Tree leaf disease classifier

Antonin PAOLI (VR511432)[1]

[1]Master 1 student at the ENSIMAG school in Grenoble, France

Academic year 2023 - 2024

Git link with the code

Computer Vision et Deep Learning

Project evaluated and presented in front of Professor Vittorio MURINO and doctoral student Andrea AVOGARO

Contents

1 MOTIVATION and RATIONALE **3**

2 STATE OF THE ART **3**

3 OBJECTIVES **3**

4 METHODOLOGY **3**
4.1 Datasets . . . 3
4.1.1 General presentation . . . 3
4.1.2 Class balance . . . 3
4.1.3 Class representation . . . 4
4.1.4 Focus on 2 classes . . . 6
4.1.5 Creating sets . . . 7
4.2 Classification . . . 7
4.2.1 Preprocessing . . . 7
4.2.2 Model . . . 7
4.2.3 Loop "Training - Validation" . . . 7
4.2.4 Test . . . 8

5 EXPERIMENTS and RESULTS **8**
5.1 Loss function . . . 8
5.2 Accuracy . . . 9
5.3 Accuracy per class . . . 9

6 CONCLUSIONS **10**

7 BIBLIOGRAPHY **10**

1 MOTIVATION and RATIONALE

As with fauna, flora can also fall sick. But when you're a farmer and your life depends on the life of these trees, what do you do when they fall sick? Unfortunately, there is no universal method for curing them, and there are so many different methods. But before we can take action, we still need to be able to detect what type of disease is affecting them? Some may be obvious, while others are much more complex.

To do this, either you use your own knowledge, which can sometimes be limited, or you call in professionals, who can be expensive. Now imagine that a simple photo could tell us whether a tree is healthy or sick, and if so, what type of disease it is.

My uncle is a farmer, so this subject is very important to me. I saw him rack his brains trying to find solutions to diseases that sometimes led to failure. I'm convinced that this type of solution can be a great help to people in the agricultural sector in finding the most precise treatment for their disease.

2 STATE OF THE ART

Many researchers have already looked into this problem, using various types of model to classify it. This is not an exhaustive list, but in the course of my research I came across 3 different approaches. Some have concentrated on a single supervised learning model such as B. Rajesh and Sujihelen [2020] or Guneet Sachdeva and Kaur [2021]. Others have compared models by studying their performance: Demilie [2024] and thus determine which is the most consistent for this type of classification. And finally, some have focused on pre-processing to improve the work of the model and thus achieve better performance, such as Chilakalapudi and Jayachandran [2024].

3 OBJECTIVES

As we saw in the previous section, a lot of research work has already been carried out, with some interesting results. For my part, through this project, I'd like to find ways of improving the model's performance through upstream image processing. This means modifying our original dataset to create 2 others. The first involves removing the background, but keeping the shape of the leaf. In the second, we keep only the leaf texture (no background and no leaf shape). We will then train and test our model using these different datasets to analyse the different performances.

4 METHODOLOGY

4.1 Datasets

4.1.1 General presentation

The dataset I use is known as the plantvillage dataset. It includes 54,303 images of healthy and unhealthy leaves divided into 38 categories. There are in fact 14 different types of leaf, divided into several different diseases. However, not all plants have a healthy class, such as the orange leaf, for example. If we think back to our motivation, it will be impossible to analyse a photo taken because the leaves of certain trees cannot be classified as healthy.

To increase the size of the dataset, a number of techniques have already been used : image flipping, Gamma correction, noise injection, PCA color augmentation, rotation, and Scaling.

In this section, we will analyse various aspects of our dataset in order to better understand the choices we made when training our model.

4.1.2 Class balance

We can see that some classes are much more represented than others. This will therefore be a parameter to take into account when distributing our sets to avoid unbalanced training.

