Artificial Neural Networks Challenge Report

Aldo Cumitini, Davide Gesualdi, Gabriele Passoni, Roberto Riva

November 18, 2023

Abstract

Given a dataset of plants, our objective was to build a neural network capable of achieving satisfying accuracy in classifying new images of plants as healthy or unhealthy, using the theory and methods we learned in classes and laboratories. We aim to discuss and illustrate techniques, challenges, and limitations we encountered in the development of the project and to get some insights on how neural network operate and how we could have better improved our model.

1 When the dataset gets dirty

1.1 The rise of outliers

Initially, adhering to the principle of **'better safe than sorry'**, we conducted a thorough inspection of our data using the Matplotlib library. For ease of handling, we chose to download the entire dataset in **PNG** format. While browsing through the dataset, we encountered a number of **unusual images**, notably frequent instances of **atypical "leaves"** that appeared out of place. To address this, we employed an external tool, **AntiTwin**, which enabled us to identify and examine all occurrences of memes and duplicates within the images. As a result, we made sure that irrelevant images like those featuring **Shrek** (Fig. 2a) and **Troll Face** (Fig. 2b) were excluded from our dataset, ensuring that they would not be learned by our neural network. We also removed **duplicate** images to cleanse and refine our dataset further. In Fig. 1 we show some examples of plants images in the dataset. Finally, for the sake of efficiency and ease of future use, we consolidated the cleaned dataset into a new **NPZ** file. In order to ensure full compliance with the terms of the competition, we have developed a second version of the code using a **hash function** instead of AntiTwin.

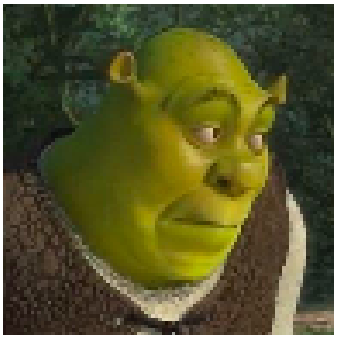
1.2 Augmenting Performances

In our project, we encountered a challenge due to limited data availability: after the cleaning process, our dataset was reduced to a total of only 4,850 images. To tackle this, we turned to **data augmentation**, which meant applying various transformations to images to artificially expand our dataset. We successfully implemented changes like **shifting, flipping, brightening**, and **rotating**, ensuring these did not negatively impact the image labels. However, we steered clear of transformations like contrast and color adjustments. These posed risks of **mislabeling**, particularly crucial for accurately detecting plant health issues where **color consistency** is key. This augmentation was applied only to the training set, allowing us to validate and test our model's real-world accuracy with unaltered data.

(a) Healthy plant image.

(b) Unhealthy plant image.

Figure 1: Example of valid data.



(a) Shrek image.

(b) Troll image.

Figure 2: Example of dataset's outliers.

2 Using pre-trained models

Considering the fairly standard nature of tasks like image classification, often the most efficient strategy is to leverage readily available solutions rather than building from scratch. With this in mind, we turned to a highly benefi-

cial feature of Keras: **keras.applications**. This library offers a variety of pre-trained Convolutional Neural Networks (CNNs), including well-known models like **MobileNet, Xception, EfficientNet** and **ConvNext**, which are the ones we selected for this project. These pre-trained models present us with two distinct methodologies to adapt them to our needs: **Transfer Learning** and **Fine Tuning**. The outcomes of applying these techniques on the four models mentioned before are detailed in Table 1. Notably, the ConvNextLarge model demonstrated superior performance in our experiments, leading us to select it as the foundational architecture for subsequent experiments.

Model	Accuracy	Precision	Recall	F1
MobileNetV2	0.756	0.759	0.778	0.753
Xception	0.859	0.848	0.854	0.850
EfficientNetV2L	0.909	0.901	0.904	0.903
ConvNeXtLarge	**0.938**	0.938	0.927	0.932

Table 1: Private test set scores for fine-tuning applied to different Keras applications.

3 Optimization techniques

3.1 Hyperparameter tuning

In order to improve our model's performance, we initiated the optimization process by searching for a method to automatically test various values for the model's hyperparameters. **KerasTuner** [1], a general-purpose hyperparameter tuning library, allowed us to find a set of suboptimal hyperparameters. We were aware of the significant time and resources required for this process. Consequently, we opted for setting limited ranges and few training epochs to enable running it on Google Colab and Kaggle, even with **free resource limitations**. KerasTuner supports various search strategies: RandomSearch, GridSearch, Hyperband, BayesianOptimization. We chose **BayesianOptimization**, as it considers the history of the hyperparameters that were tried [2].

