Implementation Of Hybrid EfficientNet V2 And Vision Transformer for Apple Leaf Diseases Classification

Sri Fuji Santoso [a,1], Surjohadi [b,2], Budi Nugroho [a,3], I Gede Susrama Mas [c,4,*]

[a] Department of Informatics, Faculty of Computer Science, UPN "Veteran" Jawa Timur, Surabaya, Indonesia
[b] Department of Industrial Engineering, Faculty of Engineering, University of Yos Soedarso Surabaya, Indonesia
[c] Master of Information System, Faculty of Computer Science, UPN "Veteran" Jawa Timur, Surabaya, Indonesia
[1] 20081010184@student.upnjatim.ac.id; [2] bestsuryo@yahoo.com; [3] budinugroho.if@upnjatim.ac.id; [4] igsusrama.if@upnjatim.ac.id
* Corresponding author

ARTICLE INFO

Article history
Received January 25, 2025
Revised April, 16 2025
Accepted May, 3 2025

Keywords
Machine Learning
EfficientNet V2
Vision Transformer
Apple Leaf Diseases

ABSTRACT

The apple farming industry faces challenges in managing apple leaf diseases. Current manual detection methods have limitations in expertise variability, time required, potential delays in identification leading to disease spread, and difficulty distinguishing diseases with similar visual symptoms. This research aims to develop an accurate, efficient, and automated apple leaf disease classification system using a hybrid approach that combines EfficientNet V2 architecture and Vision Transformer. The main objectives are to improve disease detection accuracy, reduce computational requirements, facilitate more effective plant management, and support modern agricultural practices in the apple industry. This research uses a hybrid deep learning model that integrates EfficientNet V2 and Vision Transformer components. Experiments were conducted on an apple leaf disease dataset to evaluate model performance. Results show the effectiveness of this method in classifying apple leaf diseases, achieving 98.56% accuracy and an F1 score of 0.9856 on test data. The proposed model has 15.6 million parameters, lighter than the original EfficientNetV2S model with 20 million parameters. Training time was reduced to 6 minutes 32 seconds compared to the original EfficientNetV2S model that required 8 minutes 41 seconds for 5 epochs on the same dataset.

1. Introduction

The apple farming industry plays a crucial role in Indonesia's economic growth. Apples not only contribute to food security but also serve as a source of income for fruit farmers in Indonesia. However, the industry's productivity has declined significantly. According to recent data from the Indonesian Statistics Agency (BPS), apple farming productivity in Indonesia decreased from 523,596 in 2022 to 392,563 in 2023 [1]. BBPP Ketindan informed that one factor that cause a decline in apple production is diseases that attack apple trees [2]. Apple leaf diseases can reduce apple plant health, which can impact apple fruit production. Early detection and accurate diagnosis of apple leaf diseases are essential components of effective plant management [3,4]. Traditional methods relying on visual inspection by plant pathology experts have several limitations, such as high labor intensity, potential detection delays, limited coverage, and high costs. The complexity of diagnosing apple leaf diseases can be exacerbated by factors such as symptom variations and similarities between diseases [5,6].

Advancements in computer vision and machine learning have opened new opportunities to address these challenges. Deep learning techniques, particularly Convolutional Neural Networks (CNNs),

have shown promising results in classifying infected leaf images. Previous research using VGG 16 architecture achieved high accuracy [7]. However, implementing this model in the field still faces challenges in classification accuracy and computational efficiency. Other study implemented ResNet-50 for apple leaf disease classification, they achieved 91% accuracy but this research still faced challenges computation and training speed [8]. EfficientNet was introduced by Tan and Le in 2019, represents a breakthrough in efficient CNN architecture design that uses compound scaling techniques to systematically balance network depth, width, and resolution [9]. EfficientNet V2, proposed by the same authors, further improves efficiency and accuracy through architecture optimization [10]. Dosovitskiy et al. introduced Vision Transformer (ViT), adapting the successful Transformer architecture from natural language processing to the computer vision domain. ViT shows competitive performance with state-of-the-art CNNs on large-scale image classification tasks, with advantages in capturing long-range dependencies in images [11].

Deep learning architectures for classification have advanced rapidly. EfficientNet V2 enhances efficiency, and Vision Transformer can capture complex dependencies [12]. A hybrid of EfficientNet V2 and Transformer combines their strengths. This study explores its potential aiming to improve the model for practical use.

2. Method

2.1. Data Collection

The data for this study was sourced from Kaggle.com, a platform offering various datasets in formats such as image, CSV, JSON, and SQLite. The dataset used includes 7,200 images of apple leaf diseases, categorized into four labels Apple Scab, Cedar Apple Rust, Black Rot, and Healthy.