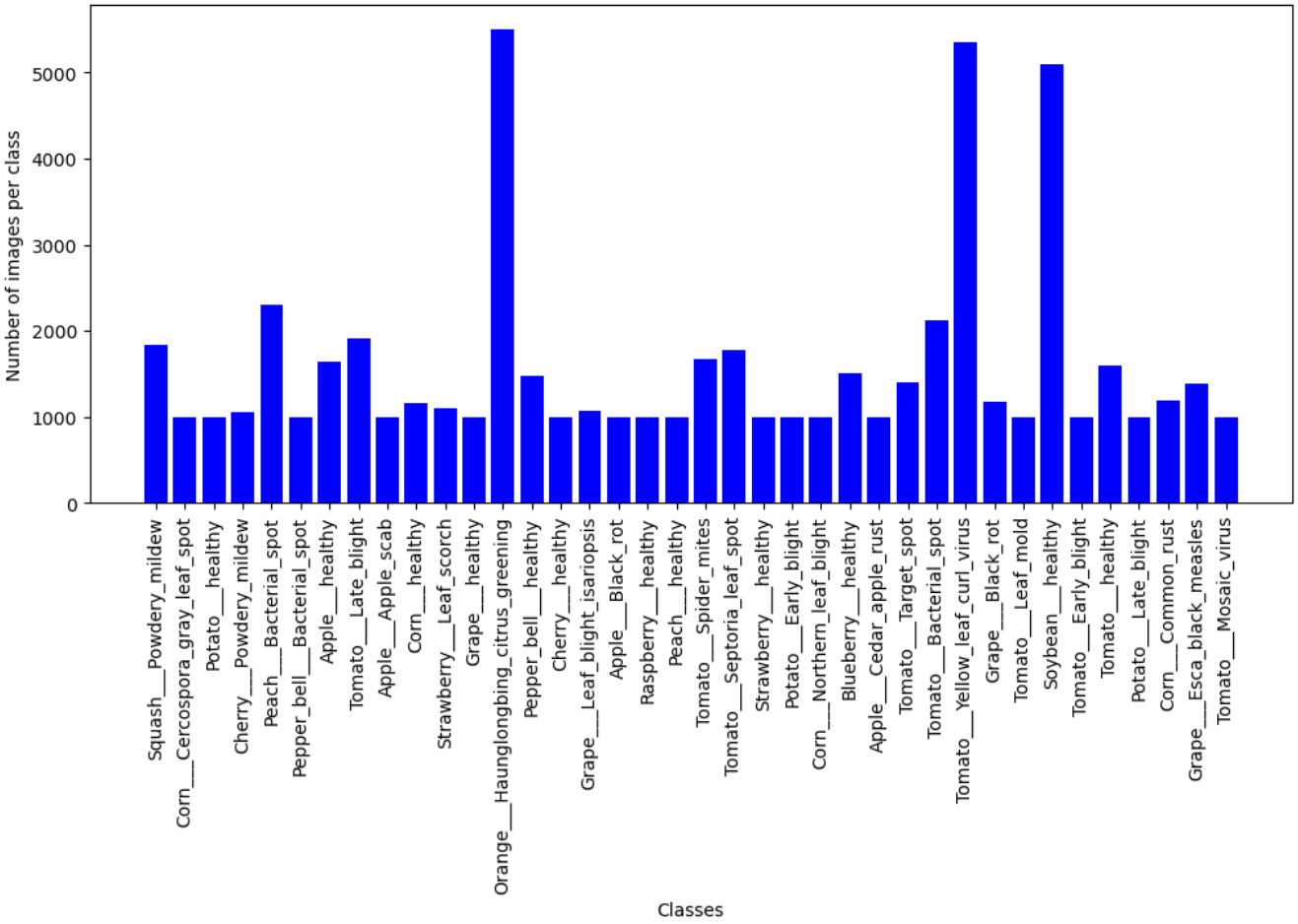


Figure 1: Distribution of images across the 38 classes

4.1.3 Class representation

This part is an analysis of figure 2. I randomly took one image per class. You can see that the background can differ slightly in colour, varying between purple and red. Some are so zoomed in that there is no background at all and one class where the background has already been removed (replaced by black pixels). Then on some of the images you can see shadows, which can make it more difficult for our model to classify. Finally, we can see that the image of the strawberry is not centred on a single leaf but a global photo of the whole plant. This information is important to analyse in order to understand our final results.

