Contents

1 Data Preprocessing **1**

2 Training **1**
2.1 Transfer Learning 1
2.2 Fine-Tuning 2

3 Learning Rate Scheduler and Optimizer **2**
3.1 Analysis of the Optimizer 2
3.2 Learning Scheduler 2

4 Augmentation **3**
4.1 RandAugment 3
4.2 Test Time Augmentation 3

5 Performance assessment and further optimization **3**

1 Data Preprocessing

The challenge involved designing and implementing a binary classifier using a deep learning architecture to distinguish between images of healthy and unhealthy leaves. The initial dataset comprised 5200 96x96 RGB images, as illustrated in Figure (2). Two groups of outliers, identified, through t-SNE algorithm, as misleading for the task, were removed, and duplicate occurrences, lacking new information for training, were also eliminated. The refined dataset consisted of 4850 images, with 3060 healthy leaves and 1760 unhealthy ones.
To address the imbalance, over-sampling was employed, generating new samples for the minority class, a proven effective technique [1]. A 90-degree rotation was applied to 1300 randomly selected samples from the unhealthy class, removing the class imbalance.

Figure 1: Random sample of 4 images,the first two images represent the two groups of outliers

2 Training

The classification task at hand, centered on discriminating between healthy and unhealthy leaves, highlighted the challenge of using a limited dataset for robust model training. Designing an architecture for effective feature extraction from scratch was incredibly demanding. Thus, adopting transfer learning proved optimal, resulting in satisfactory performance soon.

2.1 Transfer Learning

Various ImageNet pre-trained architectures were systematically evaluated. Emphasizing efficiency, we focused on smaller models within specific families, including MobileNet [2], EfficientNet [3], and ConvNeXt [4]. While larger models in these families offer superior classification, opting for smaller counterparts that allow for reduced training time, prioritizing achieving faster results.
We achieved optimal performance using the ConvNeXt family of state-of-the-art convolutional neural network architectures, specifically with ConvNeXtLarge and ConvNeXtXLarge [4]. Drawing inspiration from the Vision Transformer (ViT), ConvNeXt ingeniously integrates ViT's attention mechanisms and global context awareness into a convolutional framework. This innovative design optimizes feature learning, resulting in increased performance for our classification task [4].
In the implemented Transfer Learning, the architecture of our classifier incorporated a Softmax activation function at the output layer. In the hidden layers, we chose the Swish activation function, a decision rooted in its documented superiority over ReLU in various frameworks [5].
A Global Average Pooling (GAP) layer was inserted between the feature extraction network and the classifier, motivated by its capability of preserving spatial information, reducing parameters, preventing overfitting, and introducing translation invariance [6]. To mitigate the risk of overfitting our training set, we strategically inserted a batch normalization layer [7]. While dropout was considered, our experiments favored batch normalization: the combination of dropout and subsequent data augmentation posed excessive learning challenges, while normalization effectively countered overfitting, while ensuring a smooth learning process.
For the optimal training of our classifier, we employed several strategic approaches:

- AdamW optimizer, an extension of the classic Adam optimizer, was utilized. This variant incorporates decoupled weight decay regularization, a technique known to enhance the generalization performance [8]. The optimizer operated with a learning rate of 1×10^{-3} and a weight decay of 4×10^{-3}.
- Categorical Cross-Entropy Loss, suited for the binary classification task.
- To ensure a better understanding of the model's progress, the accuracy and loss trends were dynamically plotted during training, providing a real-time visual representation.
- A LearningRateScheduler was implemented for an exponential decay of the learning rate. This dynamic adjustment gradually reduced the step size during training, promoting efficient convergence in the early stages and fine-tuning in later epochs [9].
- Early Stopping, a preventive measure against overfitting, halted training when the model's performance on the validation set plateaued or degraded, ensuring optimal generalization with minimal training effort.

2.2 Fine-Tuning

To tailor the pre-trained model for our specific task and optimize the learned features, fine-tuning emerged as the most effective solution. To achieve this, the first half of the imported convolutional network was designated as trainable, while the remaining layers were kept frozen, so that only the fully-connected part and the newly unfrozen layers were subjected to training.
The same procedure was repeated unfreezing also the remaining layers, reaching higher and higher accuracy on the validation set.

Table 1: ConvNeXtXLarge model evaluation results

	val_loss	**val_accuracy (%)**	**train_loss**	**train_accuracy (%)**
Transfer Learning	$\mathbf{1.744} \times 10^{-1}$	**94.44**	5.285×10^{-2}	98.03
Fine Tuning	$\mathbf{1.227} \times 10^{-1}$	**97.06**	9.171×10^{-3}	99.73

3 Learning Rate Scheduler and Optimizer

This section explores the analysis of various optimizers and their impact on the performance and run-times of different models. These choices were meticulously considered to optimize the models' overall efficiency. Employing an architecture with a non-optimal fully convolutional section, different from the identified final choice, it was trained for a reduced number of epochs. The focus was on identifying the most effective optimization algorithm rather than determining the optimal architecture for the given problem.

