13.6.2024

Plant Recognition

Report #2

Inhalt [tribute to Germany ☺]

Team: Luigi Menale | Bernd Brinkmann | Arif Haidari | Alex Tavkhelidze

mentor: Romain Lesieur

DATASCIENTEST

Introduction

The following activities have been pursued during the modelling phase:

1. Classification of the problem
2. Implementing a machine learning model
3. Implementing a deep learning model
4. Explore how data augmentation can be applied for a deep learning model
5. Apply best practices from transfer learning on this task

These work packages were mutually assigned considering the preferences of the individuals. The activities and achievements are summarized in this report.

1. Classification of the problem

Since the underlying dataset represents the collection of subdirectories each containing image files of certain label/class (defined by the respective names of these subdirectories), we are unequivocally facing a multiclass classification problem.

Although initially thought and sought to combine both image classification and object recognition, the pedagogical scope of the project as well as the restriction of time and computational resources forced our broadened ambition's retreat and led to focusing our entire endeavors on the image classification constituent, which alone manifests itself as a worthy challenge to deal with.

We decided neither to reinvent the wheel and nor to be too clever by half – instead, we followed the well-established best practice [1] and employed the interplay of accuracy and (weighted-average) f1-score, which equally accounts for precision and recall, when choosing the classification metrics for our models.

As a matter of fact, the entire 'sklearn.metrics.classification_report' got scrutinized pretty much in every detail [2], especially for thinly populated classes of the underlying dataset.

2. Classic Machine Learning for Image Classification

In this chapter, we explored traditional machine learning techniques to address image classification problems. A critical aspect of using traditional machine learning in image processing is identifying an optimal feature extraction method. The second key part is selecting (finalizing) the best algorithm to train on these extracted features.

2.1. Image Processing and Feature Extraction

To evaluate the methods, we used the Support Victor Machine (SVM) algorithm to train our features, comparing which method is optimal. Due to processing resource and time constraints, the exploration was limited to a subset of the dataset. The goal was to find the optimal solution and then apply it to the entire dataset (in later chapters). Several image processing methods were experimented with to extract features from the images, including:

I. Histogram of Oriented Gradients (HOG)

This method captures gradient orientation information within an image, achieving the following results:

Table 1: HOG Method Performance

Metric	Apple___Apple_scab	Apple___Black_rot	Apple___Cedar_apple_rust	Apple___healthy
Precision	0.57	0.67	0.75	0.72
Recall	0.57	0.68	0.64	0.82
F1-score	0.57	0.68	0.69	0.76
Accuracy			0.68	

This method is effective in capturing edge and shape information and is robust to small geometric transformations but is sensitive to large variations in illumination and viewpoint. It does not capture colors very well, prompting the introduction of the next method.

II. Color Histograms + Histogram of Oriented Gradients (HOG)

To address the previous method limitation, we combined it with Color Histograms. This method represents the distribution of colors in an image. Combining Color Histograms and HOG, we achieved:

Table 2: Color Histograms + HOG Performance

Metric	Apple___Apple_scab	Apple___Black_rot	Apple___Cedar_apple_rust	Apple___healthy
Precision	0.75	0.75	0.89	0.74
Recall	0.58	0.80	0.80	0.95
F1-score	0.65	0.77	0.84	0.83
Accuracy			0.78	

This method improved feature extraction accuracy by ***10%.*** The addition of color information improves the feature extraction accuracy by capturing color-based differentiation.

III. Local Binary Patterns (LBP) + Histogram of Oriented Gradients (HOG)

LBP is computationally efficient for texture analysis. When combined with HOG, it improves texture representation. However, it is sensitive to noise, which can affect its performance. Combining HOG with LBP, we achieved:

Table 3: LBP + HOG Performance

Metric	Apple___Apple_scab	Apple___Black_rot	Apple___Cedar_apple_rust	Apple___healthy
Precision	0.59	0.73	0.77	0.72
Recall	0.68	0.68	0.68	0.74
F1-score	0.63	0.70	0.73	0.73
Accuracy			0.70	

Compared to Color Histograms, this combination improved accuracy by only 2%, likely due to sensitivity to noise.

IV. Local Binary Patterns (LBP) + HOG + Color Histograms

LBP is significant in feature extraction for their simplicity and efficiency in capturing texture information. It is computationally efficient and robust but LBP can be sensitive to noise (Gaussian Noise and Speckle Noise), which might limit its effectiveness in some scenarios. Combining LBP with other methods like HOG can enhance texture representation in images, although it may not always significantly improve accuracy when combined with other robust methods like HOG and Color Histograms. The combination yielded the same accuracy as the HOG and Color Histogram combination, suggesting that LBP adds no significant benefit. This suggests that LBP does not provide additional useful information beyond what HOG and Color Histograms already capture. The redundancy in feature representation occurs because HOG effectively captures edge information and Color Histograms capture color distribution, leaving little room for LBP to contribute uniquely.

LBP's sensitivity to noise and the redundancy in feature representation limit its effectiveness in improving classification accuracy when combined with more robust methods like HOG and Color Histograms. This illustrates that while LBP is valuable for texture analysis, its advantages are overshadowed by its drawbacks in noisy environments and its limited added value in the presence of already effective features.

Table 4: LBP + HOG + Color Histograms Performance

Metric	Apple___Apple_scab	Apple___Black_rot	Apple___Cedar_apple_rust	Apple___healthy
Precision	0.75	0.75	0.89	0.74
Recall	0.58	0.80	0.80	0.95
F1-score	0.65	0.77	0.84	0.83
Accuracy			0.78	

V. SIFT, SURF and ORB

- SIFT (Scale-Invariant Feature Transform): Detects and describes local features invariant to scale and rotation, making it robust to changes in image size and orientation.

- SURF (Speeded-Up Robust Features): Similar to SIFT but faster, offering better performance for real-time applications.