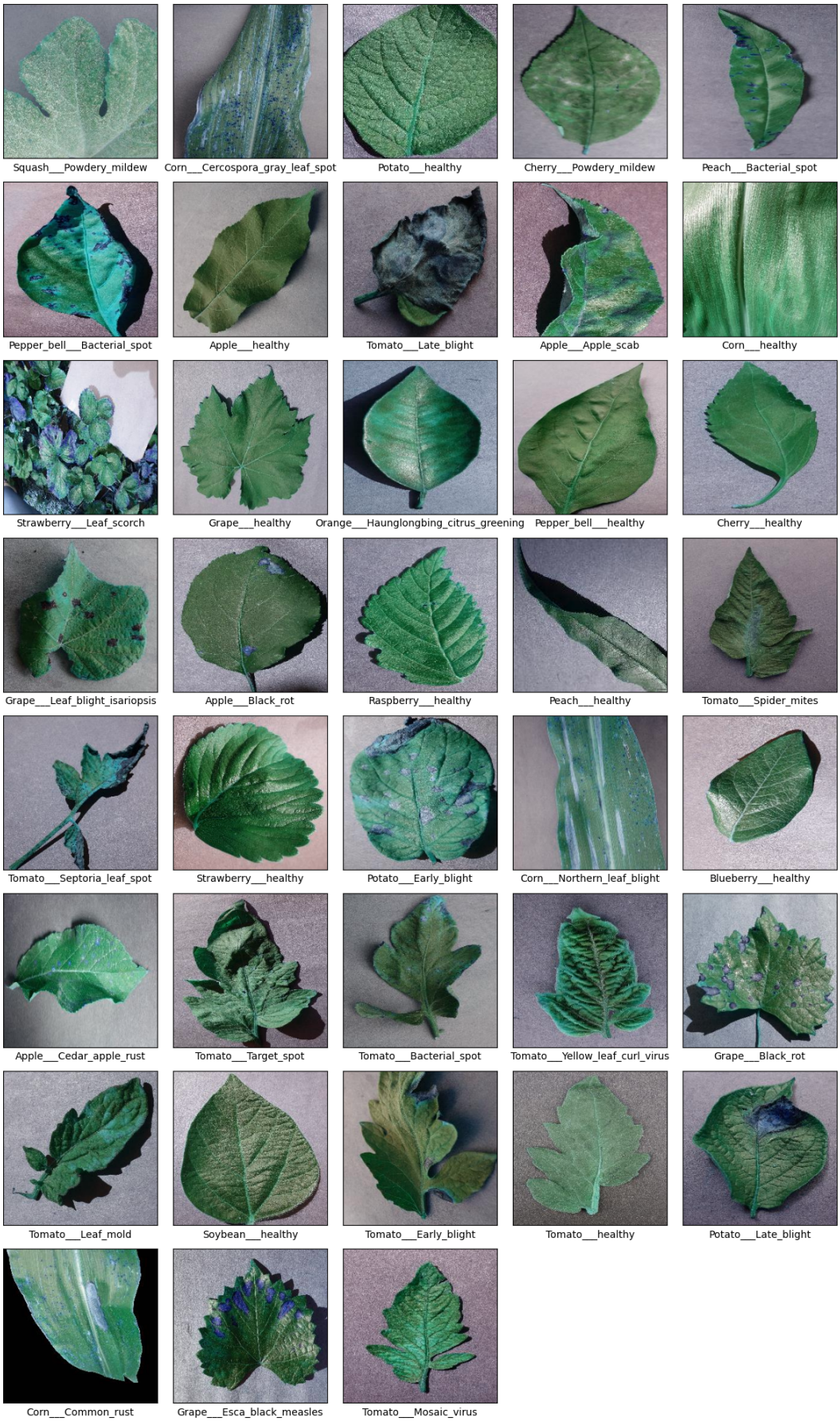


Figure 2: Distribution of images across the 38 classes

4.1.4 Focus on 2 classes

Let's start by looking at all the classes corresponding to tomatoes. You can see that the leaves can have very different shapes and colours. At first I had thought of doing two types of classification, first detecting the type of plant and then the disease. But in the end it would add more complexity to our model in view of these differences.

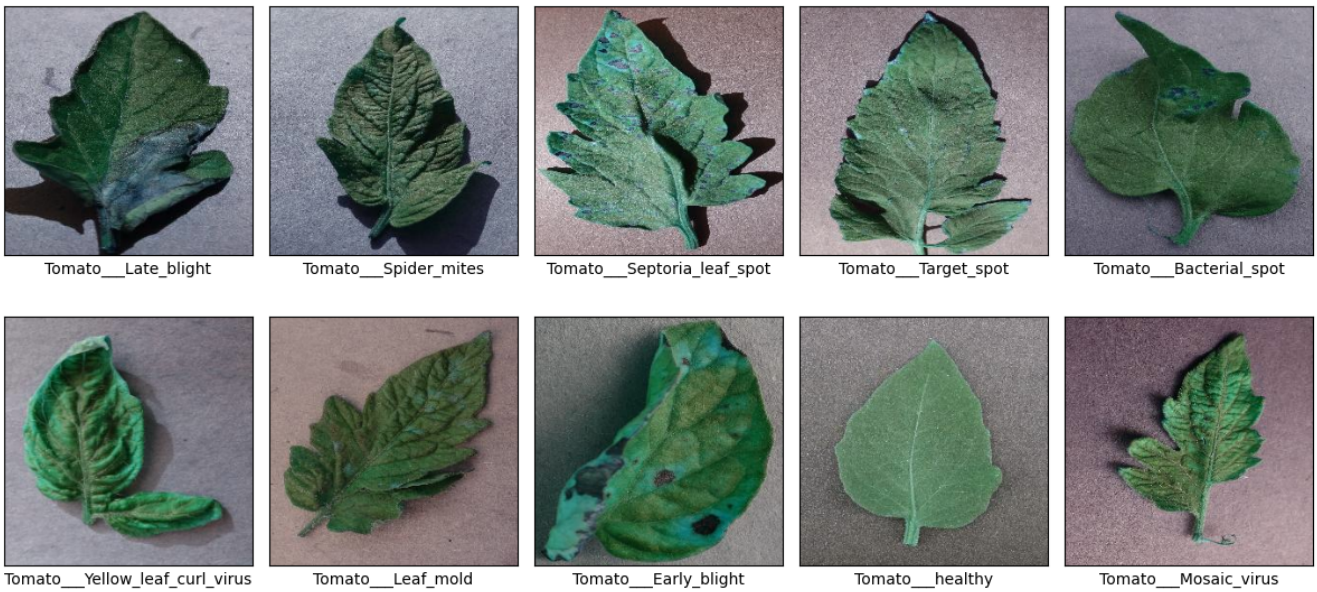


Figure 3: Tomato classes

Now on the other hand, if we take the classes corresponding to potatoes. All the leaves are similar and the diseases are very similar. In this case, it will be easy to detect whether a leaf is diseased or not, but the difference between the diseases may be complicated by the similarity of the spots on the leaves.

Figure 4: Potato classes

To sum up my thinking on this part. We can see that classifying leaves and their diseases will not be an easy task. You can end up with diseases that have very similar characteristics or, in another sense, have the same disease on leaves of very different shapes. These 2 aspects will make classification much more difficult.