3.1 Analysis of the Optimizer

In the course of the development efforts, the efficacy of four key optimizers—Experimental Stochastic Gradient Descent, Adam, AdamW, and Lion—was diligently explored, with the primary focus on optimizing the ConvNeXtLarge application. After several iterations, the most promising architecture was identified, leveraging 'Adaptive Moment Estimation' (Adam) as a baseline for subsequent evaluations. The model was systematically tested with all available optimization algorithms to determine the most suitable one for the specific problem.
Upon scrutinizing the results on the validation set, a limitation in the model's ability to generalize the problem was observed. The Adam optimizer, despite its adaptive learning rate adjustments for each network weight, exhibited a slow convergence to global/local optima. Consequently, a transition was made to Experimental Stochastic Gradient Descent. To address local optima challenges, parameters were fine-tuned, with the momentum component set to $\beta = 0.9$, and the batch size doubled from 16 to 34. While these adjustments led to faster convergence, the validation accuracy remained below the desired threshold. Our exploration extended to two widely adopted optimizers: AdamW and Lion. For consistency in evaluation, we adjusted hyperparameters, increasing the batch size by eight and scaling weight decay/learning rate by a factor of five for Lion. Following rigorous testing, we opted for AdamW over Lion. The decision hinged on the observation that Lion's performance gains diminish when subjected to strong augmentation,which was extensively integrated into the task solution. [Citations: [10], [11]]
The outcomes of our analysis are presented in Table 2.

	val_loss	**val_accuracy (%)**	**train_loss**	**train_accuracy (%)**
Adam	$\mathbf{4.470} \times 10^{-1}$	**79.2**	4.044×10^{-1}	81.49
Experimental SGD	$\mathbf{4.396} \times 10^{-1}$	**81.70**	4.076×10^{-1}	80.46
Lion	$\mathbf{3.663} \times 10^{-1}$	**83.17**	3.008×10^{-1}	86.33
AdamW	$\mathbf{3.949} \times 10^{-1}$	**81.4**	3.717×10^{-1}	83.33

Table 2: Assessment of Optimizers Performances in Transfer Learning with ConvNeXtLarge

3.2 Learning Scheduler

To enhance test set accuracy and improve stability, generalization, and convergence, an exponential learning rate decay strategy was adopted[9]. The Learning Scheduler class was configured with varying parameters at different development phases. During Transfer Learning,the learning rate decay was initiated after 60 epochs with a rate of 0.008. Subsequently, in the Fine-tuning phase of the complete architecture, more pronounced decay rate (ranging

from 0.5 to 1) commencing near the 10-15 epoch mark was set. The implementation of this approach yielded notable improvements in the overall performance of ConvNextLarge, resulting in an increase in test set accuracy by 1.5-2%.

4 Augmentation

Obtaining satisfactory performance with a limited dataset is particularly challenging in practical applications [12]. To address this issue, image augmentation has been confirmed to be an effective and efficient strategy [13], [14]. A variety of augmentation techniques were implemented to let the model generalize better since not all types of augmentations that are present in the literature can be applied: some can alter the image in a way that the task of classifying healthy and unhealthy leaves becomes unfeasible. For example, CutOut augmentation [15] could cut the unhealthy bit of the leaf out, making the image of an unhealthy labeled leaf show a healthy leaf. CutMix and MixUp augmentation techniques [16] could not be used because by adding bits of the image of an unhealthy leaf to the image an healthy one would make the former image show an actually unhealthy leaf. Furthermore the mixed label of an half-healthy leaf does not make sense: the presence of a defect in the image of leaf would make that image show a clear unhealthy leaf.

4.1 RandAugment

RandAugment has been shown to provide improved image classification results across numerous datasets [17] and implements the following augmentations: identity, autoContrast, equalize, rotate, solarize, color, posterize, contrast, brightness, sharpness, shear-x, shear-y, translate-x, translate-y. A number of augmentations per image can be selected and by setting the rate parameter of the keras_cv implementation of RandAugment at 10/11 in our implementation, the behavior is identical to sampling an Identity augmentation 10/11th of the time as implemented in [17]. RandAugment made the pretrained architectures MobileNet [2], EfficientNet [3], and ConvNeXt [4] improve accuracy performances by at least 5% on the test set.
The keras_cv implementation of the RandAugment however is intended as preprocess layer: the "training" attribute is not present, making it impossible to switch off at test-time. By performing RandAugment inside the declaration of the model made the predictions of random augmented images also on test time. To tackle this problem the train_step() method in the Model class was overridden to selectively activate the RandAugment preprocessing layer only during training, rather than during validation or testing. This ensures that the RandAugment layer is invoked externally to the Model, but consistently applies its random transformations each time an image is fed into the model during training. By customizing the train_step() behavior the RandAugment functionality is integrated seamlessly into the training process.