2.2. Data Preproccessing

Before the dataset is used, it is preprocessed to match the model's expected input format. Data augmentation techniques, such as resizing, random rotation, affine transformation, horizontal flips, and color jitter, are applied to create varied images [13].

2.2.1 Resize

The chosen image input size is 224 x 224 pixels to balance computational efficiency and image detail clarity. Because larger sizes would increase computation, while smaller sizes might lose important details. This size is also widely used in computer vision research. In PyTorch, the `resize` function typically uses bilinear interpolation, which estimates unknown values based on the four nearest known points [14]. Matematically it's formulated as follow.

$$P = (1 - t\_y)[(1 - t\_x)Q11 + t\_x\ Q21] + t\_y[(1 - t\_x)Q12 + t\_x\ Q22] \quad (3.1)$$

Where P is the value to be estimated at point (x, y). Q11, Q12, Q21, Q22 are known values at the four corner points of the rectangle enclosing point (x, y). (x1, y1) is the coordinate of the bottom-left corner point of the rectangle. (x2, y2) is the coordinate of the top-right corner point of the rectangle. t_x is the relative distance between x and x1 on the x-axis, calculated as t_x = (x - x1) / (x2 - x1). t_y is the relative distance between y and y1 on the y-axis, calculated as t_y = (y - y1) / (y2 - y1)

2.1.2 Random Rotation

Objects input to the model in real world scenarios may be tilted or rotated at various angles. To address this, random rotation up to 20 degrees is applied to train the model to recognize objects regardless of orientation. Mathematically, rotation can be expressed as follows.

$$x' = \cos(theta) * (x - center\_x) - \sin(theta) * (y - center\_y) + center\_x \quad (3.2)$$

$$y' = \sin(theta) * (x - center\_x) + \cos(theta) * (y - center\_y) + center\_y \quad (3.2)$$

Where (x, y) are the original pixel coordinates in the image, (x', y') are the pixel coordinates after rotation and translation, theta is the random rotation angle chosen from a specified range (in radians) center_x, center_y are the normalized rotation center coordinates between 0 and 1

2.1.3 Random Affine

In real world use, input images may be zoomed in or out. To handle this, random affine transformation is applied that consist of random rotations, translations, and scaling. Mathematically, the random affine operation can be expressed as follows.

$$x' = a * x + b * y + c \quad (3.3)$$

$$y' = d * x + e * y + f \quad (3.4)$$

Where (x, y) are the original pixel coordinates (x', y') are the pixel coordinates after transformation, a and e are scaling and rotation parameters on x and y axes, b and d are shearing parameters on x and y axes, c and f are translation parameters on x and y axes.

2.1. 4 Random Horizontal Flip

To further reduce the model's reliance on image orientation, random horizontal flip augmentation is applied. Mathematically, the random horizontal flip operation can be expressed as follows.

$$x' = width - x - 1 \quad (3.5)$$

$$y' = y \quad (3.6)$$

Where (x, y) are the original pixel coordinates, (x', y') are the pixel coordinates after horizontal flipping width is the image width, y remains unchanged as the processing is horizontal

2.1. 4 Random Color Jitter

In real world use, input image lighting can vary greatly depending on whether photos are taken in the morning or evening. To address this, color jitter augmentation was applied to randomly adjust image contrast and brightness by 20% to trains the model to recognize objects despite minor color or lighting changes. Mathematically, the color jitter operation can be expressed as:

$$((old\_value - mean) * contrast\_factor + mean) * saturation\_factor + (1 - saturation\_factor) * grayscale + brightness\_facto \quad (3.7)$$

Where old_value represents the original value of a color channel (red, green, or blue) in a pixel, mean is the average value of all pixels in that color channel. Grayscale is the grayscale value of the pixel, calculated as the average of the three color channels (R, G, B). Brightness_factor, contrast_factor, and saturation_factor are randomly selected adjustment factors within a specified range. new_valu is the transformed value of the color channel after applying the adjustments.