3.2 AdamW optimization

L_2-regularization and **weight** decay are equivalent in standard **stochastic gradient descent** (SGD) when rescaled by the learning rate. However, as shown in [3], this equivalence does not hold for adaptive gradient algorithms like **Adam**. One reason why Adam and similar methods might be outperformed by SGD with momentum is that common deep learning libraries typically implement L_2-regularization, not the original weight decay. Adam shows significantly **better generalization** when *decoupling the weight decay from the gradient-based update* rather than using L_2-regularization, as also our results in Table 2 shows. For the following equations we refer to [3] for notation and details. In Adam, L_2-regularization is implemented as follows, where w_t represents the weight decay rate at time t:

$$g_t = \nabla f(\boldsymbol{\theta}_t) + w_t \boldsymbol{\theta}_t,$$

Therefore, the regularization term is added to the cost function which is then derived to calculate the gradients g and thus forces the network to learn smaller weights. Adam keeps track of (exponential moving) averages of the gradient (called the *first moment*, denoted as m) and the square of the gradients (called *raw second moment*, denoted as v). However, if one adds the weight decay term at this point, the moving averages of the gradient and its square (m and v) keep track not only of the gradients of the loss function, but also of the regularization term (α is the learning rate and the parameter β_1 control how quickly the averages decay).

$$\theta_t = \theta_{t-1} - \alpha \frac{\beta_1 \mathrm{m}_{t-1} + (1-\beta_1)(\nabla \mathrm{f}_t + w\theta_{t-1})}{\sqrt{\mathrm{v}_t} + \varepsilon} \quad (1)$$

As you can see in Eq. (1) the weight decay is normalized by $\sqrt{v}$ as well. If the gradient of a certain weight is large (or is changing a lot), the corresponding v is large too and the weight is regularized less than weights with small and slowly changing gradients. This means that L_2-regularization does not work as intended and is not as effective as with SGD, which is why SGD yields models that generalize better and has been used for most state-of-the-art results.
Therefore, in [3], they suggest an improved version of Adam called **AdamW**, where the weight decay is performed only after controlling the parameter-wise step size, as it is possible to see in Eq. (2).

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \left(\alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} + w\boldsymbol{\theta}_{t-1} \right), \quad \forall t. \quad (2)$$

The weight decay or regularization term does not end up in the moving averages and is thus only proportional to the weight itself.
In our code we implemented AdamW just compiling the model by setting as optimizers the instance of Keras' AdamW class.

Optimizer	Accuracy	Precision	Recall	F1
Adam	0.901	0.905	0.881	0.890
AdamW	**0.938**	0.938	0.927	0.932

Table 2: Private test set score for Adam vs AdamW in fine tuning of ConvNeXtLarge.

3.3 Dropout and Batch Normalization

Dropout and **Batch Normalization** (BN) often lead to a worse performance when they are combined together, but cooperate well in recent models. As reported in [4], Dropout **shifts the variance** of a specific neural unit when we transfer the state of that network from training to test. However, BN maintains its **statistical variance**, which is accumulated from the entire learning procedure, in the test phase. The inconsistency of variances in Dropout and BN causes the **unstable numerical behavior** in inference that leads to erroneous predictions finally. Meanwhile, the large feature dimension in modern models further reduces the "variance shift" to bring benefits to the overall performance. We obtained the expected outcomes as summarized in Table 3, reporting the **best perfomance obtained with ConvNeXtLarge** (using dropout rate = 0.5):

Approach	Accuracy	Precision	Recall	F1
Dropout only	0.890	0.898	0.865	0.878
BN only	0.933	0.929	0.925	0.927
Dropout + BN	**0.936**	0.936	0.926	0.930

Table 3: Private test set score for Dropout vs BN vs both in fine tuning of ConvNeXtLarge.