4.1.5 Creating sets

To be able to train and test my model, I've separated the dataset into 3 new sets. Firstly, a training set: this is balanced to give a certain ratio of images (70% in my code). If a class is over-presented then fewer ratio of images will be sent to this set. Secondly, a validation set and a test set: These are not balanced but that's not very important in our case. They are made by separating into 2 the remaining images, in other words those that have not been sent in the training set.

I've also coded a function that removes the background and thus creates another division of our initial dataset. Unfortunately the results are not very convincing for some classes. Models should have been used to do the segemntation in a much more optimal way. The results were much more interesting, but unfortunately it took far too long to do it on all the images in our dataset.

4.2 Classification

4.2.1 Preprocessing

Our pre-processing step is fairly straightforward. We simply carry out the necessary steps to prepare our images so that they are in the right format for our model.

For the test set, we define a random area of the image measuring 224x224. This makes it easier to generalise our model. Then we transform them into Pytorch tensors so that they can be used by our model. Then we normalise them to improve the convergence of our model and avoid problems such as overfitting.

For our validation and test set, it's pretty much the same thing. The only difference is that instead of taking a random part of the image, we'll take the central part of our image with a dimension of 224x224. To understand this choice, when a future user takes a photo, they will actually centre the sheet on the image, so this is the most important part to evaluate.

4.2.2 Model

Even if it would have been interesting to compare several models to understand which one is the most powerful, I've decided to choose just one model to better manage my computer's resources. This is the **Resnet18 model**. Firstly, it's known for its interesting performance on tasks such as classification, while not being as deep as its competitors, the resnet50 or the resnet101, which is a good idea in terms of performance/cost. Secondly, as my dataset is small in terms of the number of images out of the number of classes, its already trained version will be an advantage for classification.

4.2.3 Loop "Training - Validation"

You'll see in my code that training and validation are done together in a set of loops (epochs). At first we'll train our model on our training set and calculate our loss function. Then we will call our function to test our model on the validation set. We will also calculate our loss function on this set which has never been seen before by our model. The 2 functions are quite similar! The only difference is that we don't train our model on the validation set but just calculate our metrics. This means that there is no gradient descent during this phase. We loop in this way until we reach the number of epochs initially set or when a stop condition occurs. In other words, either our validation score deteriorates (overfitting), or the improvement is very small, in which case we stop to save resources.

For the loss function, I chose `CrossEntropyLoss()` because it is widely used in the case of multiclass classification and therefore corresponds perfectly to our problem. What's more, it takes into account the imbalance of the classes by weighting them if necessary. I didn't know how to use it properly in my code, but it would have been interesting to put the weightings in the validation and test functions.