2.2. Model Creation

Table 1. Model Architecture

No	Layer	Channel/Dim	Stride	Total
1	Conv 3x3	24	2	1
2	Fused MBConv 3x3, e = 1	32	1	1
3	MBConv 16x16 Patch Embedding, e = 1	192	8	1
4	Transformer Encoder, e = 2	192	-	3
5	MBConv 3x3, e = 2	192	1	6
6	MBConv 5x5 Patch Embedding, e = 1	384	2	1
7	Transformer Encoder, e = 2	384	-	6
8	MBConv 3x3, e = 2	384	1	6
9	MBConv 3x3 Patch Embedding, e = 1	768	1	1
10	Transformer Encoder , e = 2	768	-	2
11	Adaptive Average Pooling	768	-	1
12	Linear	768	-	1

e = expansion ration of the channel expansion

In the EfficientNetV2 paper, it is explained that depthwise convolution operates slowly when implemented in early layers but becomes effective when applied in middle and final layers. Therefore, the initial layers, as in the original EfficientNetV2 model, use a standard 3x3 convolution and FusedMBConv. The first layer employs a stride of 2 to halve the input's spatial dimensions, improving computational efficiency. The Fused MBConv and MBConv blocks remain identical to those in the original model.

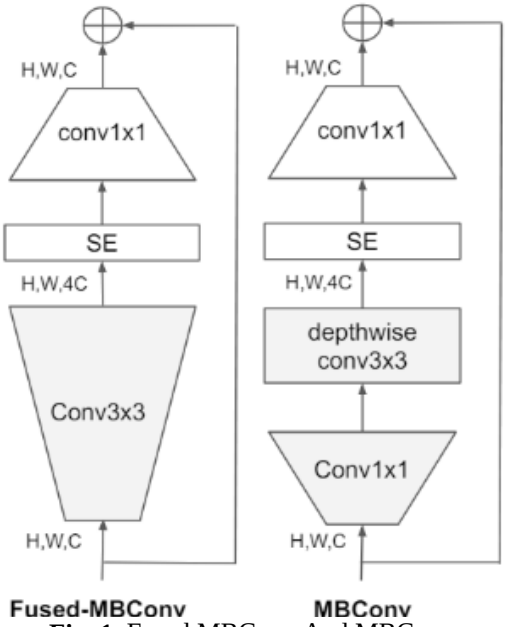

Fig. 1. Fused MBConv And MBConv

The SE in figure 1 is Squeeze Exitation that is a channel attention mechanism which in mathematical can be described as below.

$$SE = Sigmoid(Conv2(SiLU(Conv1(Avg\_Pool(x))))) \quad (3.8)$$

Where sigmoid represents sigmoid activation, conv is 1x1 convolution, silu is silu activation, and avg_pool is average pooling. MBConv is also used to create overlapping input patches for the Transformer Encoder, and repatch to change the patch size while doubling the channel. The modifications include adding Transformer Encoders to the architecture to capture global relationships, the Transformer Encoder can be explained as below.

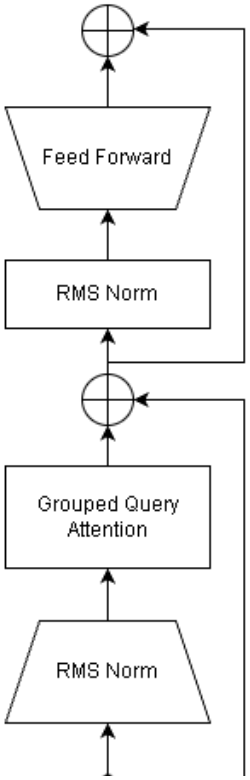

Fig. 2. Transformer Encoder

The encoder consists of three components, RMS Norm for input normalization, Multi Head Self Attention for extracting global features, and a Feed Forward Network that processes the extracted features into a higher-dimensional space. RMS Norm is used instead of Layer Norm because, as explained in the RMS Norm paper, the beneficial effect of Layer Norm comes from its rescaling rather than its recentering. In RMS Norm, this recentering is eliminated, providing the same effect with lighter computation since it doesn't need to calculate the mean. The RMS Norm is as follows.

$$a_i = \frac{a_i}{RMS(a)} g_i \tag{3.9}$$

$$RMS(a) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} a_i^2} \tag{3.10}$$

Where a represents input vector and ai represents each input element in the vector, and gi is a learnable parameter.