- ORB (Oriented FAST and Rotated BRIEF): A fast and efficient alternative to SIFT [3] and SURF, suitable for applications requiring speed and efficiency.

Since ORB is the optimal version of SIFT and SURF, so we only use ORB method to extract the features.

This method, with an accuracy of 57%, is not suitable for feature extraction and here is why:

i. Feature Sensitivity: ORB relies on corner detection, missing other critical features needed for our dataset as our type of solving problems not only on edges but colors as well.
ii. Lighting and Texture Variations: ORB struggles with these variations, which are crucial for plant disease detection.
iii. Lack of Color Information: ORB focuses on grayscale images, ignoring essential color differences.

Table 5: ORB Method Performance

Metric	Apple___Apple_scab	Apple___Black_rot	Apple___Cedar_apple_rust	Apple___healthy
Precision	0.51	0.53	0.64	0.61
Recall	0.43	0.64	0.64	0.60
F1-score	0.47	0.58	0.64	0.60
Accuracy			0.57	

VI. Bag of Visual Words (BoVW)

This method uses SIFT descriptors, clustering (e.g., K-means), and a histogram of visual words to represent images. While BoVW can capture complex features, it has significant computational overhead. The calculation was very time-consuming, taking over 9 hours, which makes it inefficient for practical use in this scenario.

VII. Fourier and Wavelet Transform + PCA

Using Fourier and Wavelet Transform methods from the article on Kaggle [4], we initially faced the challenge of high-dimensional feature spaces. Both transforms are powerful for capturing frequency and localized frequency information, respectively. However, they resulted in an enormous number of dimensions, making the computation very expensive. To address this, we applied Principal Component Analysis (PCA) after extracting features to reduce dimensionality. Despite this modification, the results were not as good as some previous methods. The Fourier and Wavelet Transform approach did not perform as well as HOG and Color Histograms, achieving lower accuracy and requiring more computational resources.

Table 6: Fourier and Wavelet Transform + PCA Performance

Metric	Apple___Apple_scab	Apple___Black_rot	Apple___Cedar_apple_rust	Apple___healthy
Precision	0.54	0.68	0.84	0.89
Recall	0.70	0.65	0.72	0.78
F1-score	0.61	0.67	0.77	0.83
Accuracy			0.71	

Conclusion: Optimal Feature Extraction Method

After analysis and experimentation, the combination of Histogram of Oriented Gradients (HOG) and Color Histograms was selected as the optimal method for feature extraction. This combination achieved a notable 78% accuracy using the SVM algorithm. Several factors influenced this choice:

I. Computational Efficiency: HOG and Color Histograms are computationally efficient, providing good results in a relatively short amount of time. This efficiency is crucial for practical applications where computational resources and processing time are limited.

II. Complementary Strengths: HOG captures edge and gradient information, which is essential for detecting the shapes and structures within an image. Color Histograms, on the other hand, capture the distribution of colors, which is particularly important in our case since the color variations help differentiate between different diseases in plant leaves. The combination of these methods ensures

that both structural and color information are effectively utilized, enhancing the overall feature representation.

III. Robustness to Variations: The combination of HOG and Color Histograms proved robust in various scenarios. For instance, diseases on leaves can present similar patterns in grayscale images, making it difficult to differentiate using edge information alone. The color information captured by Color Histograms complements HOG by providing additional discriminative features, which are critical for accurate classification.

IV. Limited Experimentation Scope: Although the limited scope of our experiments (~10 methods) might suggest that other combinations of methods could be explored, the theoretical number of combinations (2^10 = 1024) makes exhaustive testing impractical. Our experiments indicate that the chosen combination provides a balanced trade-off between performance and computational efficiency.

V. Flexibility and Adaptability: While other methods might perform better for specific tasks, the combination of HOG and Color Histograms has shown to be versatile across different methods that we have experimented. This adaptability makes it a reliable choice for our image classification tasks.

VI. Mathematical Insights: HOG works by dividing the image into small connected regions called cells and computing the histogram of gradient directions or edge orientations in these cells. This local gradient information is then combined into a descriptor. Color Histograms, on the other hand, summarize the distribution of colors in an image. The synergy between these two methods leverages both spatial and color information, making the feature extraction process more comprehensive.

In conclusion, the combination of HOG and Color Histograms was chosen for its computational friendliness, robust performance, and complementary strengths in capturing both structural and color information. This approach offers a practical and effective solution for feature extraction, suitable for the specific requirements of our image classification problem involving plant disease detection.

2.2. Model Selection and Training

Various algorithms were considered, but XGBoost (XGBClassifier) was chosen for its superior performance. The algorithms were examined using the optimal feature extraction method. The ranking of algorithms based on accuracy is as follows:

I. XGBClassifier (97%)
II. Support Victor Machine (78%)
III. RandomForestClassifier (75%)
IV. LogisticRegression (71%)

XGBClassifier

The XGBoost (XGBClassifier) algorithm was selected for its superior performance in handling large datasets and its robustness against overfitting. My conclusion in choosing this algorithm is based on both experimental results and extensive review of algorithm documentation. Experiments demonstrated that XGBClassifier achieved the highest accuracy (97%) among all tested algorithms, significantly outperforming Support Victor Machine (SVM), RandomForestClassifier, and LogisticRegression. Additionally, documentation highlights its gradient boosting framework, which optimizes model performance and provides scalability, making it ideal for complex image classification tasks.

