

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

SC4000: Machine Learning
Project Report

Name	Email	Matric Number
Pu Fanyi	FPU001@e.ntu.edu.sg	U2220175K
Agarwal Anusha	ANUSHA009@e.ntu.edu.sg	U2023105H
Chen Gordon Tian Xiao	GCHEN019@e.ntu.edu.sg	U2140820A
Shan Yi	SH0005YI@e.ntu.edu.sg	U2222846C
Shao Siyang	SHA00054@e.ntu.edu.sg	U2120544F

Course Coordinator: Asst Prof Ke Yiping, Kelly

College of Computing and Data Science
Nanyang Technological University, Singapore

2024/2025 Semester 1

Cassava Leaf Disease Classification

Anusha Agarwal* Fanyi Pu* Gordon Tian Xiao Chen* Siyang Shao* Yi Shan*
College of Computing and Data Science
Nanyang Technological University
Singapore 639798
{ANUSHA009, FPU001, GCHE019, SHA00054, SH0005YI}@e.ntu.edu.sg

Abstract

With the advent of the 2020s AI era, it seems that all efforts have been focused on Transformers. However, we believe ResNet still holds its place. By leveraging a voting mechanism between ResNet and Transformer architectures, we may achieve different results. We propose a new model voting method to add a dimension for voting. In the Kaggle Cassava Leaf Disease Classification competition, our model achieved rank 4/3901 on the Public Benchmark and rank 2/3901 on the Private Benchmark. Our training code can be found in [GitHub](#) and all our models can be found in [Hugging Face](#).

1 Introduction

As outlined in the competition overview [6], cassava is an important crop in Africa, but its yields are significantly affected by diseases. The current disease monitoring primarily relies on visual inspection by agricultural experts, which is labor-intensive and costly. Thus, it is essential to identify common cassava diseases to address this issue. As illustrated in Fig. 1, there are four main types of cassava diseases: Bacterial Blight, Brown Streak Disease, Green Mottle, and Mosaic Disease. Each disease displays distinct characteristics on the leaves, allowing for differentiation from healthy cassava leaves through image classifiers.

Bacterial Blight (Fig. 1a) is characterized by dead leaf spots surrounded by yellow halos [16]. Brown Streak Disease (Fig. 1b) presents as patches of yellow intermingled with the standard green color of the leaves [17]. Green Mottle Disease (Fig. 1c) is identifiable by green spots on the leaves that appear as if they were painted on [18]. Lastly, Mosaic Disease (Fig. 1d) results in symptoms such as discoloration, with leaves showcasing variations of yellow, white, light, and dark shades [3].



(a) Bacterial Blight

(b) Brown Streak

(c) Green Mottle

(d) Mosaic

Figure 1: Cassava Disease Type

The task aims to classify cassava leaf images based on the input data, to determine whether the leaf is healthy or infected by a specific type of virus illustrated before. Specifically, the input consists of image data with a resolution of 800×600 pixels, and the output is a category label indicating the

*Equal contribution. Listing order is alphabetical.

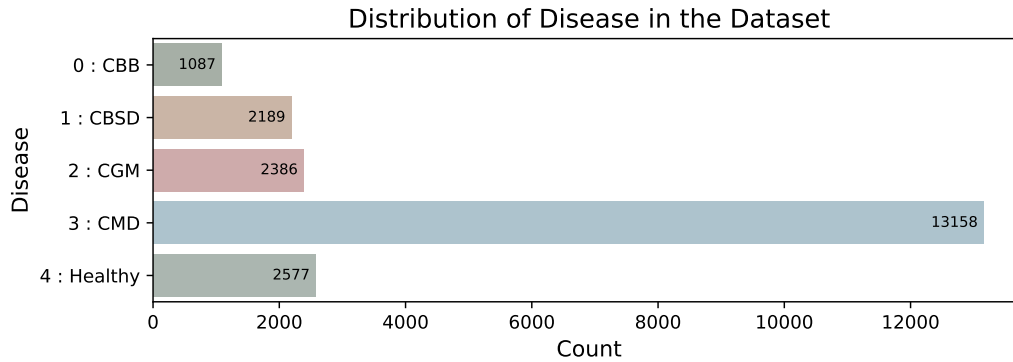


Figure 2: Imbalanced Cassava Disease Label Distribution

leaf’s condition. We categorize diseases from types 0 to 3 as Bacterial Blight through Mosaic and use category 4 to label healthy cassava leaf images.

There are several challenges in the task. First, the original dataset has a significant class imbalance, with many more samples available for the Mosaic disease type than for the others. This imbalance can lead to biased model predictions, where the classifier favors the majority classes and underperforms the minority classes. Second, our analysis shows that a single model struggles to accurately capture and differentiate the characteristics of all virus types due to subtle visual differences among some categories. To address these challenges and improve overall classification performance, it is necessary to use strategies such as data augmentation, re-sampling techniques, and model fusion.

Our dataset comes from the public dataset on Kaggles [6], which includes 21367 labeled images during routine surveys conducted in Uganda. These images were captured by the farmers in their cassava gardens, which show real-world scenarios to distinguish different kinds of diseases.

2 Data Cleaning and Processing

This section details the steps undertaken to prepare the dataset for training and testing the Cassava Leaf Disease Classification model. The dataset was analyzed, cleaned, and processed to ensure proper training and evaluation.

2.1 Exploratory Data Analysis (EDA)

The dataset initially contained 21,397 images categorized into five classes. Analysis of the training dataset revealed the following class distribution shown in Figure 2.