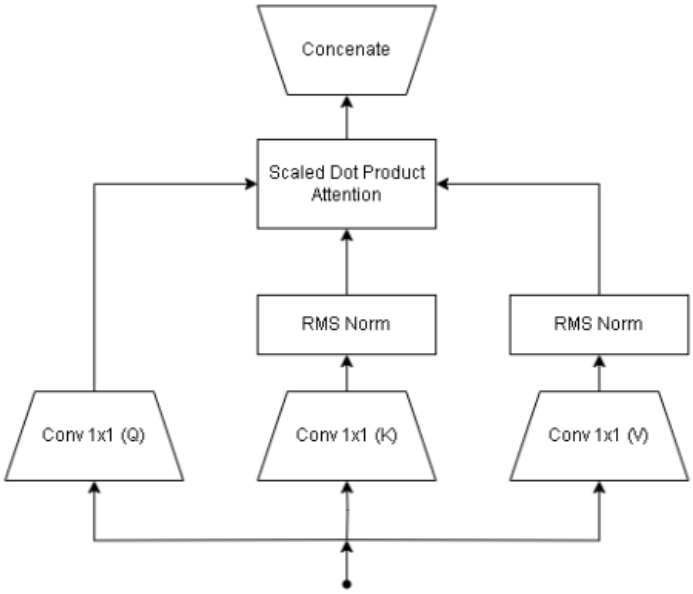

Fig. 3. Multihead Self Attention

In the Multi-Head Self-Attention, three linear projections are applied to the input, resulting in matrices Q (query), K (key), and V (value) [8]. This is done using either a linear layer or a 1x1 convolution with zero padding. In this study, a 1x1 convolution is used to combine the advantages of convolution and the self-attention mechanism. Here RMS Norm is also used to normalize key and value because in the experiment it prevents gradient exploding in the modified model. Each Q, K, and V is divided into multiple groups, referred to as heads, with the number of heads equal to the total number of groups. Each head has a dimension of (initial dimension) / (number of heads). The Multi Head Self Attention used is not the standard version, but a variation called Grouped Query Attention. The only difference is several query groups share the same key and value to reduce computational cost.

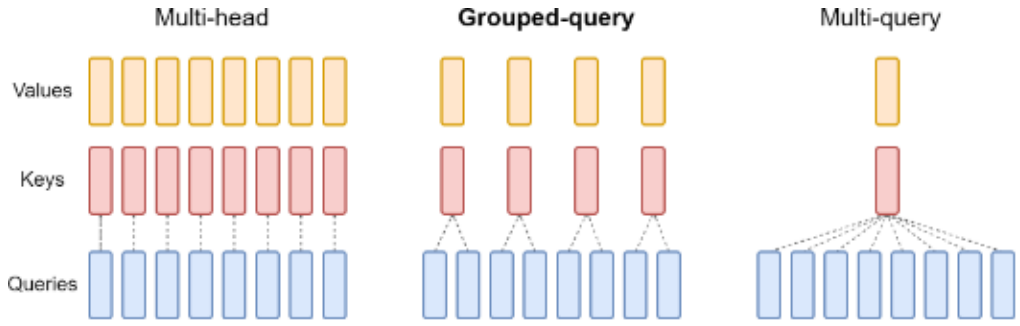

Fig. 4. Traditional MHSA, GPQA Variance, MQA Variance

The image above illustrates three variations of Multi Head Self Attention. On the left, each query, key, and value has its own distinct values. In the middle, a group of queries shares the same key and value, where one group can consist of two or more queries. On the right, all queries use the same key and value. The middle variation is chosen to balance computational efficiency and accuracy. Each group consists of two queries. Then, attention is calculated for each head using the following mathematical formula.

$$Attention\ Output = Softmax\left(\frac{Q\ @\ K^T}{\sqrt{h\_dim}}\right)@\ V \tag{3.11}$$

Where Q is the query matrix, K is the transpose of the key matrix, V is the value matrix, h_dim is the dimension of the head, @ represents matrix multiplication, and softmax is a function that converts values into a range between 0 and 1, defined by the following formula:

$$Softmax\ (x_i) = \frac{\exp(x_i)}{\sum_j \exp\ (x_j)} \tag{3.12}$$

Where exp denotes the exponential function, xi is the input, and ∑_j exp(x_j) represents the sum of all exponential results for each x in the same row.

In the feed forward network, 1x1 convolutions are also used instead of linear layers as in the original Transformer architecture. Furthermore, SiLU activation is employed instead of GELU because, in the experiment SiLU yielded higher accuracy. The input to SiLU is first scaled by multiplying it with a learnable parameter, this scaling resulted in more stable accuracy improvements across epochs in the training in the experiment.

In the original Vision Transformer, classification is performed using a class token. The class token is a special token added to the embedding of image patches and represents the overall information of all tokens. It acts as a kind of representative for all tokens. In the Vision Transformer paper, it is also explained that classification can be done by averaging all tokens. This architecture does not use a class token but instead employs a different approach by applying global average pooling to all tokens using the `nn.AdaptiveAvgPool1d` function, which averages the values of all tokens. This approach has the advantage of utilizing all available tokens.