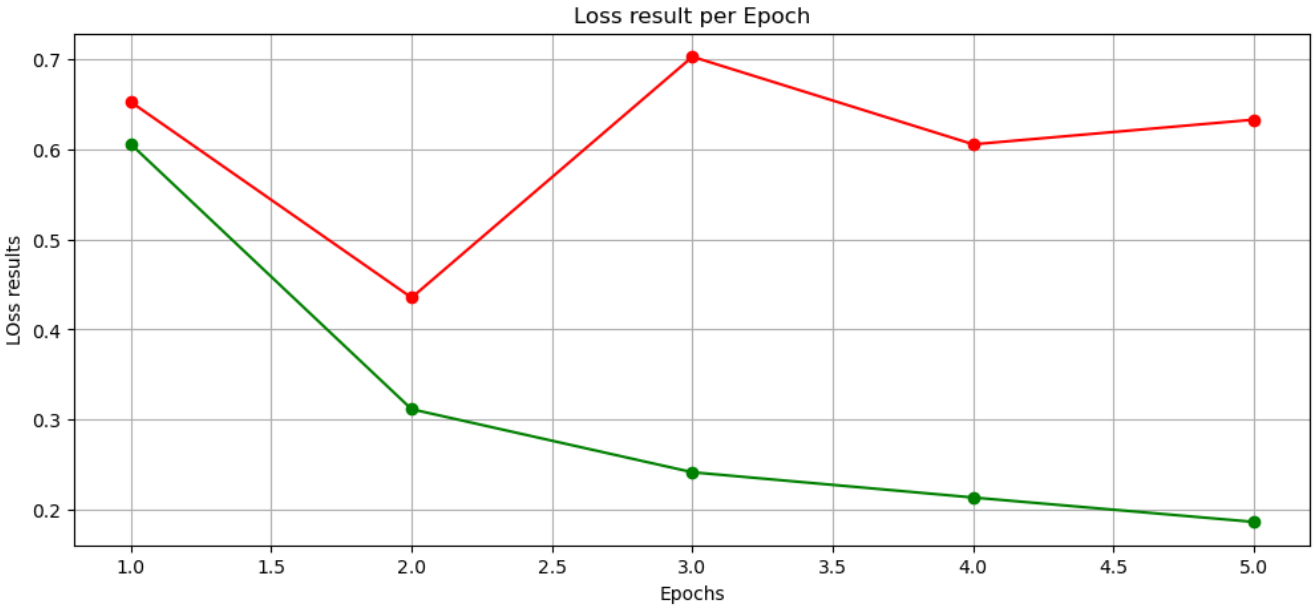
For the optimiser, I chose `optim.Adam()` because it performs well with the resnet18 model and converges fairly quickly, which is interesting in terms of resources. Perhaps it would have been interesting to test different optimisers, such as `optim.AdamW()`, to see which performs better with a problem like ours.

4.2.4 Test

Our test function is very similar to our evaluation function. We calculate additional metrics to better analyse the work done by our model. We will interpret these metrics in the next section. They will enable us to better interpret the final results on the performance of our model.

5 EXPERIMENTS and RESULTS

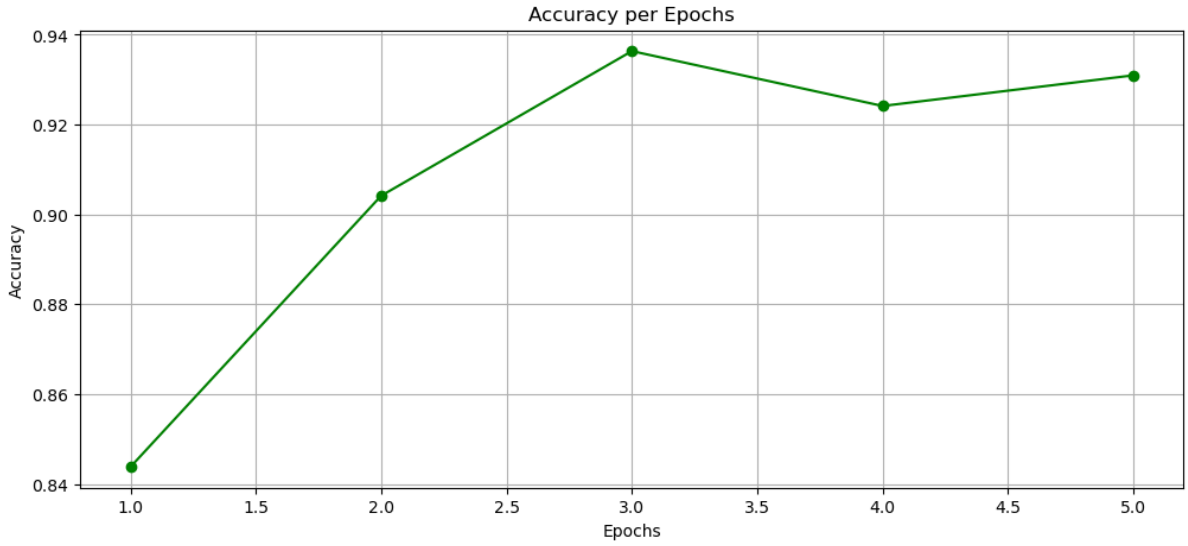
5.1 Loss function



The green curve corresponds to the result of our loss function on our trainings. We can see that as each epoch passes, this value decreases, so we can see that we are improving our model with each iteration. The decrease is less and less significant, so the following iterations would not have drastically improved our model.