Figure 2 highlighted a significant class imbalance, with Label 3 accounting for over 60% of the dataset, while Label 0 represented only 5%. To address this imbalance and explore its impact on the model’s performance, two dataset preparation strategies were employed: Balanced Data and Imbalanced Data with Train-Validation Split.

2.2 Dataset Preparation

2.2.1 Balancing the Data

To address the class imbalance in the training dataset, a validation set of 500 samples was created by randomly sampling 100 images from each class. The remaining samples were allocated to the training set, resulting in the new distribution shown in Figure 3.

- **Train Set:** 20,897 samples.
- **Validation Set:** 500 samples.
- **Test Set:** 1 sample (reserved for evaluation consistency).

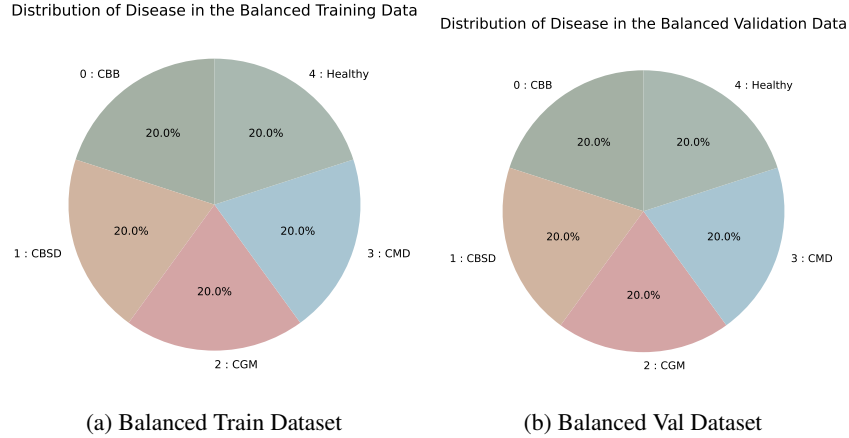


Figure 3: Balanced Cassava Disease Label Distribution

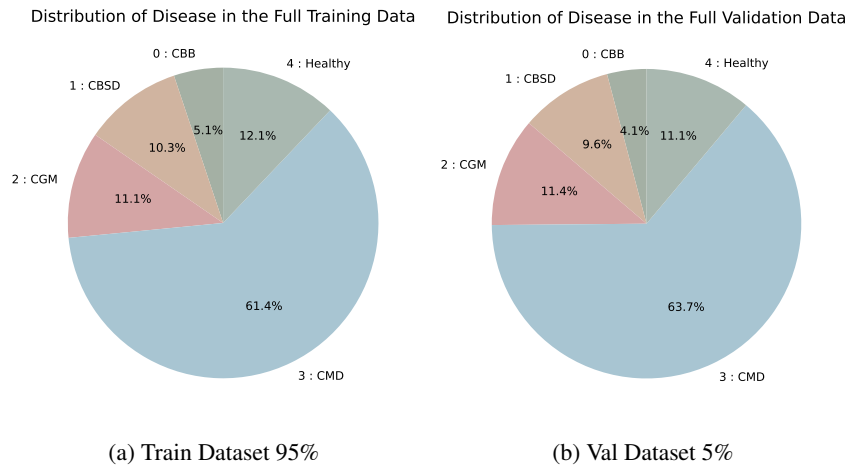


Figure 4: Train-Val Split Cassava Disease Label Distribution for the Full Dataset

The training data was further balanced by down-sampling all classes to the size of the smallest class (**987 samples**). This ensured an equal representation of all classes as seen in Figure[3]

By addressing the imbalance, this approach aims to ensure that the model receives equal exposure to all classes during training. This is particularly beneficial for smaller classes that might otherwise be under-represented.

2.2.2 Train-Validation Split with Imbalanced Data

To maintain the original class distribution, a separate train-validation split was performed without balancing the dataset:

- **Train Set:** 95% of the dataset (**20,327 samples**).
- **Validation Set:** 5% of the dataset (**1,070 samples**).
- **Test Set:** 1 sample.

The distribution is described in Figure 4. This split allows the model to learn from the imbalanced distribution, simulating a real-world scenario where imbalances exist. Maintaining the imbalance is essential to evaluate how the model performs without any balancing techniques applied.

2.3 Data Augmentation

Data augmentation is a crucial technique in computer vision that is used to enhance the diversity of the training dataset without the need to collect additional data. It helps improve the model's generalization ability by simulating real-world variations, such as changes in orientation, scale, lighting, and color. For this analysis, data augmentation was applied to the training dataset to make the model robust to noise and variations in cassava leaf images.

2.3.1 Preprocessing Steps for Training

Focused on increasing dataset variability through augmentation (e.g., random rotations, flips, and color jitter) to improve the model's generalization ability and robustness. The detailed reason we do this is described in Section 3.2.1.

The following augmentation techniques were applied during the training phase:

- **Random Rotation:** The images were randomly rotated by up to 45 degrees to account for variations in orientation.
- **Random Resized Crop:** Images were randomly cropped and resized to ensure consistent input size while preserving variability in the content.
- **Random Horizontal Flip:** Horizontal flipping of images was performed to simulate variations in leaf direction.
- **Random Vertical Flip:** Vertical flipping was applied to enhance the diversity of leaf orientations in the training data.
- **Color Jitter:** Random adjustments were made to the brightness, contrast, saturation, and hue of images to account for variations in lighting and color conditions.
- **ToTensor:** The ToTensor transformation was applied to convert the augmented images into PyTorch tensors, ensuring compatibility with deep learning models.
- **Normalization:** Finally, the images were normalized using the mean and standard deviation values obtained from the pretrained image processor. This scales pixel values to a standard range, aiding in faster convergence during training.