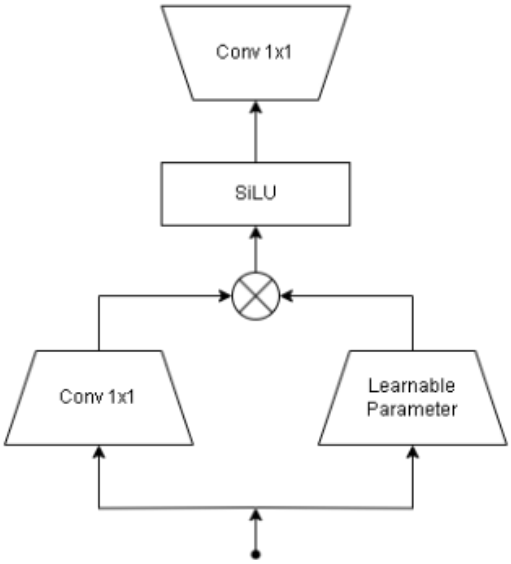

Fig. 5. Feed Forward

2.3. Model Training

In the model training phase, both the modified model and the original model trained using CrossEntropyLoss to measure discrepancies between predicted and actual label distributions. The AdamW optimizer is employed with a 0.0005 learning rate and L2 regularization to prevent overfitting, while StepLR scheduling reduces the learning rate every 3 epochs by 0.97. Training runs for 5 epochs on an Nvidia T4 16GB GPU via Google Colab.

2.4. Model Testing

The evaluation will be performed using accuracy metrics, F1 Score, ROC AUC, Cohen's Kappa, and balanced accuracy using the Scikit Learn library. The data used for evaluation consists of about 250 images for each label. Then, the evaluation results will be compared.

To use all that evaluations, first using 4 metrics, True Positives (TP) for class I that is the number of samples actually belonging to class i and predicted to belong to class i by the model, False Positives (FP) for class I that is the number of samples not actually belonging to class i but predicted to belong to class i by the model, True Negatives (TN) for class I that is the number of samples not actually belonging to class i and predicted not to belong to class i by the model, False Negatives (FN) for class I that is the number of samples that actually Accuracy measures the proportion of correct predictions out of total predictions.This metric is the simplest. Mathematically, accuracy is calculated with the formula

$$\text{Accuracy} = (\Sigma i = 1 \text{ to } n\_classes\ TP\_i) / (\text{Total number of samples}) \quad (3.13)$$

Where n_classes is the number of classes, TP_i is True Positives for class i.

F1 Score combines precision and recall into one metric. A high F1-score indicates a good balance between precision and recall. Precision measures how accurate the model is in predicting certain classes, calculated by TP_i / (TP_i + FP_i), and recall measures how well the model finds all samples that actually belong to certain classes, calculated by TP_i / (TP_i + FN_i). Thus mathematically, F1 score is formulated as follows

$$F1\_score\_i = 2 * (Precision\_i * Recall\_i) / (Precision\_i + Recall\_i) \quad (3.14)$$

Cohen's Kappa measures the level of agreement between model predictions and actual labels, considering the possibility of agreement occurring by chance. A high Kappa indicates that the agreement between predictions and actual labels is not coincidental. Cohen's Kappa is mathematically formulated as

$$\kappa = (po - pe) / (1 - pe) \quad (3.15)$$

Where po is the proportion of observed agreement between predictions and actual labels, pe is the proportion of agreement expected by chance.

Log Loss measures how well the probabilities predicted by the model match the actual labels. The lower the log loss, the better the model's probability calibration. The log loss formula is like below

$$\text{Log Loss} = -1/N * \Sigma i = 1 \text{ to } N\ \Sigma j = 1 \text{ to } M\ yij \log(pij) \quad (3.16)$$

Where N is the number of samples, M is the number of classes, yij is 1 if sample i belongs to class j, 0 if not, pij is the model's predicted probability that sample i belongs to class j.

Balanced Accuracy measures model accuracy while considering class imbalance. This is to validate that the model remains good if some classes have more samples than others. The balanced accuracy formula is like below

$$\text{Balanced Accuracy} = 1/nclasses * \Sigma i = 1 \text{ to } nclasses\ TP\_i / (TP\_i + FN\_i) \quad (3.17)$$

Where n_classes is the number of classes, TP_i is true positives for class i, FN_i is false negatives for class i.

2.5. Model Deployment

In the deployment phase, the EfficientNetV2S model is hosted on a website's backend server using Flask to handle computational demands, rather than running on user devices. The frontend is built with SvelteKit for a responsive interface. This architecture ensures efficient performance by processing model inference on the server, preventing memory constraints on user devices and enabling smooth operation regardless of device capabilities.

