Convolutional Neural Network: Healthy/Unhealthy Leaf Classification

Artificial Neural Networks and Deep Learning – A.Y. 2023/2024

Romina Saljooghi*, Mahdieh Jalali† and Reza Shams‡
M.Sc. Automation and Control Engineering, Politecnico di Milano - Milan, Italy
*Email: *romina.saljooghi@mail.polimi.it, †mahdieh.jalali@mail.polimi.it, ‡reza.shams@mail.polimi.it*
*Student ID: *10888850, †10754004, ‡10811768*
Codalab Group: "Soft98"

1. Introduction

This challenge aimed to train models capable of classifying images of leaves into 2 different classes: healthy and unhealthy. To build the classifier, many choices about the model had to be made. Since the settings that can be tweaked in the model would generate many more configurations than our computational power could handle, we followed an incremental approach. Starting with inspecting our dataset, we initially approached the classification task using a simple CNN model and then tried to improve the performance by exploring 4 different pre-trained models. Many decisions were made in developing each model, which will be further described in the subsequent sections.

2. Data Pipeline

2.1. Data Inspection

Before designing the model, we first analyzed the dataset. The provided dataset consisted of 5200 RGB photos with a resolution of 96x96, along with their corresponding labels. By plotting a selection of sample images from the dataset, we immediately observed several pictures unrelated to the task. To address this issue, we applied the 'Histogram Similarity Check' method following the procedure outlined below:

1) Calculate a reference histogram derived from a known "normal" image, specifically a leaf image. *2*) For each image in the dataset, compute a color histogram. *3*) Determine the intersection between the reference histogram and the histogram of each image. *4*) If the intersection falls below a specified threshold (`similarity_threshold`), categorize the image as an outlier and add it to the `odd_images` list.

2.2. Data Augmentation

Since the dataset we had was not very large, we employed Data Augmentation techniques right from the start in all our models. This approach aided the network in generalizing better and performing more effectively.

We utilized **traditional transformations** such as rotating, zooming, flipping, adjusting brightness, shifting, and shearing. The parameters for these transformations were selected to avoid altering the nature of the items. While the incorporation of all these augmentations slightly enhanced performance, we observed that shear did not have a positive impact.

We also considered using also more **advanced transformations** like CutMix or MixUp in CNN Network. However, since the dataset involved leaves that were more or less similar in each sample, we decided to discard these options. Instead, we focused our attention on developing more complex architectures for the time being.

2.3. Test-Time Augmentation

Once we had our final model (section 6) we also implemented **Test-Time Augmentation (TTA)** to build more robust predictions and reduce variance. The details can be found in Section 7.1.

2.4. Pre-Processing

According to the documentation, the Features Extractors (section 3.1) that we chose to work best with standardized inputs. Therefore, data processing was done according to the `preprocess_input` contained in `tensorflow.keras.applications`.It is worth mentioning that for CNN, we simply apply a rescale to the range [0:1].

3. Convolutional Neural Network

We divided the dataset into a 90:10 train-validation ratio for all models. We opted for categorical classification with CategoricalCrossEntropy as the loss function over binary classification because it offers several advantages. For instance, the Softmax activation function provides more meaningful probabilities and a more interpretable output than sigmoid. Additionally, it includes techniques to handle class imbalance, which expected to leads to more robust models. We used Accuracy as a metric to evaluate the model's performance and also considered F1-score per class to identify its weaknesses.

To begin with, we used a baseline model adopting the classical architecture of modern CNNs. We used 7 blocks of `Conv2D` + `MaxPooling2D` and 3 `Dense` layers with different neurons. To prevent overfitting, we included `BatchNormalization` between Conv2D layers and `Dropou` layers. Instead of using `Flatten` to convert the output of the feature extractor into a one-dimensional tensor, we utilized `Global Max Pooling`. This helped better summarize the features extracted by the model.

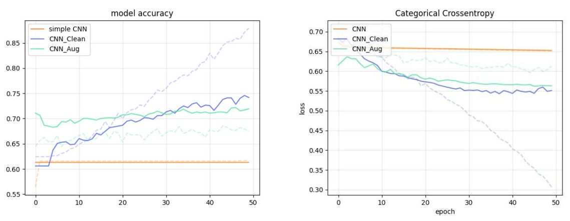

Figure 1. Accuracy and Loss

As can be seen from the tables, after removing outliers from the dataset, the model showed improvement but was overfitted (the accuracy reached from 61.15% to 74.68). By applying data augmentation, we were able to generalize the model, but at the cost of some accuracy.

In addition, we used **Keras Tunner** to automate the exploration of hyperparameter space to enhance the performance of a CNN model. Finally, we obtained a starting accuracy of 81.45% on the Validation set with this method.

Then, we turned to some pre-trained models to perform transfer learning. Given the reduced size of the dataset, we were quite sure this would greatly cut training time and increase performance.