These transformations were applied during the training phase using the following pipeline:

```
1 self.train_transforms = Compose(  
2     [  
3         RandomRotation(degrees=45),  
4         RandomResizedCrop(size),  
5         RandomHorizontalFlip(),  
6         RandomVerticalFlip(),  
7         ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1,  
8             hue=0.1),  
9         ToTensor(),  
10        normalize,  
11    ]  
12 )
```

To better understand the impact of data augmentation, visualization was performed on the transformations applied to a sample image from the training dataset. Each transformation demonstrates how the training data is modified to improve the model's robustness and generalization as seen in Figure 5.

2.3.2 Preprocessing Steps for Validation

In contrast to the training dataset, the validation dataset underwent simpler preprocessing steps designed to preprocess the images uniformly without augmentation. Although each model has its own image processor, we decided to redefine image processing to ensure the correct order of transformations. This ensures that the evaluation reflects how the model performs on real-world, unaltered images:

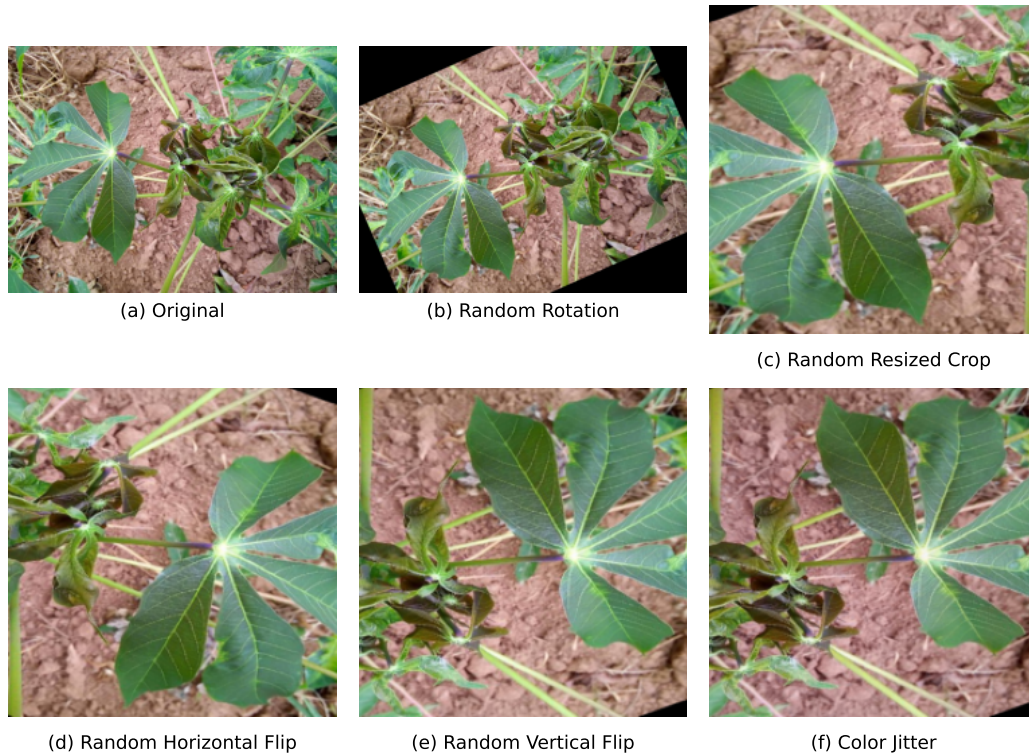


Figure 5: Data Augmentation on Train Sample

- **Resize:** Images were resized to match the input size expected by the model.
- **Center Crop:** A center crop ensured that the main content of the image remained intact while maintaining uniform input dimensions.
- **ToTensor:** Like the training dataset, the validation images were converted into PyTorch tensors to facilitate model compatibility.
- **Normalization:** The same normalization parameters (mean and standard deviation) as used in the training dataset were applied to maintain consistency.

For validation, a simpler transformation pipeline was used to standardize the input images:

```

1 self.val_transforms = Compose(
2     [
3         Resize(size),
4         CenterCrop(size),
5         ToTensor(),
6         normalize,
7     ]
8 )

```

A similar visualization is seen for each step performed in the transformation standardization pipeline for validation data in Figure 6.

Data augmentation during training enhances the model's ability to generalize, while simpler validation preprocessing ensures consistent and unbiased evaluation of the model's performance. This dual approach strikes a balance between robustness and reliability.

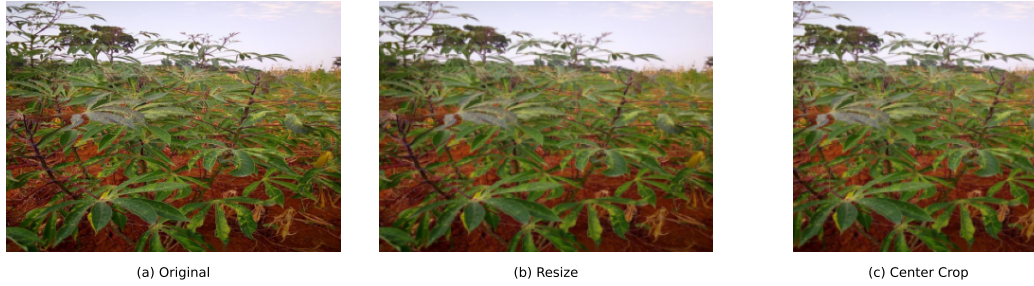


Figure 6: Data Augmentation on Val Sample

2.3.3 Further Improvement: Reducing noise with Segment Anything Model

Most images contain irrelevant background details, such as dirt or leaves from other species, which can introduce localized noise. To address this during preprocessing, we utilize the Segment Anything Model (SAM), an efficient tool for predicting object masks from images and input prompts.