Fig. 6. Image of Apple Scab, Cedar Apple Rust, Black Root, Healthy

3. Results and Discussion

3.1. Data Collection Results

The result of data collection obtained from the website kaggle.com is 7,200 images of apple leaf diseases. In these images, there are 4 classes: Apple Scab, Cedar Apple Rust, Black Rot, and Healthy.

3.2. Data Preproccessing

Preprocessing applied involves resizing images and performing data augmentation. The augmentation includes random rotation, random affine, random horizontal flip, and random color jitter. Data augmentation results in appearances different from the original data. The results include the following.

Fig. 7. Random Rotation, Random Affine, Random Horizontal Flip, Color Jitter

3.3. Model Creation Result

The resulting model is a hybrid EfficientNetV2S-Vision Transformer architecture. The trained model for this research dataset can be downloaded at https://drive.google.com/file/d/1qR0pjnlzaV7uXf81TJCYxKYLOEh2Z8B/view?usp=sharing. Additionally, the model architecture has been packaged into a library for easy application to other datasets, available at https://github.com/fuji184/fuji_model.

3.4. Model Training Result

```
train Loss: 1.1545 Acc: 0.6740
val Loss: 0.3809 Acc: 0.8260

Epoch 1/4
----------
train Loss: 0.5442 Acc: 0.8192
val Loss: 0.2356 Acc: 0.9176

Epoch 2/4
----------
train Loss: 0.4061 Acc: 0.8642
val Loss: 0.3187 Acc: 0.8733

Epoch 3/4
----------
train Loss: 0.3781 Acc: 0.8731
val Loss: 0.1828 Acc: 0.9537

Epoch 4/4
----------
train Loss: 0.3082 Acc: 0.8995
val Loss: 0.0634 Acc: 0.9815

Training complete in 6m 32s
Best val Acc: 0.981462
```

```
train Loss: 1.0761 Acc: 0.5101
val Loss: 0.7251 Acc: 0.6931

Epoch 1/4
----------
train Loss: 0.7889 Acc: 0.6833
val Loss: 0.3181 Acc: 0.9073

Epoch 2/4
----------
train Loss: 0.5144 Acc: 0.8144
val Loss: 0.1520 Acc: 0.9526

Epoch 3/4
----------
train Loss: 0.3482 Acc: 0.8743
val Loss: 0.1055 Acc: 0.9670

Epoch 4/4
----------
train Loss: 0.2329 Acc: 0.9200
val Loss: 0.0579 Acc: 0.9856

Training complete in 8m 41s
Best val Acc: 0.985582
```

Fig. 8. Training of the modified model and the original model

Both the modified and original models achieved comparable final validation accuracies on validation data, approximately 0.98. However, the modified model demonstrated reduction in training time, completing training in 6 minutes 32 seconds compared to the original model's 8 minutes 41 seconds. This faster training time suggests improved training efficiency, which can be particularly beneficial in scenarios with limited computational resources.

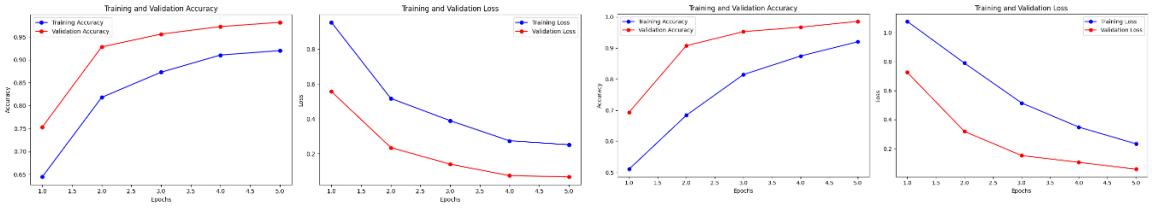

Fig. 9. Training graph of the modified model (left) and training graph of the original model (right)

Both models exhibit a similar trend in accuracy improvement and a similar decline in loss, with both showing stable performance, indicating that the modified model performs comparably to the original model on the validation data.