4.2 Test Time Augmentation

It is possible to apply image augmentation also at test-time: Test-Time Augmentation (TTA) is a powerful heuristic that produce averaged output on different augmented images. It has been proven that the expected error of the TTA is less than or equal to the average error of a model without [18]. TTA was implemented in the submissions using 11 different geometric tranformations such as rotation and flip which made the error on the validation split consistent with the error on the hidden test set.

5 Performance assessment and further optimization

In evaluating the performance of a binary classifier with continuous output, akin to the posterior probabilities generated by models such as Howard et al.'s MobileNets (2017), Tan et al.'s EfficientNet (2020), and Liu et al.'s ConvNet (2022) [2–4], the Receiver Operating Characteristic (ROC) curve can be used as an assessment tool across the potential thresholds for the predictor variable [19]. The Area Under the Curve (AUC) of the ROC curve stands as a quantification of the classifier's overall ability to distinguish between the two outcomes. In this study, the ROC curve was generated using predictions derived from the validation dataset, showcasing nearly perfect results. Leveraging the ROC curve analysis, the optimal threshold ($t = 0.4407$) was computed (2) and integrated into the finalized model for predictions on the test dataset making the model ConvNextLarge-based model's error in validation set go from **0.9673** to **0.969**.

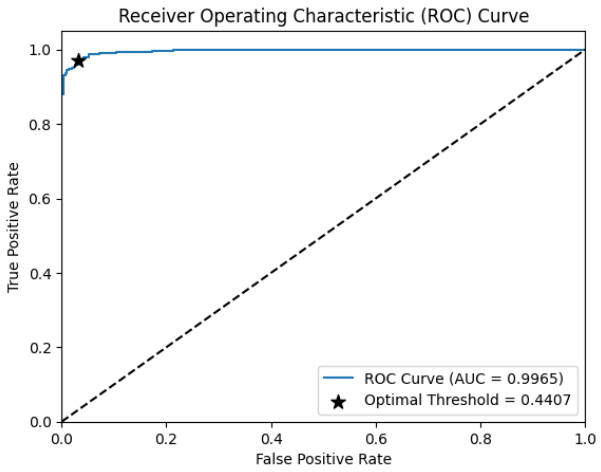

Figure 2: ROC curve with AUC and the optimal threshold using ConvNextLarge

References

[1] Ding H. et al. *KA-Ensemble: towards imbalanced image classification ensembling under-sampling and over-sampling*. 2020. DOI: https://doi.org/10.1007/s11042-019-07856-y.

[2] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[3] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[4] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV].

[5] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[6] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2014. arXiv: 1312.4400 [cs.NE].

[7] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[8] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[9] Sanjeev Arora Zhiyuan Li. *An Exponential Learning Rate Schedule for Deep Learning*. URL: https://doi.org/10.48550/arXiv.1910.07454.

[10] Frank Hutter Ilya Loshchilov. *Decoupled Weight Decay Regularization*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[11] Xiangning Chen et al. *Symbolic Discovery of Optimization Algorithms*. 2023. arXiv: 2302.06675 [cs.LG].

[12] Mingle Xu et al. "A Comprehensive Survey of Image Augmentation Techniques for Deep Learning". In: *Pattern Recognition, Volume 137* (2023).

[13] Sebastien C. Wong et al. "Understanding Data Augmentation for Classification: When to Warp?" In: (2016), pp. 1–6. DOI: 10.1109/DICTA.2016.7797091.

[14] Luke Taylor and Geoff Nitschke. "Improving Deep Learning with Generic Data Augmentation". In: (2018), pp. 1542–1547. DOI: 10.1109/SSCI.2018.8628742.

[15] Rajneesh Tiwari, Aritra Sen, and Arindam Banerjee. "Generalization through augmentation in deep neural networks for handwritten character recognition". In: *International Journal of Scientific & Engineering Research* 11.9 (2020).

[16] Lewy D. and Mańdziuk J. "An overview of mixing augmentation methods and augmentation strategies." In: *Artif Intell Rev, 56* (2023). DOI: https://doi.org/10.1007/s10462-022-10227-z.

[17] Ekin D. Cubuk et al. *RandAugment: Practical automated data augmentation with a reduced search space*. 2019. arXiv: 1909.13719 [cs.CV].

[18] Masanari Kimura. "Understanding Test-Time Augmentation". In: *Neural Information Processing*. Ed. by Teddy Mantoro et al. Cham: Springer International Publishing, 2021, pp. 558–569. ISBN: 978-3-030-92185-9.

[19] Jayawant N. Mandrekar. "Receiver Operating Characteristic Curve in Diagnostic Test Assessment". In: *Journal of Thoracic Oncology* 5.9 (2010), pp. 1315–1316. ISSN: 1556-0864. DOI: https://doi.org/10.1097/JTO.0b013e3181ec173d. URL: https://www.sciencedirect.com/science/article/pii/S1556086415306043.