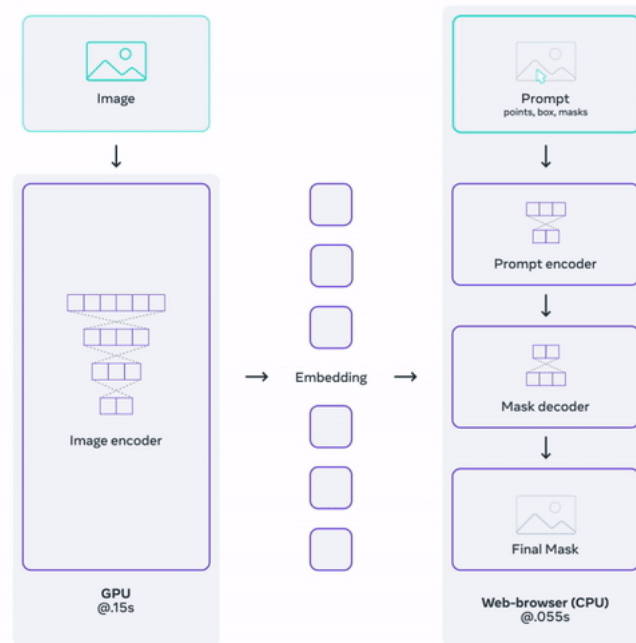


Figure 7: Segment Anything Model Pipeline (Source: Section 3.1c <https://segment-anything.com/>)

Figure 7 illustrates how the Segment Anything Model (SAM) operates. SAM uses an image encoder to extract image embeddings and a prompt encoder to process various input prompts, including point coordinates, bounding boxes, and low-resolution masks. The encoded prompts are combined with image embeddings and passed to a mask decoder, which produces the final object masks.

To enable segmentation guided by natural language prompts, we use GroundingDino in conjunction with SAM. GroundingDino is a text-to-bounding-box model that allows users to input an image and a text prompt. It performs zero-shot text-to-bounding-box object detection, effectively identifying objects of interest in an image based on natural language descriptions. The resulting bounding boxes serve as input prompts for SAM, which refines them into accurate segmentation masks.

Figure 8 illustrates the image before and after processing with SAM. The inverse mask is blacked out to eliminate irrelevant features that contribute to noise. In rare instances where no mask is generated, the entire image is preserved without modifications instead of applying blacking out.

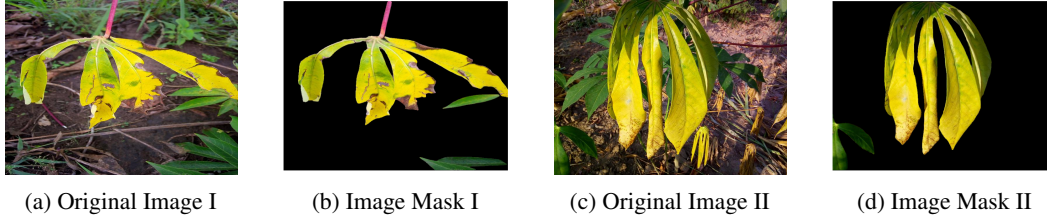


Figure 8: Segmentation Masks obtained using Segment Anything Model (SAM)

3 Model Construction

3.1 Model Selection

Given the high complexity of this classification problem, we aim to conserve computational resources by avoiding retraining the model from scratch. Instead, we perform fine-tuning on multiple models separately and integrate their results to complete the experiments.

VISION TRANSFORMER In recent years, the Transformers [20] architecture has demonstrated remarkable superiority in various vision tasks. Therefore, we plan to use VISION TRANSFORMER (ViT) [5] as one of our training models. We conducted small-scale fine-tuning experiments on models with varying sizes and resolutions using limited data.

Our findings show that, for ViT, increasing model size and resolution makes the model have a better performance. As a result, we choose ViT Large model with resolution 384×384^2 as our pretraining model.

CONVNEXTV2 According to [12], although TRANSFORMERS perform exceptionally well on a wide range of tasks, ResNet [8] can be “modernized” to compete with Transformers in vision tasks. To increase model diversity, we chose CONVNEXTV2 [23], a purely convolution-based model, as the second pretraining model.

After some small-scale experiments, we found that model size seems to have minimal impact on performance. However, higher resolution significantly improved the model’s effectiveness. Based on this, we decided to select the CONVNEXTV2 Base model with resolution 384×384^3 as the final pretraining model, as it offers a smaller model size with a higher resolution. This choice maximizes computational efficiency without compromising model performance.

CROPNET We selected CROPNET, using MOBILENETV3 [9] as the base architecture and has been trained on the *iNaturalist and plants* subset of ImageNet-21K [4]. This provides the model with a wealth of knowledge about plant diseases. We further fine-tuned it to achieve better performance. Additionally, the relatively simple architecture of MOBILENETV3 allowed us to improve training efficiency.