3.5. Model Evaluation Result

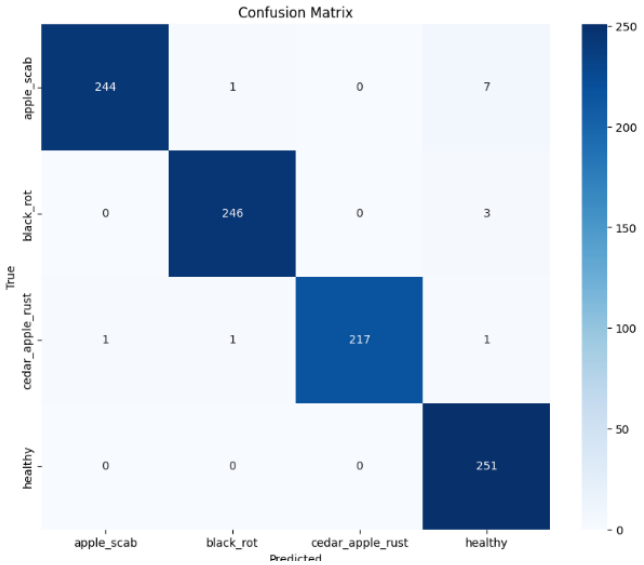

Fig. 10. Confussion Matriks

The confusion matrix above illustrates the performance of the model. The diagonal values represent correctly classified samples, with high accuracy observed across all classes. Aapple Scab has 244 correct predictions with 8 misclassifications, Black Rot has 246 correct predictions with only 3 errors, Cedar Apple Rust shows 217 correct predictions and 2 misclassifications, and Healthy has 251 correct predictions with 3 errors. The small number of off diagonal values indicates that the model performs well.

	precision	recall	f1-score	support
apple_scab	0.9959	0.9683	0.9819	252
black_rot	0.9919	0.9880	0.9899	249
cedar_apple_rust	1.0000	0.9864	0.9931	220
healthy	0.9580	1.0000	0.9786	251
accuracy			0.9856	972
macro avg	0.9865	0.9856	0.9859	972
weighted avg	0.9860	0.9856	0.9856	972

	precision	recall	f1-score	support
apple_scab	0.9960	0.9802	0.9880	252
black_rot	0.9880	0.9960	0.9920	249
cedar_apple_rust	0.9778	1.0000	0.9888	220
healthy	0.9839	0.9721	0.9780	251
accuracy			0.9866	972
macro avg	0.9864	0.9871	0.9867	972
weighted avg	0.9867	0.9866	0.9866	972

Fig. 11. Classification report of the modified model (left) and the original model (right)

The precision and recall values are consistently high across all categories, indicating that the modified model performs well in correctly identifying both positive instances and minimizing false positives. Specifically, the highest precision (1.000) is achieved for cedar apple rust, while the lowest precision (0.9580) is observed for the healthy class. The overall accuracy of the model is 98.56%, with a macro average F1-score of 98.56% and a weighted average F1-score of 98.56%, reflecting balanced and reliable performance across all classes. The overal performance of the modified model and the original model are very similar, 98.56% and 98.66%.

Metric	Value
ROC AUC	1.000
Cohen's Kappa	0.981
Log Loss	0.051
Balanced Accuracy	0.986

Fig. 12. Additional report

The figure 12 summarizes additional evaluation metrics for the model's performance. The ROC AUC score is 1.000, indicating perfect discrimination between classes. Cohen's Kappa is 0.981, suggesting a very high level of agreement between the model's predictions and the true labels. The Log Loss is 0.051, showing that the model's probabilistic predictions are highly confident and accurate. Lastly, the Balanced Accuracy is 0.986, reflecting the model’s consistent performance across all classes, even in the presence of potential class imbalance. These metrics further confirm the robustness and reliability of the model.

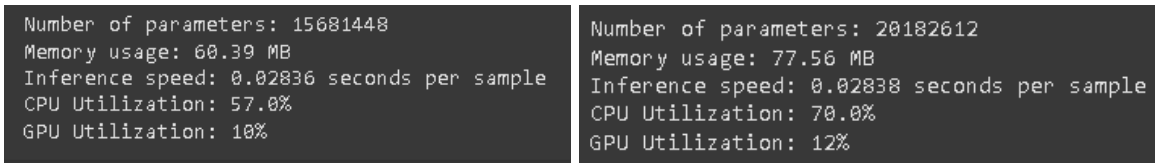

Fig. 13. The modified model (left) and the original model (right)

A further comparison of the modified architecture and the original architecture is that modified model has 15,681,440 parameters and a 60.39 MB memory footprint, outperforms original architecture, which has 20,102,612 parameters and uses 77.56 MB of memory. The modified model also more CPU efficient (57.0%) than the original model (70.0%). While both models exhibit GPU efficientcy, overall, the modified model is more efficient than the orriginal model, makes it a good choice for resource constrained deployments.

3.6. Model Deployment Result

After the model is fully developed and trained, the model is deployed to website to make it easier to be used. The main feature of the website is to take input from the user and send it to the model for prediction then send the result back to the user.