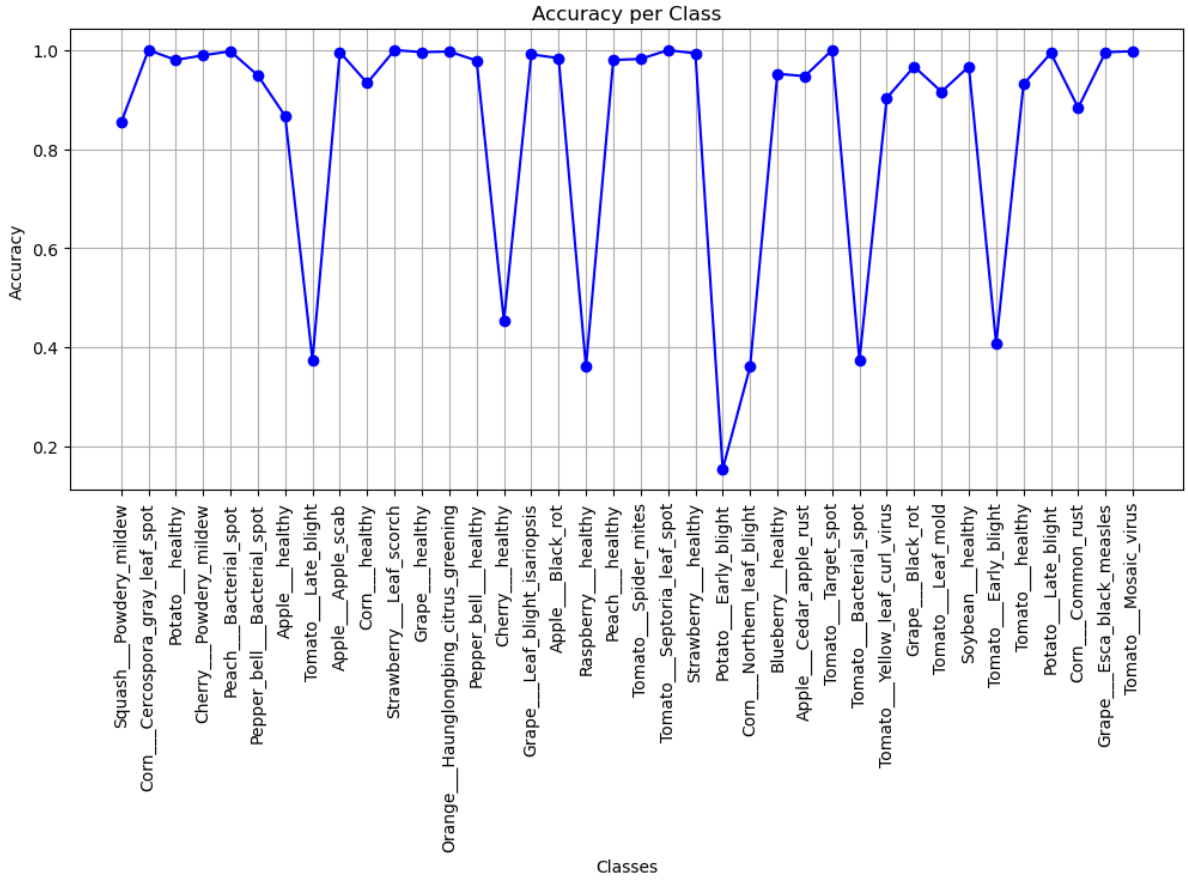
For the red curve, which corresponds to the results of our loss function on the evaluations of the validation set, we can see that it varies between decreases and increases. Normally you'd expect it to go down like the green curve and stop if it starts to go up, which would imply overfitting. However, despite the stop conditions in our function, I've decided to run all 5 epochs because this behaviour may result from the lack of weighting of the classes. In fact, our classes are more than unbalanced in our validation set, which could lead to this curve shape. The weights of each class should have been added as a parameter to the function and it should have been determined whether this fluctuation is the product of this imbalance or of another source. It could also be due to the size of our batches, the way in which our dataset has been increased, the choice of optimiser, etc.