3.2 Loss Function

Cross Entropy loss has been proven to be an excellent loss function for classification tasks. We use cross entropy as the initial loss function with minor modifications.

Label Smoothing Considering that this task involves extensive knowledge in botany and medicine, with high similarity between images of different categories, which often require experts for accurate identification, we propose using label smoothing [19] to reduce the model’s over-confidence and improve its calibration [15]. With label smoothing coefficient α , we define the new loss function

²Obtained from <https://huggingface.co/google/vit-large-patch16-384>.

³Obtained from <https://huggingface.co/facebook/convnextv2-base-22k-384>.

$$\mathcal{L}(\hat{p}, y) = - \left[(1 - \alpha) \log p_y + \alpha \cdot \frac{1}{C} \sum_{c=1}^C \log p_c \right] \quad (1)$$

where \hat{p} is the predicted probabilities for each class, y is the answer, C is the number of classes.

In our actual training process, we have $\alpha = 0.06$.

Balanced Cross Entropy As mentioned in Figure 2, our training data is imbalanced. Although downsampling is a reasonable approach to balancing the dataset, this method leads to the loss of a significant amount of valuable training data. Inspired by [1], we decided to try changing the loss to

$$\mathcal{L}(\hat{p}, y) = -w_y \cdot \left[(1 - \alpha) \log p_y + \alpha \cdot \frac{1}{C} \sum_{i=c}^C \log p_c \right] \quad (2)$$

And the weight vector w is defined as

$$w_c = \frac{n}{C \cdot \sum_{i=1}^n \mathbb{1}_{y_i=c}} \quad (3)$$

where n is the number of training data.

In Equation 2, data with smaller quantities holds a higher weight in the loss calculation, causing the model to pay more attention to this subset during training. However, the drawback of this approach is that it no longer reflects the proportion of training data. If the test set and training set share the same proportions, this modification might actually reduce the model’s performance. This outcome is reasonable since, in real-world scenarios, the prevalence rates of different diseases vary. This modification might not be necessary if both the test set and training set are sampled according to real-world distributions.

We conducted experiments on CONVNEXTV2 using the same parameters. We submitted the trained model and obtained the results shown in Table 1. We found that balancing was not necessary in the actual competition. Therefore, we decided to use the combination of Equation 1 and Full Data for all subsequent experiments.

Loss Function	Training Data	Public Leaderboard ⁴
Equation 1	Full Data	89.50%
Equation 1	Balanced Data	85.25%
Equation 2	Full Data	85.59%

Table 1: CONVNEXTV2 on different loss functions and training data

3.2.1 Mitigating Overfitting

Doing transformations in Section 2.3.1 has effectively prevented overfitting. Figure 9 clearly shows that without transformations, the model had a significant overfit after certain steps. The idea is that transformations make the task harder, so that models can learn more from the training data.

3.2.2 Model Fine-tuning Details

Training Pipeline We built the training pipeline using PyTorch. An abstract class MODEL was used as the interface, with other models inheriting from this class. By standardizing data loading and processing, we significantly reduced the time required to integrate individual models. Hugging Face Trainer [22] and Weights & Biases [21] are integrated for training.

However, the pretraining vector for CROPNET is stored in TensorFlow format. As a result, although it is still integrated into the pipeline, we used Keras to load and train the model.

⁴Although we can see the private leaderboard because we participated after the competition, we do not believe it should be referenced, so we only list the score for public leaderboard.

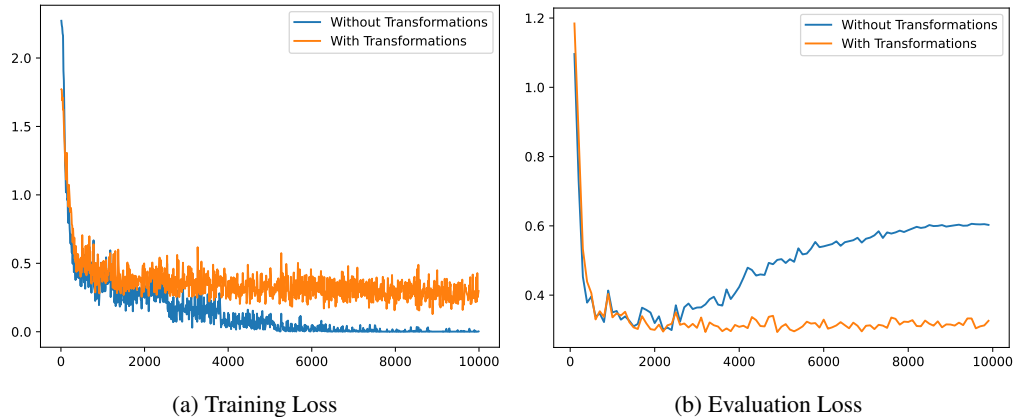


Figure 9: Comparison of ViT Large training process with/without training transformations (first 10k steps only). A significant model overfit can be found without training transformations.

Optimizer We found no evidence that changing the optimizer affected the experimental results. Therefore, we decided to use AdamW [14] for all models. Detailed optimizer config is listed in Appendix A.3.

Learning Rate The learning rate indeed affects the model’s performance. We experimented with various learning rate schedules. We finally decided to use ReduceLRonPlateau for ViT and CROPNET, CosineAnnealingWarmRestarts [13] for CONVNEXTV2. Although we did not observe any effect from adjusting the warmup [7] steps, we still applied warmup with 100 steps to the CONVNEXTV2 and ViT in our final training. We didn’t do warmup on CROPNET. Detailed parameters is listed in Appendix A.1.