Fig. 14. Deployment

The left side is the homepage, which serves as the main navigation area. On the right of it is upload page where user can submit their image. The right side is the result page that display the model prediction result and treatments.

4. Conclusion

Based on the study, modifying the EfficientNetV2-S architecture with an optimized Transformer Encoder allows for maintaining 98% accuracy, similar to the original model, while reducing parameters and computational costs. The modified model trains faster in 6 minutes and 32 seconds, compared to 8 minutes and 41 seconds in the original model. The modified model has fewer parameters, 15 million compared to 20 million in the original model. It also requires less RAM for inference 60 MB compared to 77 MB in the original model. This demonstrates that CNN and Transformer can work together efficiently, to achieve high accuracy with reduced resource consumption.

Declarations

Author contribution. Creating model architecture that combine EfficientNet V2 and Vision Transformer by selection different parts of both model and then designing a new model that consist of both parts.

Data and Software Availability Statements

The Google Colab repository where the experiment was conducted can be accessed here https://colab.research.google.com/drive/1g9o66Y2jcD2a66aquVSN1uRdl1d5HHvG?usp=sharing. The data used in the experiment can be accessed here https://drive.google.com/drive/folders/1-AbGWIkuv8IFpNfQfmNAjP0acaen2zOk?usp=sharing. The model code, packaged as a library, can be accessed here https://github.com/fuji-184/fuji_model. The website code can be accessed here https://github.com/fuji-184/AppLeDiTion.

References

[1] BPS, "Produksi Tanaman Buah-buahan 2021-2023," 2024. [Online]. Available: URL:https://www.bps.go.id/id/statistics-table/2/NjIjMg==/produksi-tanamanbuah-buahan.html.

[2] Yeniartha, "Upgrade Kapasitas Dan Kelembagaan Petani, Kementan Tingkatkan Produksi Komoditas Apel Malang," 2024. [Online]. Available: URL:https://bbppketindan.bppsdmp.pertanian.go.id/blog/post/upgradekapasitas-dan-kelembagaan-petani-kementan-tingkatkan-produksi-komoditasapel-malang.

[3] Bansal, P., Kumar, R., & Kumar, S. (2021). Disease detection in apple leaves using deep convolutional neural network. *Agriculture*, *11*(7), 617.

[4] Khan, A. I., Quadri, S. M. K., Banday, S., & Shah, J. L. (2022). Deep diagnosis: A real-time apple leaf disease detection system based on deep learning. *computers and Electronics in Agriculture*, *198*, 107093.

[5] Gao, Y., Cao, Z., Cai, W., Gong, G., Zhou, G., & Li, L. (2023). Apple leaf disease identification in complex background based on BAM-net. Agronomy, 13(5), 1240.

[6] Vishnoi, V. K., Kumar, K., Kumar, B., Mohan, S., & Khan, A. A. (2022). Detection of apple plant diseases using leaf images through convolutional neural network. IEEE Access, 11, 6594-6609.

[7] Paymode, A. S., & Malode, V. B. (2022). Transfer learning for multi-crop leaf disease image classification using convolutional neural network VGG. Artificial Intelligence in Agriculture, 6, 23-33.

[8] Demilie, W. B. (2024). Plant disease detection and classification techniques: a comparative study of the performances. Journal of Big Data, 11(1), 5.

[9] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Machine Learning, vol. 97, pp. 6105–6114, September 2020.

[10] M. Tan and Q. V. Le, "EfficientNetV2: Smaller Models and Faster Training," Computer Vision and Pattern Recognition, vol. 139, pp. 10096–10106, June 2021.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," Computer Vision and Pattern Recognition, vol. 26, pp. 6-8, June 2021.

[12] Hayat, M., Ahmad, N., Nasir, A., & Tariq, Z. A. (2024). Hybrid Deep Learning EfficientNetV2 and Vision Transformer (EffNetV2-ViT) Model for Breast Cancer Histopathological Image Classification. IEEE Access.

[13] Boudouh, N. (2025). Incorporating Deep Learning and Optimization Techniques with Data Augmentation for Improved Image Analysis and Classification (Doctoral dissertation, Université Mohamed Khider (Biskra-Algérie)).

[14] Cheung, W. K., Pakzad, A., Mogulkoc, N., Needleman, S. H., Rangelov, B., Gudmundsson, E., ... & Jacob, J. (2024). Interpolation-split: a data-centric deep learning approach with big interpolated data to boost airway segmentation performance. Journal of big Data, 11(1), 104.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, "Attention Is All You Need," Computation and Language, vol. 30, August 2023.