5.2 Accuracy



This curve represents our accuracy over the years. We can see that, globally, we're improving our model's performance over time. As we saw on the previous green curve, our model is improving its performance much more reliably in recent epochs. The choice of stopping here allows us to keep a good classification and save resources.

5.3 Accuracy per class



This graph is quite interesting because it shows how our dataset was evaluated by the model in relation to the observations we made at the beginning of this report. We can see that, globally, the classes obtain good scores. Except for a few classes that lower our final performance. There is quite a big difference between these 2 categories of classes. We can see that the over-represented classes generally score better than the others. It is therefore not these classes that lower the final performance.

Firstly, for the classes with poorer performance, some are coherent, like the tomato classes, which as we said at the beginning have very very different images for the same class, making classification very complex. If you look at the images in the `Tomato__late_blight` class, for example, you can quickly understand the result. Another class that was somewhat expected was the potato classes. The 2 diseased classes are so similar that it's possible that the model classifies all the diseased leaves as `Potato__late_blight` (which could explain a result of 1 for this class) and therefore gets all the leaves corresponding to the `Potato_early_blight` disease wrong.

Secondly, there are others that are much more surprising, such as `Raspberry__healty` and `Cherry__healthy`. Both have images of leaves of a quality quite different from the other classes. And finally, for the `Corn__northern_leaf_blight` class, we can see that it also strongly resembles the 2nd sick class. From this we can make 2 assumptions: either we have the same schema as for the potatoes, or the model is much more efficient in terms of training on images from which we have previously removed the background (using black pixels).

6 CONCLUSIONS

In conclusion, leaf classification is not a trivial task. There are many parameters involved: diseases can look the same, leaves can have similar physical characteristics, a single disease can exist on several very different types of leaf, etc. There could be many social advantages if this project could perform well, but we can see that there has to be a lot of upstream processing before we have a model that performs well in all its classes. On this subject, I said at the beginning of my report that I wanted to test performance on 3 datasets that had been processed differently on the images. In the end, we could only see one class. If we were to make some assumptions about the results we got: the class that had upstream processing with background removal performed well. We can assume that removing the background, while keeping the shape of the sheet and all the details that make it up, could improve performance. However, the class with full images, in other words without any background (`Corn__healthy`) still performs well, but a little less well. We might imagine that training on these images would be optimal if the leaves had very different textures. However, we have seen throughout this project that this is not the case. The shape of the leaf will therefore also be important in this classification, so it's a notion that should be kept in mind in order to improve performance.

7 BIBLIOGRAPHY

References

M. Vishnu Sai Vardhan B. Rajesh and L. Sujihelen. Leaf disease detection and classification by decision tree. *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, 2020. URL `https://ieeexplore.ieee.org/document/9142988/authors#authors`.

Malathi Chilakalapudi and Sheela Jayachandran. Multi-classification of disease induced in plant leaf using chronological flamingo search optimization with transfer learning. *Bilal Alatas*, 2024. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC11042004/`.

Wubetu Barud Demilie. Plant disease detection and classification techniques: a comparative study of the performances. *Journal of Big Data*, 2024. URL `https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00863-9`.

Preeti Singh Guneet Sachdeva and Pardeep Kaur. Plant leaf disease classification using deep convolutional neural network with bayesian learning. *Second International Conference on Aspects of Materials Science and Engineering (ICAMSE 2021)*, 45, 2021. URL `https://www.sciencedirect.com/science/article/abs/pii/S2214785321014115`.