Early Stopping For CONVNEXTV2 and ViT, since we can easily monitor and log in real-time through Weights & Biases, we decided not to use Early Stopping. However, due to the difficulty of integrating Weights & Biases with Keras, we can only output logs to the console when training CROPNET, which makes real-time monitoring inconvenient. Therefore, we are considering using Early Stopping for CROPNET.

Low-Rank Adaptation (LoRA) When initially selecting models, we needed to conduct numerous experiments. To enable the simultaneous training of a large number of models with limited GPU resources, we integrated Low-Rank Adaptation (LoRA) [10]. This allowed us to quickly filter out underperforming models during the early stages of model selection, saving significant computational resources. However, during the final training phase, we removed this step, considering that LoRA might compromise the model’s performance.

DDP and Batch Processing Distributed Data Parallel (DDP) [11] is used for our finalize training. To save time for training, we used 32×4 (GPUs) as our batch size.

Model Publishing and Submitting Since the competition requires the submission of inference code rather than results, we have uploaded all the trained models to Hugging Face, which can be accessed by [this link](#). As the inference code needs to be offline, we have stored all the parameters in Kaggle Models for inference.

4 Merging Strategy and Error Analysis

Output Analysis We first analyze the individual model performance of ViT, CROPNET, and CONVNEXTV2 on the validation set.

Fig 10 presents the confusion matrices for the three models: ViT, CROPNET, and CONVNEXTV2. Each matrix illustrates how well the respective model classifies the validation dataset. The horizontal

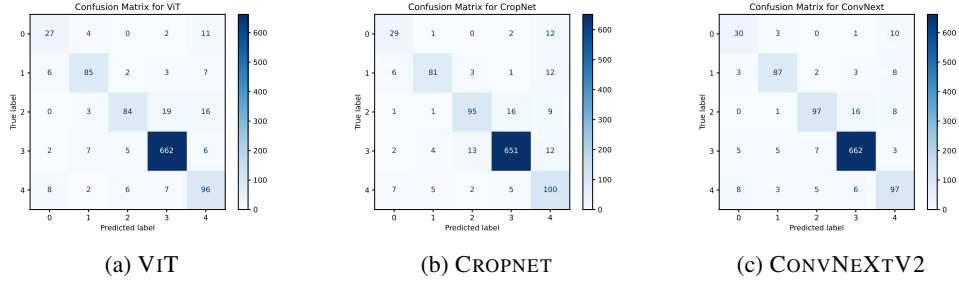


Figure 10: Confusion Matrix for Three Models

axis represents the label predicted by the model, while the vertical axis indicates the actual results. Thus, the diagonal elements indicate the correctly classified images, while the off-diagonal elements represent the misclassified instances.

The evaluation of the three models indicates that the dataset is imbalanced, with class 3 dominating the data. In addition, all models achieved their highest accuracy in class 3. Conversely, class 0 presents the greatest challenge, resulting in the lowest accuracy across all models. Among them, CONVNEXTV2 stands out for its overall accuracy, excelling in classes 0, 1, 2, and 3, though CROPNET slightly performs better in class 4. Notably, CROPNET and ViT demonstrate complementary strengths, with CROPNET performing better in classes 0, 2, and 4, while ViT shows an advantage in classes 1 and 3. Therefore, combining these models together could yield better results.

Hard Voting Hard voting, also known as majority voting, involves each model casting a "vote" for a specific class, with the final outcome determined by the majority of votes from all models. This method does not take into account the probabilities associated with each model's prediction. As demonstrated in Algorithm 1, we implemented hard voting based on the three models discussed previously. In the event of a tie, we have observed that we will default to the result from CONVNEXTV2 as the final outcome.

Algorithm 1: Hard Voting Ensemble with Tie-Breaker

Data: Input data X , Models CONVNEXTV2, CROPNET, ViT

Result: Final prediction y

```

1  $y_{conv} \leftarrow \text{CONVNEXTV2.predict}(X)$ ;
2  $y_{crop} \leftarrow \text{CROPNET.predict}(X)$ ;
3  $y_{vit} \leftarrow \text{ViT.predict}(X)$ ;
4  $votes \leftarrow \{y_{conv}, y_{crop}, y_{vit}\}$ ;
5  $count \leftarrow$  count occurrences of each class in  $votes$ ;
6 if maximum count in  $count$  is unique then
7   |  $y \leftarrow$  class with maximum count in  $count$ ;
8 else
9   |  $y \leftarrow y_{conv}$ ; // Tie-breaker: choose CONVNEXTV2 prediction
10 end
11 return  $y$ 

```

As shown in Fig 12a, the hard voting algorithm allowed us to combine the strengths of each individual model, reducing wrong predictions and enhancing overall accuracy. However, hard voting could also lead to suboptimal results when models vary significantly in accuracy. For example, if a model is indecisive between two classes, using its hard vote may lead to an inaccurate result.

Soft Voting In contrast, soft voting addresses this issue by calculating the probability scores from each model's predictions. As illustrated in Fig 11, we compared the probability differences in cumulative distribution between the ground truth and predicted variables. The results indicate that approximately 20% shows no significant difference, while 40% shows only minor differences. This suggests that soft voting could also yield favorable results.

