. S. Berwal, and W. Ghai, “Performance analysis of deep learning CNN models for disease detection in plants using image segmentation,” Information Processing in Agriculture, vol. 7, no. 4, pp. 566–574, Dec. 2020, doi: 10.1016/j.inpa.2019.11.001.

[19] A. M. Roy and J. Bhaduri, “A Deep Learning Enabled Multi-Class Plant Disease Detection Model Based on Computer Vision,” AI, vol. 2, no. 3, pp. 413–428, Aug. 2021, doi: 10.3390/ai2030026.

[20] S. Wu, Y. Sun, and H. Huang, “Multi-granularity Feature Extraction Based on Vision Transformer for Tomato Leaf Disease Recognition,” 2021 3rd International Academic Exchange Conference on Science and Technology Innovation (IAECST), Dec. 2021, doi:10.1109/iaecst54258.2021.9695688.

[21] J. Pandian, V. Kumar, O. Geman, M. Hnatiuc, M. Arif, and K. Kanchanadevi, “Plant Disease Detection Using Deep Convolutional Neural Network,” Applied Sciences, vol. 12, no. 14, p. 6982, Jul. 2022, doi: 10.3390/app12146982.

[22] D. Wang, J. Wang, W. Li, and P. Guan, “T-CNN: Trilinear convolutional neural networks model for visual detection of plant diseases,” Computers and Electronics in Agriculture, vol. 190, p. 106468, Nov. 2021, doi: 10.1016/j.compag.2021.106468.

[23] D. Argüeso et al., “Few-Shot Learning approach for plant disease classification using images taken in the field,” Computers and Electronics in Agriculture, vol. 175, p. 105542, Aug. 2020, doi:10.1016/j.compag.2020.105542.

[24] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” Oct. 2020, [Online]. Available: http://arxiv.org/abs/2010.11929

[25] S. H. Lee, H. Goëau, P. Bonnet, and A. Joly, “New perspectives on plant disease characterization based on deep learning,” Computers and Electronics in Agriculture, vol. 170, p. 105220, Mar. 2020, doi:10.1016/j.compag.2020.105220.

[26] S. Ramesh and D. Vydeki, “Recognition and classification of paddy leaf diseases using Optimized Deep Neural network with Jaya algorithm,” Information Processing in Agriculture, vol. 7, no. 2, pp. 249–260, Jun. 2020, doi: 10.1016/j.inpa.2019.09.002.

[27] J. Annrose, N. H. A. Rufus, C. R. E. S. Rex, and D. G. Immanuel, “A Cloud-Based Platform for Soybean Plant Disease Classification Using Archimedes Optimization Based Hybrid Deep Learning Model,” Wireless Personal Communications, vol. 122, no. 4, pp. 2995–3017, Sep. 2021, doi: 10.1007/s11277-021-09038-2.

[28] M. Agarwal, S. Kr. Gupta, and K. K. Biswas, “Development of Efficient CNN model for Tomato crop disease identification,” Sustainable Computing: Informatics and Systems, vol. 28, p. 100407, Dec. 2020, doi: 10.1016/j.suscom.2020.100407.

[29] R. and I. T. Association of Knowledge, Jāmiʻat Ibn Zuhr. École nationale des sciences appliquées d’Agadir, and Institute of Electrical and Electronics Engineers, *Proceedings of 2019 International Conference of Computer Science and Renewable Energies (ICCSRE) : 2019 July 22-24*.

[30] Institute of Electrical and Electronics Engineers and Hindusthan Institute of Technology, *Proceedings of the International Conference on Electronics and Sustainable Communication Systems (ICESC 2020) : 02-04, July 2020.*