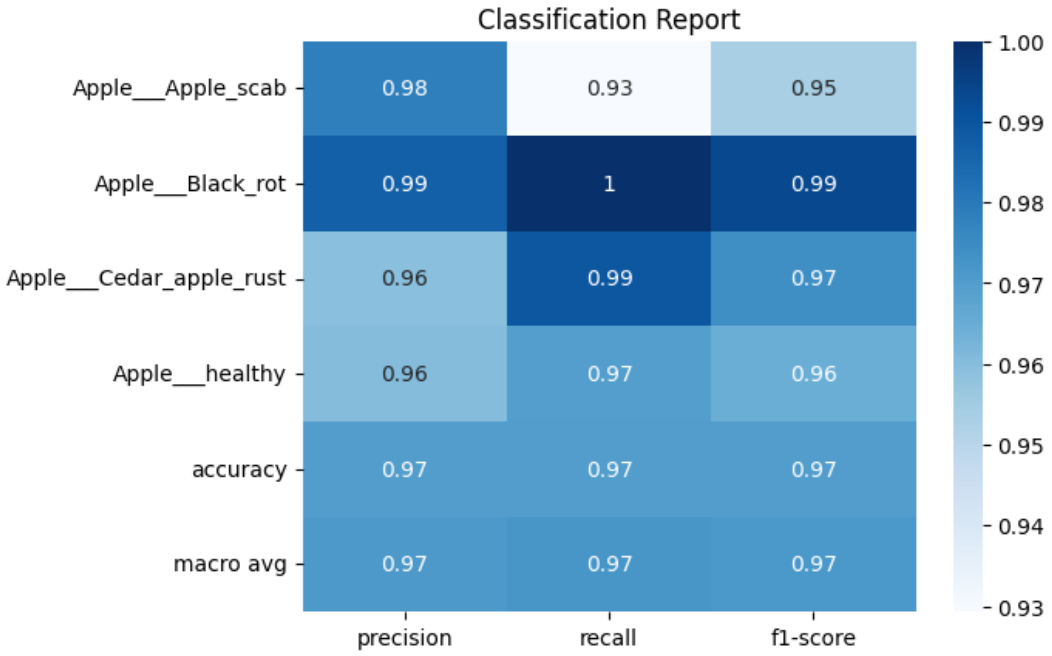

Figure 1: Classification Report

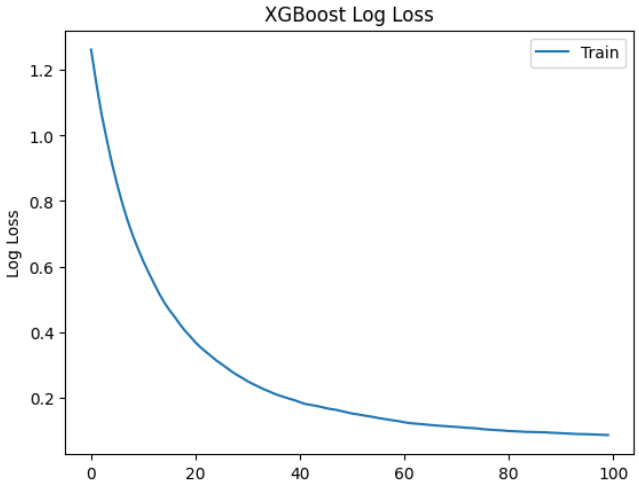

Figure 2: Log Loss function on train dataset

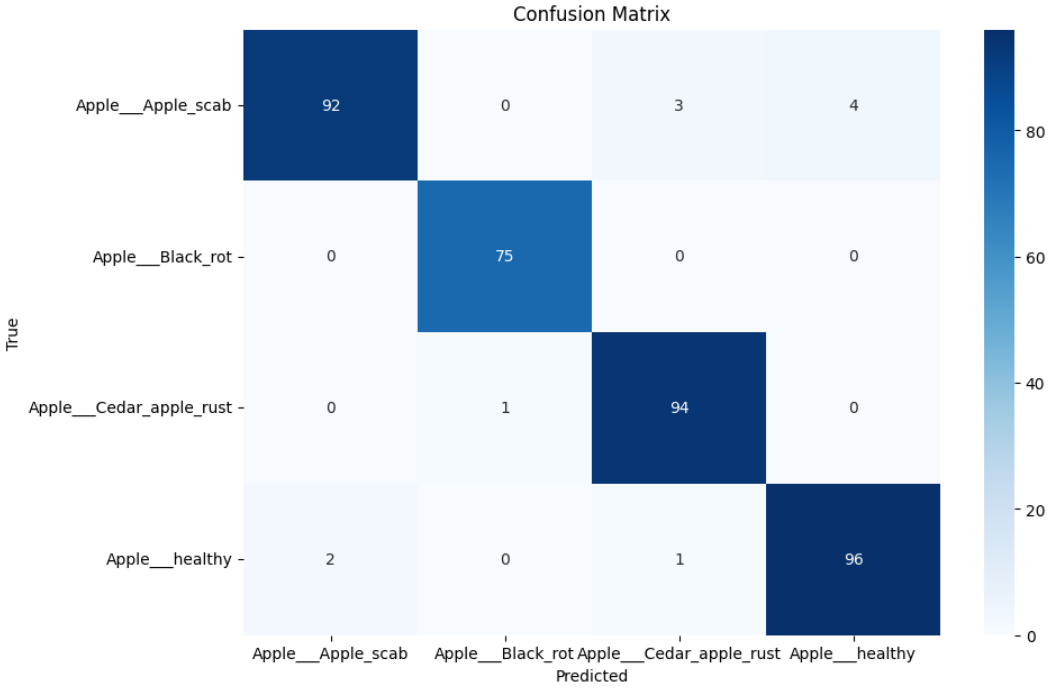

Figure 3: confusion matrix

Predictions on some images that model have never seen them:

Table 7: Predictions Report

Type of Prediction	Image Name	Predicted Class
Correct Predictions	AppleCedarRust1.JPG	Apple___Cedar_apple_rust
	AppleScab3.JPG	Apple___Apple_scab
	image (954).JPG	Apple___healthy
	2-black_rot.jpeg	Apple___Black_rot
	1-cedar_rust.jpeg	Apple___Cedar_apple_rust
	2-cedar_rust.jpeg	Apple___Cedar_apple_rust
Wrong Predictions	1-black_rot.jpeg	Apple___Cedar_apple_rust
	3-cedar_rust.jpeg	Apple___Black_rot
	apple-1.jpeg	Apple___Cedar_apple_rust
	apple-2.jpeg	Apple___Cedar_apple_rust
	apple-4.jpeg	Apple___Cedar_apple_rust

Pre-trained VGG16 Model vs. HOG and Color Histograms

It is premature to conclude that the combination of HOG and Color Histograms is the best feature extraction method given the availability of pre-trained models like VGG16. The combination of Histogram of Oriented Gradients (HOG) and Color Histograms proves to be superior to using a pre-trained VGG16 model for feature extraction in our specific case of plant disease detection. Although VGG16 achieved a similar accuracy (0.9674 ~ 0.97), its generalization from the ImageNet dataset may negatively impact its performance on our specialized dataset of 51 classes and over 60,000 images. The HOG and Color Histograms method, tailored to capture edges, gradients, and color variations, is more suitable for this task. Moreover, VGG16 requires significantly more computational time and resources

for feature extraction compared to the efficient HOG and Color Histograms approach. Specific tests on images like '2-cedar_rust.jpeg' and '2-black_rot.jpeg' showed that the latter method provided more accurate predictions, highlighting its reliability. Given the size of our dataset, using a pre-trained model like VGG16 is not recommended due to its generic nature and computational demands. The specialized feature extraction methods of HOG and Color Histograms are more effective and efficient for our classification task, making them the optimal choice for our plant disease detection project.

Table 8: Pre-trained VGG16 Model Performance

Metric	Apple___Apple_scab	Apple___Black_rot	Apple___Cedar_apple_rust	Apple___healthy
Precision	0.96	0.97	0.99	0.95
Recall	0.94	0.95	0.98	1.00
F1-score	0.95	0.96	0.98	0.98
Accuracy	~ 0.97			

2.3. Disclaimer

The findings and conclusions presented in this report are based on a combination of experimental results and a surface review of existing documentation available in the public domain. It is important to note that the effectiveness of different algorithms and methodologies can vary significantly depending on the specific problem, dataset, and computational resources available.

We did attempt to train traditional machine learning models on the entire dataset. However, due to limited computational power, this effort was not successful. Instead, we leveraged our available resources, including a laptop and the limited free tier of Google Colab, to train a deep learning model. This approach proved to be effective. This demonstrates that resource availability, performance, and time constraints are crucial factors in the decision-making process for selecting the optimal model.

Our conclusions are based on comprehensive experimentation and review of relevant documents, some of which are cited in the reference of this report. It is essential to acknowledge that using different tools, datasets, and addressing different problems may require alternative approaches to achieve optimal results.

Future work should consider exploring a variety of algorithms and methodologies, as the best solution may vary depending on the specific context and constraints.

2.4. Conclusion

Given the limitations of traditional machine learning in terms of robustness and computational concerns, we trained a deep learning model using the same subset of the dataset to solve our problem. Initial validation accuracy was 98.91%, confirming the decision to use a deep learning model for this project.

Using the deep learning model, we achieved correct predictions similar to XGBClassifier but also correctly predicted previously misclassified images (e.g., 1-black_rot.jpeg and apple-1.jpeg).

This chapter successfully explored various feature extraction methods and algorithms for image classification. The optimal feature extraction method identified was the combination of HOG and Color Histograms, achieving 78% accuracy with (Support Victor Machine) SVM. However, the use of the XGBClassifier algorithm significantly improved accuracy to 97%. A pre-trained VGG16 model for feature extraction provided comparable results but had limitations in certain predictions. The final validation confirmed that a deep learning model is the most effective solution, achieving an impressive accuracy

of 98.91% (subset of dataset). This study demonstrates the importance of feature extraction and model selection in developing accurate image classification systems such as Plant Detection.

3. Deep Learning

3.1. Proof of concept

The first approach was to validate the idea of implementing a deep learning model to accomplish the project task. For this proof of concept, a first deep learning network was implemented and trained using a subset of the entire dataset. The limitation of a subset for this proof of concept was selected to simplify and to gain speed from working with reduced amount of data.

This evaluation was executed on a standard laptop (CPU: AMD Ryzen 7 7730U, RAM: 16 GB, Windows 11).

The approach taken can be summarized as follows:

- A convolutional neural network was implemented following the module 151.1 - Dense Neural Networks with Keras
- For the sample images of Apples, Strawberry and Sugar beet were used. With this both of the original datasets (Plant seedlings and Plantvillage dataset) were represented. To further simplify the diseases were ignored in this initial test and the model was only trained to identify the plants. So, this sample contained 3 classes with ca. 5.000 images and represented ~8% of the full dataset
- The sample dataset was isolated from the full dataset. This was done to maintain the integrity of the full dataset as these images were modified in a preprocessing step. During this the images were standardized to the format 128 x 128 x 3 using the OpenCV module
- The standardized dataset was loaded sequentially into a dataframe. Afterwards a training and a testing dataset was created from this dataframe using a split of 80:20
- The above-mentioned model was trained with the sample data for 10 epochs
- The result was analyzed in detail. During this check the integrity of the result was confirmed
- A classification report and a confusion matrix had been created and explored

This initial approach was indeed very encouraging as immediately an F1-Score of 95% on the test dataset was reached at the first run without any tuning!