3.4 Model Average Ensemble

Deep neural networks are nonlinear models that learn through a **stochastic training mechanism**. As a result, they are highly flexible and capable of learning complex relationships between variables, approximating any mapping function for given training data. However, this flexibility comes with a drawback: they are **sensitive** to the specifics of the **training data** and **random initialization**, leading to high variance in the final model, especially with **small datasets**. One successful approach to mitigate this high variance issue is **ensemble learning**. We employed a model average ensemble [5], comprising **six** neural networks. These were derived by applying fine-tuning to ConvNeXtLarge, with 20% of the base model layers frozen, and trained with the same shuffled and randomly initialized data. The final prediction is the **average** of all predictions from these models.

The results in Table 4 show that the model average ensemble **surpasses** the performance of each individual model. The tests were conducted on our **Dropout + BN** version of fine tuning on ConvNeXtLarge, with the test set fixed for each model for the purpose of visualization. However the test set was different with respect to the randomly shuffled one involved for the other subsections.

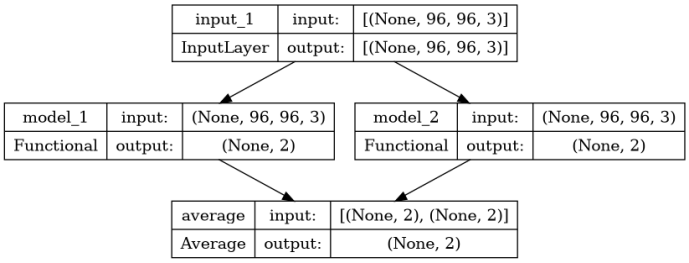


Figure 3: Model Average Ensemble for 2 models.

Model	Accuracy	Precision	Recall	F1
model1	0.932	0.925	0.928	0.927
model2	0.929	0.923	0.927	0.925
model3	0.925	0.921	0.919	0.920
model4	0.932	0.925	0.928	0.927
model5	0.936	0.932	0.929	0.931
model6	0.932	0.929	0.924	0.926
Ensemble	**0.938**	0.935	0.931	0.933

Table 4: Private test set score for Single Models vs Model Average Ensemble performance

Lastly, in the second phase of the challenge, we tested a **10 models version** of the Model Average Ensemble, slightly increasing the accuracy of predictions, as shown in Table 5.

Model	Accuracy	Precision	Recall	F1
6 models	0.897	0.852	0.881	0.866
10 models	**0.905**	0.870	0.881	0.875

Table 5: Second-phase Codalab scores for various versions of the Model Average Ensemble

4 Conclusions and Future Developments

This study has successfully demonstrated the potential of CNNs in classifying images of plants as healthy or unhealthy. Employing various preprocessing techniques like dataset cleaning and data augmentation, alongside the use of pre-trained models from Keras, our project has navigated the challenges inherent in machine learning tasks. For future research, exploring the concept of ensemble learning, but where a combination of different models is used, could provide a more comprehensive and resilient approach to plant health classification. This method could potentially leverage the strengths of various architectures to improve overall predictive performance.

5 Contributions

5.1 Authors

- **Aldo Cumitini**: Tested different EfficientNet architectures and Test Time Augmentation, Latex expert, contributed writing the report.
- **Davide Gesualdi**: Tested ConvNext architectures, model Average Ensemble, AdamW and Hyperparameter tuning, contributed writing the report.
- **Gabriele Passoni**: Tested oversampling and ensemble on VGG and EfficientNet, contributed writing the report.
- **Roberto Riva**: Tested EfficientNetV2L and VGG19, dataset cleaning expert, contributed writing the report.

5.2 Materials

- Github
- Google Drive

5.3 Acknowledgements

We extend our heartfelt gratitude to ChatGPT for its invaluable assistance throughout this project. Its contributions were instrumental in refining our code and enhancing the quality of our manuscript by meticulously correcting grammatical errors in the English language.

References

[1] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[2] Shakti Wadekar. *Hyperparameter Tuning in Keras: TensorFlow 2: With Keras Tuner: RandomSearch, Hyperband, BayesianOptimization*. https://medium.com/swlh/hyperparameter-tuning-in-keras-tensorflow-2-with-keras-tuner-randomsearch-hyperband-3e212647778f. 2021.

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

[4] Xiang Li et al. *Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift*. 2018. arXiv: 1801.05134 [cs.LG].

[5] Van Hiep Phung and Eun Joo Rhee. "A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets". In: *Applied Sciences* 9.21 (2019). ISSN: 2076-3417. DOI: 10.3390/app9214500. URL: https://www.mdpi.com/2076-3417/9/21/4500.