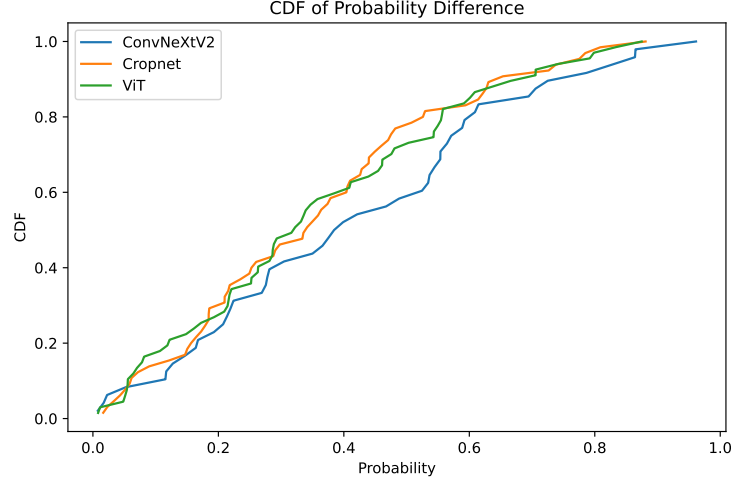


Figure 11: CDF for Probability Difference

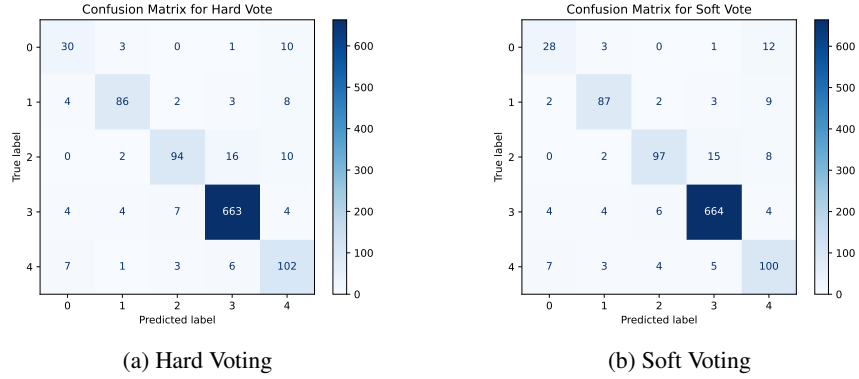


Figure 12: Confusion Matrix for Hard Voting and Soft Voting

We implemented a soft voting algorithm as shown in Algorithm 2. Each model predicts a corresponding probability for each class. Then, we sum these probabilities to calculate an overall probability. Finally, we select the class with the highest overall probability as the final predicted result.

Algorithm 2: Soft Voting Ensemble

Data: Input data X , Models CONVNEXTV2, CROPNET, ViT with probability outputs

Result: Final prediction y

```

12  $P_{\text{conv}} \leftarrow \text{ConvNet.predict\_proba}(X)$ ; // Probability output of ConvNet
13  $P_{\text{crop}} \leftarrow \text{CropNet.predict\_proba}(X)$ ; // Probability output of CropNet
14  $P_{\text{vit}} \leftarrow \text{ViT.predict\_proba}(X)$ ; // Probability output of ViT
15  $P_{\text{total}} \leftarrow P_{\text{conv}} + P_{\text{crop}} + P_{\text{vit}}$ ; // Select class with highest
   summed probability
16 return  $y$ 

```

Conclusion Fig 12 shows the confusion matrix for both hard and soft voting. Both algorithms share the common advantage of effectively improving overall accuracy. However, there are slight differences in performance across specific classes. Compared to soft voting, hard voting yields more accurate predictions for classes 0 and 4 but performs less effectively for classes 1, 2, and 3. Overall, soft voting achieves a higher accuracy rate.

Compared to hard voting, soft voting benefits from the use of more granular probability scores, which reduces sensitivity to wrong predictions, better handles class imbalances in the original data, and offers improved generalization ability. Therefore, we have decided to use soft voting as our chosen method.

Our soft voting method reached 90.79% (rank 17/3901) in the public benchmark.

5 Further Improvement in The Merging Process

Besides Hard Voting and Soft Voting discussed in Section 4, we proposed a voting method called *Soft Voting with Confidence*. The general idea is that if a model generates more extreme probabilities, it should be more confident, and we should assign it greater voting weight. Conversely, if the model's results are ambiguous, we should reduce its weight.

We define a **confidence** $c(p)$ as a function

$$c : [0, 1] \rightarrow [0, \infty) \quad (4)$$

with the following properties:

1. $c(0) = c(1) = 1$;
2. c is continuous in $[0, 1]$;
3. c is a quasiconvex function, which means:

$$\forall x_1, x_2, t \in [0, 1]^3, c(tx_1 + (1-t)x_2) \leq \max\{c(x_1), c(x_2)\} \quad (5)$$

Property 1 suggests that if the model predicts probability 0 or 1, it should have full confidence. Property 2 ensures a slight change in prediction won't affect too much. Property 3 defines the shape of the confidence function.

We define the voting weight

$$w(p) = p \cdot c(p) \quad (6)$$

So the final value is

$$v_{\text{total}}(i) = \sum_{m \in \text{models}} w(p_m(i)) \cdot p_m(i) = \sum_{m \in \text{models}} (p_m(i))^2 \cdot c(p_m(i)) \quad (7)$$

We selected confidence functions

$$c(p) = \frac{\cosh(2p-1)}{\cosh(0)} = \frac{e^{2p-1} + e^{-2p+1}}{e + e^{-1}} \quad (8)$$

in our model.