Following additional findings were obtained:

- In general, using a standard laptop for this project was confirmed to be feasible
- The performance for creating and fitting the (sample) model took around 2 min. This was perceived to be acceptable
- However, the I/O of the images from the disk to create the training and testing dataset was very slow (~30 min for 5.000 images). This point was to be addressed in order to manage the full scope

3.2. Testing the well-known LeNet algorithm on the problem

The following steps were executed in order to scale to the full dataset:

- The Lenet model was selected as introduced in the DataScientest training modules ‘151.2 - Convolutional Neural Networks with Keras (EN)’ and ‘155 - Tensorflow (EN)’
- The scope was extended to the full dataset (51 classes with 60.992 images)

- The dataset was split into training (80%), validation (16%) and testing (4%)
- The model was trained using the datasets for training and validation. The testing dataset was kept separate and only used for evaluation
- The routine to load the data and to generate the datasets was reworked: The Keras function 'keras.utils.image_dataset_from_directory()' was introduced replacing loading the images sequentially into a dataframe. This indeed led to a performance improvement of factor 4.2 for loading the data! An additional benefit of this approach is that this method can directly be applied on the raw dataset. Means any required preprocessing could be incorporated into this dataset. So, no "external preprocessing" of the data was required any longer
- The approach to convert the images to 128 x 128 x 3 was kept. However, using the dataset functionality this conversion now was done directly during loading.

This initial setup already led to reasonable results:

- For the test set an **F1-Score of 0.85** was obtained using 10 epochs
- The training time for the model was ca. 33 min (196 s / epoch)

The training history was visualized using the following diagrams:

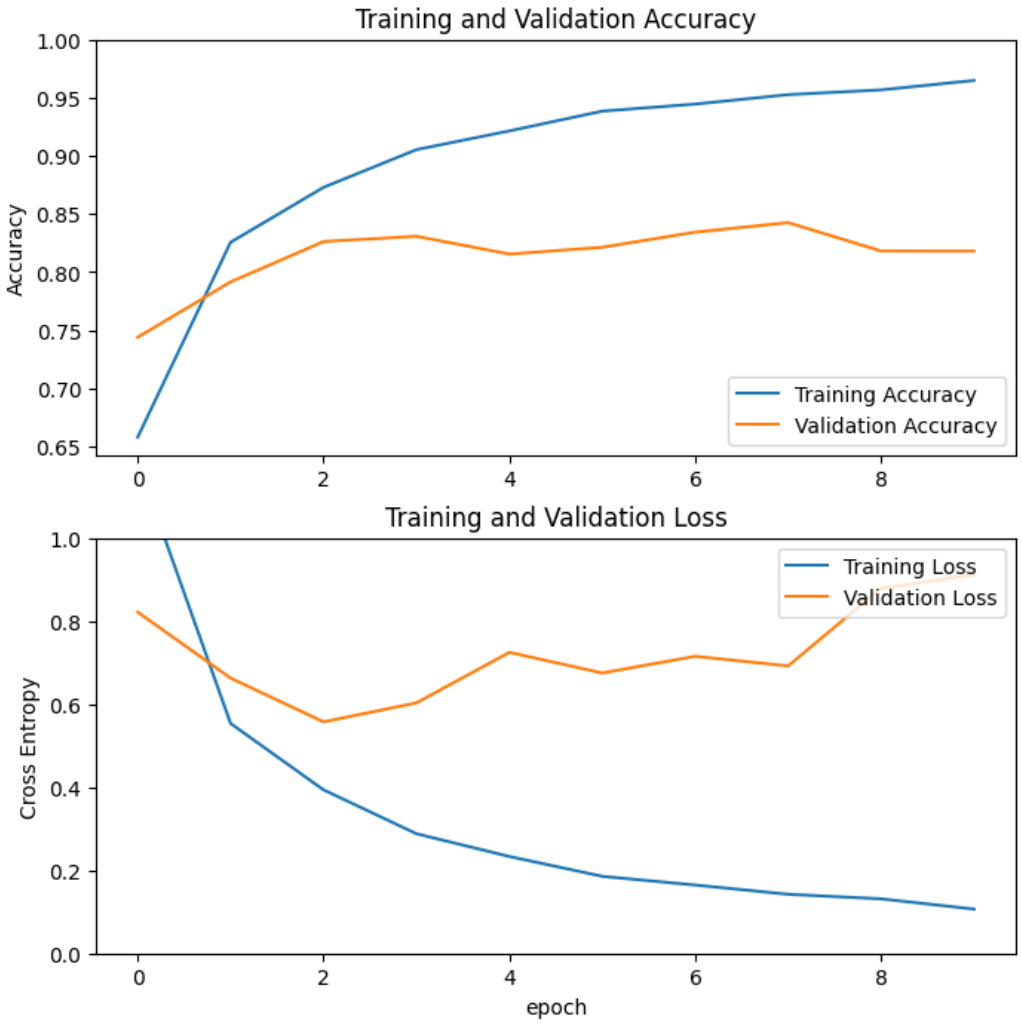

Figure 4: plots showing the evolution of accuracy and loss

First observation was on the delta between the scores of the training and test dataset: The score for the training data was 0.96 while the score for the test data was "only" 0.85. So, this model showed a significant tendency to overfit.

Another observation is that the underrepresented classes suffered in the F1-Score. This was visualized in the following scatter plot diagram which displays the F1-Score by class in reference to the support (number of images in the test dataset):