3.1. Feature Extraction

We evaluated the feature extraction potential performance of various pre-trained models by comparing several well-known architectures with a basic fully connected classifier consisting of 512 neurons. Also we are curious about our designed CNN. The simulations were run for 50 epochs.

MobileNet	82.4%
EfficientnetB0	88.53 %
ResNet50	**92.0%**
InceptionV3	88.0 %
CNN	73.9%

According to the results in Table 3.1, the best potential feature extractor model is ResNet50,

4. Transfer Learning

4.1. MobileNet_v2

During the evaluation of about 10 different head architectures mounted on the lightweight MobileNet_v2 , an architecture consisting of `Dropout(0.2)` + `Dense(512)` + `Dropout(0.5)` + `Dense(128)` + `Dense(2)` achieved our best overall accuracy of 82% on the test set. Fine-tuning techniques were applied to the last 50 layers of MobileNet_v2 along with the head layers, which resulted in a validation accuracy of approximately 90%.However, the best performance on the test set remained at 82%.

To discover the best possible accuracy while mitigating overfitting, several alternative scenarios were explored, including testing different activation functions such as relu and Elu, as well as overfitting prevention measures like L2 regularization and dropout in the MobileNet model.

4.2. ResNet-50

Among different versions of resnet architectures, we applied ResNet-50 as a result of the trade-off between complexity and feature extraction ability. A series of experiments were carried out, starting with fully fixed convolutional layers and evolving to revise the features obtained from ResNet-50 by fine-tuning at a low pace, as in each layer of it there are a huge number of parameters that can easily lead us to be overfitted. In the end, after testing various models, the following architecture gave the best ResNet50 accuracy: `base_model(resnet)` + `GAP` + `Dropout(0.2)` + `Dense(128)` + `batch_normalization` + `Dropout(0.5)` + `prediction_layer` in which we used Global Average Pooling (GAP) to reduce the dimension of the `base_model` and last 8 layers of ResNet is fine-tuned. The best accuracy obtained from ResNet50 was 76%.

4.3. EfficientNet

Due to the low ratio between the number of parameters and layers, besides acceptable feature extraction in efficientnetB0, we found it more suitable to overcome overfitting. Among many models tested, the following architecture gave the best results among EfficientNet models: `base_model` + `GAP` + `Dropout(0.2)` + `Dense(512)` + `batch_normalization` + `Dropout(0.3)` + `Dense(256)` + `batch_normalization` + `Dropout(0.1)` + `prediction_layer` Applying preprocessing corresponds to what has been done on imagenet dataset and fine-tuning by freezing 90 layers out of 214, we got 79% accuracy on the test data.

5. Training techniques

5.1. Class imbalance

During the training process, a weighted loss function was used to avoid bias towards any specific class, even though the data did not suffer from a significant imbalance.

6. Ensemble

The last technique we used was the ensemble method in order to improve the following 3 models' prediction performances by combining them: MobileNet, ResNet50, and EfficientNet with 0.82, 0.76, and 0.79 accuracy on test unseen data, respectively. To do this, we removed the specified seed from our code and used two mobilenet, two efficientnet, and one ResNet50 as after considering confusion mtrices. As it was expected, we got an accuracy of 0.86 on the unseen test dataset due to the fact that these models could match each other very well. CNN and Inception were not considered for this purpose.

		MobileNet	ResNet-50	EfficientNet
F1-scores	0	0.87	0.93	0.9
	1	0.76	0.89	0.83
	Acc.	83.40%	91.33%	88%

The ensemble showed an improvement in almost all the metrics, therefore that is the model we chose.

7. Conclusion

7.1. Performance

Finally, we added TTA (section 2.3) to MobileNet, while expecting to see prediction improvement. The result on validation set are:

	Without TTA	With TTA
Test acc.	82.10%	**83.70%**

7.2. Further Developments

It is acknowledged that our workflow could benefit from additional procedures to increase the generalization power of the model. For instance, we could explore more advanced augmentation techniques, experiment with other pre-trained models, try different optimizers, or use self-supervised techniques such as autoencoders.

It can be seen that, although we implemented class weights (section 5.1) to address class imbalance, there was no significant improvement in the F1-scores of the test dataset. However, we can implement further procedures to tackle this issue.

Considering all the relevant factors and our limited resources, we are completely satisfied with the outcome.

7.3. Contribution

The successful completion of this project was a result of the collaborative efforts and synergy of all team members. In-depth analysis of the problem was done together, while a thoughtful division of tasks allowed us to cumulate our focus on specific tasks effectively. Romina dedicated her time to enhancing the baseline CNN, utilizing data processing methods to sample, trying out some experiments with InceptionV3, and effectively multitasking within our team. Reza after conducting analysis on ResNet and EfficientNet, recommended using the Ensemble method for improved results, while Mahdieh demonstrated her expertise by focusing on MobileNet which delivered our best result. Through regular communication and sharing of insights, we were able to integrate our work seamlessly, ensuring a comprehensive exploration of diverse convolutional neural network architectures.