6 Conclusion

This project focuses on developing models for classifying cassava leaf diseases using advanced machine-learning techniques. It addresses challenges such as data imbalance and noise found in real-world datasets. By employing data augmentation, we combine traditional data pre-processing methods with advanced techniques like the Segment Anything Model. Additionally, we evaluated and integrated results from various model architectures, including ViT, CONVNEXTV2, and CROP-NET. By utilizing a soft voting algorithm, we were able to enhance classification accuracy and generalization.

Finally, our model achieved 90.95% (Rank 4/3901, Top 0.10%) on Public Benchmark and 90.60% (Rank 2/3901, Top 0.05%) on Private Benchmark. Detailed submission results and codes can be accessed in Appendix B.

References

- [1] Yuri Sousa Aurelio, Gustavo Matheus De Almeida, Cristiano Leite de Castro, and Antonio Padua Braga. Learning from imbalanced data sets with weighted cross-entropy function. *Neural processing letters*, 50:1937–1949, 2019.
- [2] Kerim Büyükakyüz. Olor: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*, 2024.
- [3] Patrick Chiza Chikoti, Rabson Mpundu Mulenga, Mathias Tembo, and Peter Sseruwagi. Cassava mosaic disease: a review of a threat to cassava production in zambia. *Journal of Plant Pathology*, 101(3):467–477, 2019.
- [4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.
- [6] ErnestMwebaze, Jesse Mostipak, Joyce, Julia Elliott, and Sohier Dane. Cassava leaf disease classification. <https://kaggle.com/competitions/cassava-leaf-disease-classification>, 2020. Kaggle.
- [7] P Goyal. Accurate, large minibatch sg d: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [9] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.
- [10] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [11] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.
- [12] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.
- [13] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.
- [14] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
- [15] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in neural information processing systems*, 32, 2019.
- [16] University of Wisconsin–Madison. Bacterial blight (uw plant disease diagnostics clinic).

- [17] Basavaprabhu L Patil, James P Legg, Edward Kanju, and Claude M Fauquet. Cassava brown streak disease: a threat to food security in africa. *Journal of General Virology*, 96(5):956–968, 2015.
- [18] Fran Robson, Diane L Hird, and Eric Boa. Cassava brown streak: A deadly virus on the move. *Plant Pathology*, 73(2):221–241, 2024.
- [19] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [21] Weights & Biases. *W&B Docs | Weights & Biases Documentation*, 2024. Accessed: 2024-11-18.
- [22] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.
- [23] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023.

A Training Details

A.1 Learning Rate

Table 2 lists learning rate configs for each model. For ViT and CONVNEXTV2, we have warmup = 500. For CROPNET, we don't have warmup steps.

Model	Learning Rate ⁵	Scheduler	Details
ViT	10^{-4}	ReduceLROnPlateau	$\left\{ \begin{array}{l} \text{patience} = 10 \\ \text{factor} = 0.5 \\ \text{min_lr} = 5 \times 10^{-6} \end{array} \right.$
CONVNEXTV2	10^{-4}	CosineAnnealingWarmRestarts	num_cycles = 0.5
CROPNET	10^{-5}	ReduceLROnPlateau	$\left\{ \begin{array}{l} \text{patience} = 5 \\ \text{factor} = 0.5 \\ \text{min_lr} = 10^{-8} \end{array} \right.$

Table 2: Learning Rates Details for Each Models

A.2 LoRA Cofig

In model selection period, we performed low-rank adaptation (LoRA) [10] for all experiments. We used the same LoRA config for all models to achieve the fairness.

We applied LoRA on all linear layers. Low rank matrices A and B are generated with the same method as in [2]. The rank matrices are set to 8, and the α parameter for LoRA scaling is set to 8.

A.3 Optimizer Config

As section 3.2.2 mentioned, we found no evidence that changing the optimizer affected the experimental results. So we use AdamW with hyperparameter $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ for all training processes.

B Submission Details

We submitted our results to [Kaggle](#).

Our final submission code can be found [here](#).

Our final submission achieved 90.95% (Rank 4/3901, Top 0.10%) on Public Benchmark and 90.60% (Rank 2/3901, Top 0.05%) on Private Benchmark, shown in Figure 13, 14 and 15.


Submission and Description	Private Score	Public Score	Selected
 exp-convnextv2-base-x-vit-large-x-mobile-net - Version 8 Succeeded (after deadline) · 5h ago	0.9060	0.9095	<input type="checkbox"/>

Figure 13: Submission Results

⁵Initial learning rate, do not include warmup. So the actual Initial learning rate is $\frac{\text{Learning Rate}}{\text{Warmup Steps}}$.





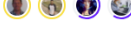



#	Team	Members	Score	Entries	Last	Solution
1	golddiggaz		0.9152	131	4y	
2	Overfit?		0.9100	274	4y	
3	High Hopes!		0.9100	431	4y	
4	Atanas Atanasov		0.9091	224	4y	
5	NaN		0.9091	113	4y	

Figure 14: Public Leaderboard












#	△	Team	Members	Score	Entries	Last	Solution
1	—	golddiggaz		 0.9132	131	4y	
2	▲ 658	Devon Stanfield		 0.9043	3	4y	
3	▲ 26	T0m		 0.9028	244	4y	
4	▲ 182	treesky		 0.9020	16	4y	

Figure 15: Private Leaderboard