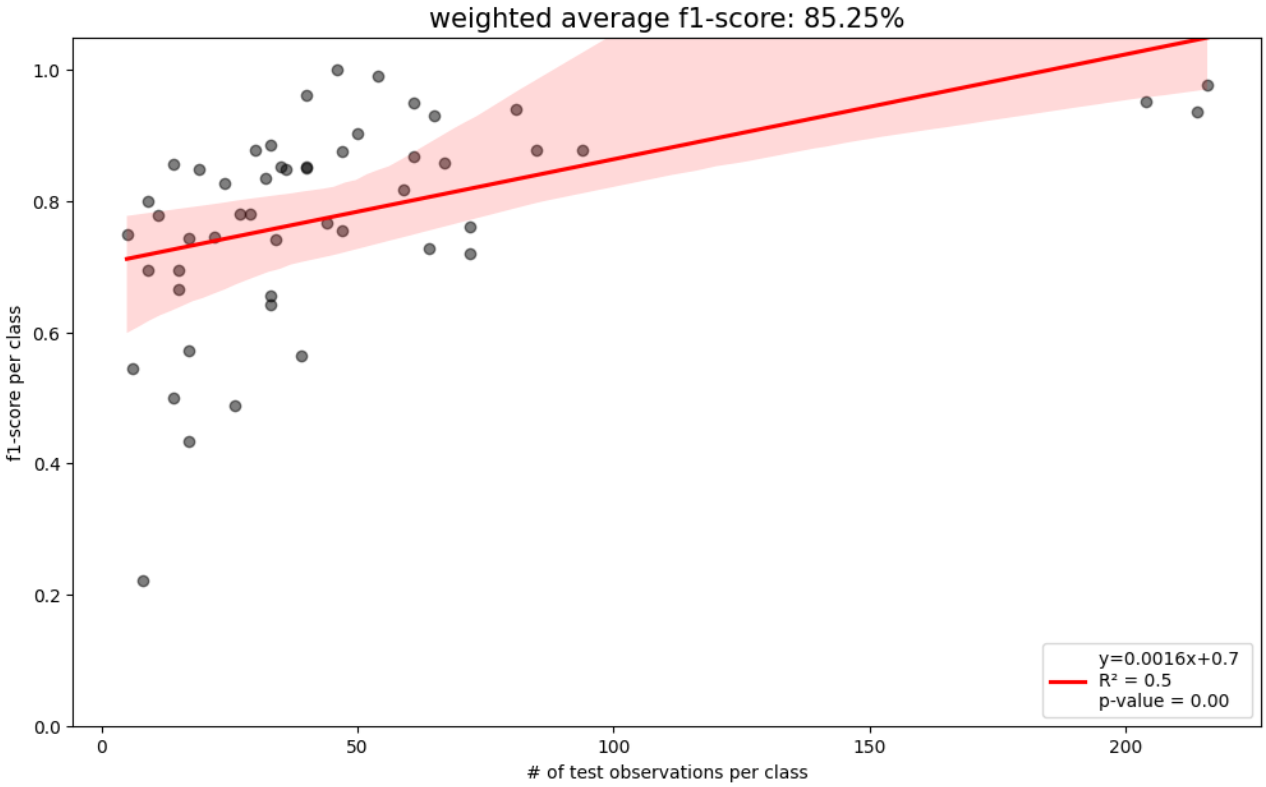

Figure 5: fitted linear regression model for the interplay between f1-score and support

These observations were addressed in the following steps.

3.3. Tuning the LeNet model

Several trials on parameters were executed. In addition, methods described on the Tensorflow page were explored to especially address the observed overfitting behavior.

Trials carried out:

i. The effect of different image sizes was explored. Surprisingly a further reduction of the image size to 64x64x3 resulted in an improved F1-Score of 0.87. At the same time the training time improved to 59s per epoch. The improvement of the training time can be explained as there was less data to be processed. So, this new image size was kept

ii. The model was trained using different batch sizes (between 32 and 256). There was no obvious effect from this. So, a batch size of 32 was kept

iii. Additional dropout layers were added (note: previous model already had one dropout layer). This however had a negative effect on F1-Score and training time. So, this change was revoked

iv. Adjusting the dropout rate for the existing dropout layer from 0.2 to 0.5. This had a small positive effect as the F1-Score improved to 0.88. This improvement was perceived to be marginal, so the previous dropout rate of 0.2 was kept

v. Applying L2 weight regularization on one and on three layers. Also, this change led to a negative effect on F1-Score and training time. Again, this change was revoked

vi. Adjusting class_weights for underrepresented classes: Also, this led to a reduction of the F1-Score and again the change was revoked

vii. Adjustment of learning rate using the callback method. At the same time a callback for early-stopping was also implemented. The number of maximum epochs was increased to 200. This setup led to the best performance obtained with the LeNet until this point.

The final LeNet model can be summarized as following:

- Image size standardized to 64 x 64 x 3 at loading
- Dropout rate in the dropout layer kept at 0.2
- Callbacks for adjusting the learning rate when reaching a plateau and early stopping applied

The technical summary of this model is the following:

Table 9: Model "sequential_4"

Layer (type)	Output Shape	Param #
rescaling_4 (Rescaling)	(None, 64, 64, 3)	0
conv2d_8 (Conv2D)	(None, 60, 60, 30)	2,280
max_pooling2d_8 (MaxPooling2D)	(None, 30, 30, 30)	0
conv2d_9 (Conv2D)	(None, 28, 28, 16)	4,336
max_pooling2d_9 (MaxPooling2D)	(None, 14, 14, 16)	0
flatten_4 (Flatten)	(None, 3136)	0
dropout_4 (Dropout)	(None, 3136)	0
dense_8 (Dense)	(None, 128)	401,536
dense_9 (Dense)	(None, 51)	6,579

Total params: 1,244,195 (4.75 MB)
Trainable params: 414,731 (1.58 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 829,464 (3.16 MB)

Applying this model indeed led to satisfying results:

- For the test set an F1-Score of 0.92 was obtained using 73 epochs
- The training time for the model was ca. 40 min (33 s / epoch)

There were however only marginal improvements after passing ca. 30 epochs:

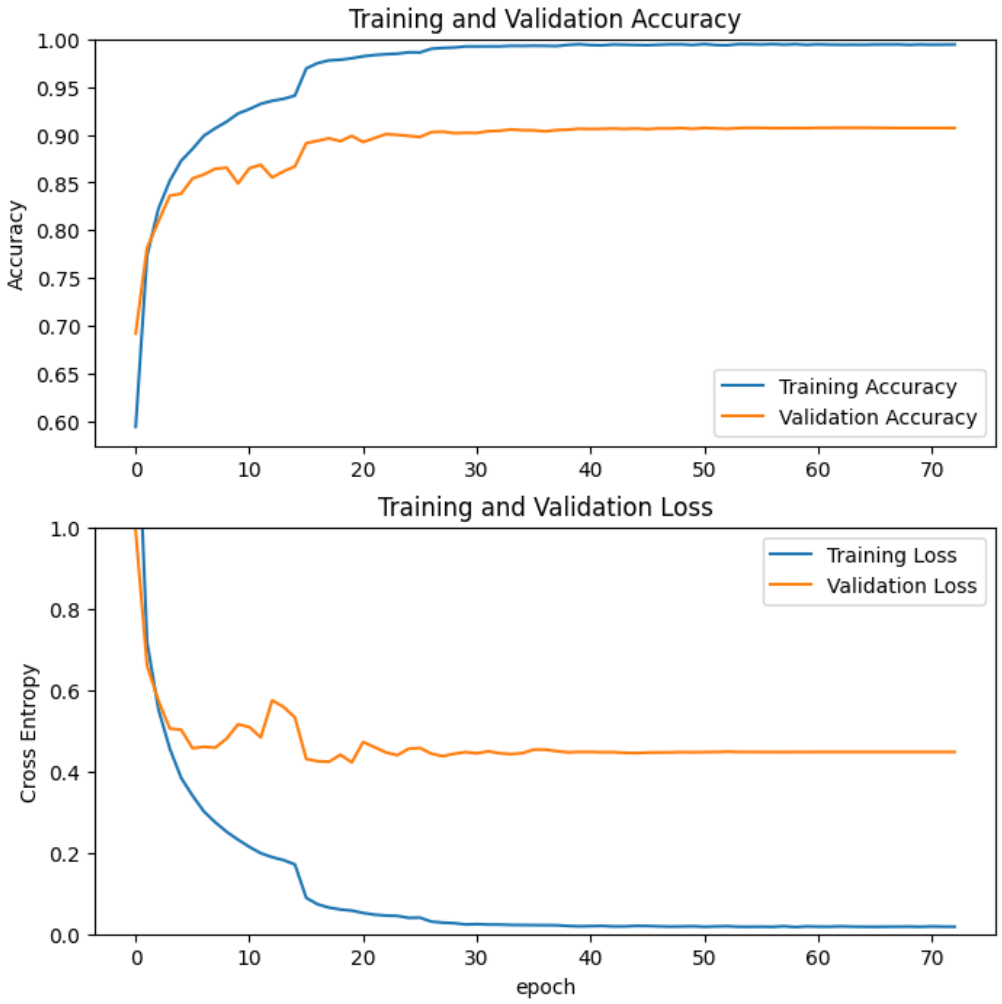

Figure 6: evolution of the accuracy and loss

We created the confusion matrix for the test dataset of around 2.400 images, confirming the overall good quality of the result for most of the classes:

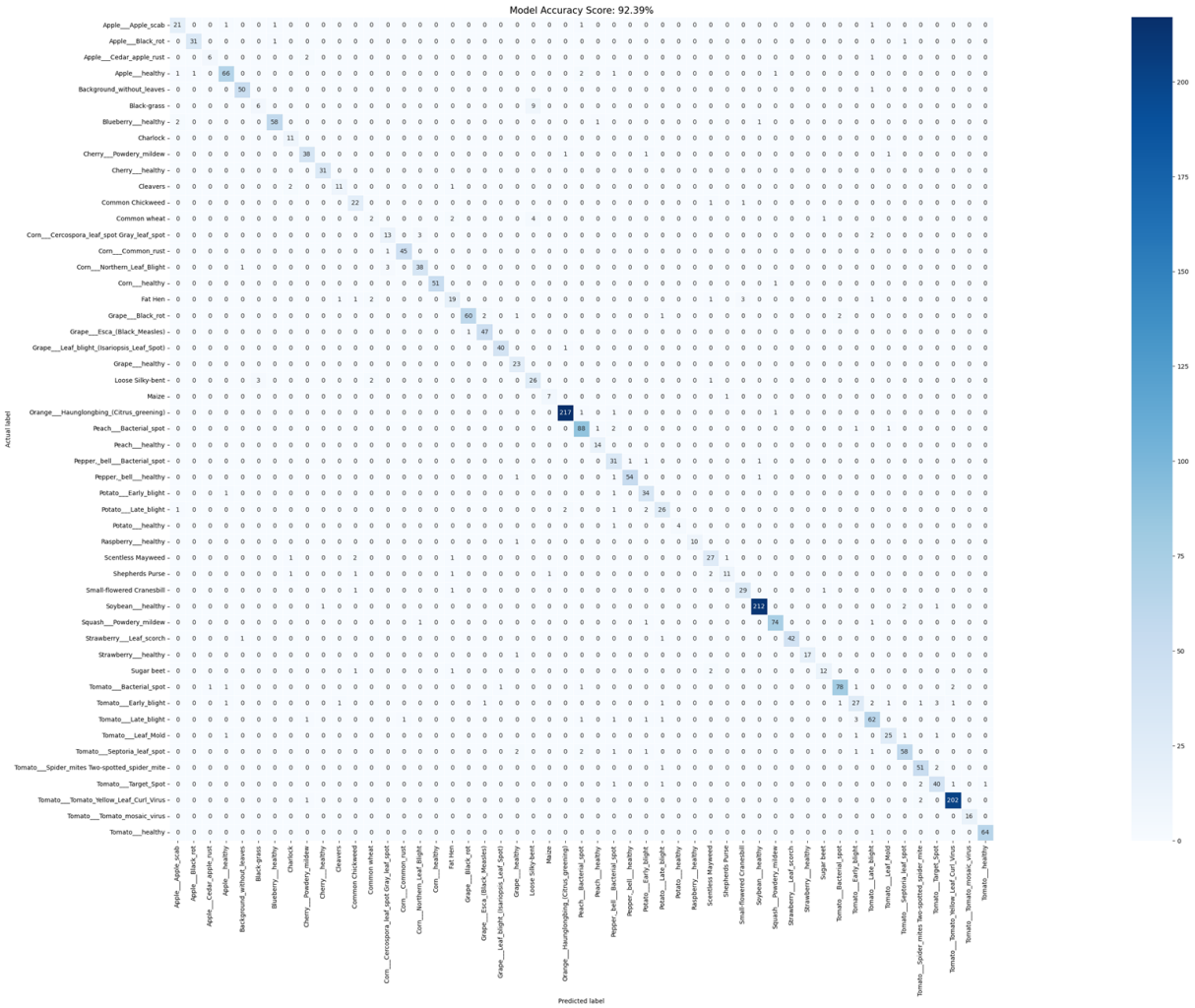

Figure 7: confusion matrix

Previous observations of overfitting behavior and having the underrepresented classes suffering in the F1-Score was persisting as visible from the scatter plot diagram introduced earlier:

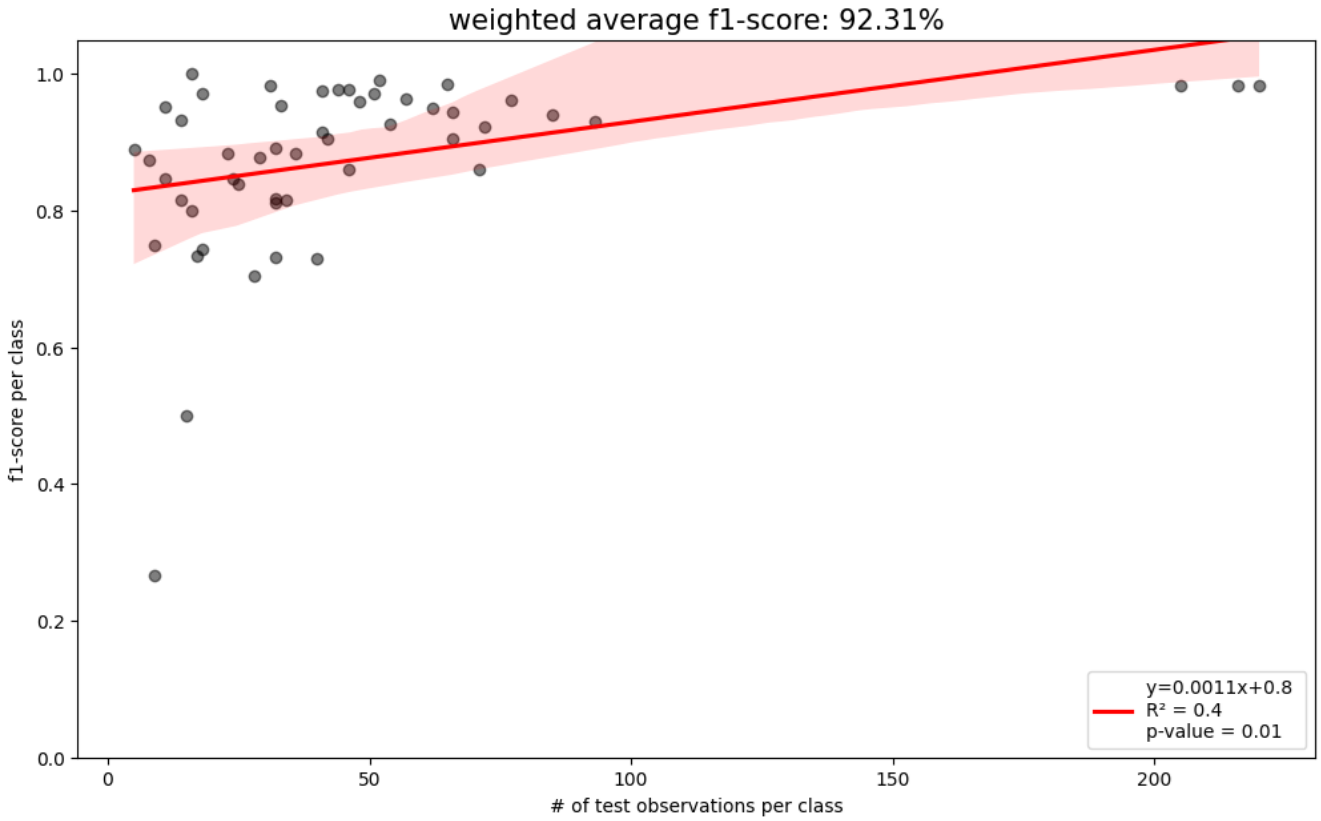

Figure 8: linear regression model between f1-score and support

There are many classes with F1-Scores between 0.95 and 1.0!

The classes with low F1-Score and thus most difficult to be predicted correctly had been:

Class	precision	recall	f1-score	support
Common wheat	0.33	0.25	0.29	8
Black-grass	0.62	0.36	0.45	14
Fat Hen	0.68	0.65	0.67	26

Interestingly these three classes are all originally from the Plant seedlings dataset.

4. Augmentation-aided Deep Learning

Yet another Deep Learning approach we've employed specifically for the purpose of dealing with the augmentation-aided image classification problem was building and training the 7-layer-deep artificial neural network (with over 6 million trainable and over 12 million optimizer parameters), closely resembling (up to mimicing) the original architecture of the world-renowned if not also pioneering Convolutional Neural Network (CNN) dubbed "LeNet-5".

Of course, the possibilities to build (thanks to developers of Keras, with ease), train, fine-tune and retrain a CNN are almost endless, and decent results, at least for our relatively homogeneously

formatted dataset, would neither be the 8th world wonder nor like searching the needle in the haystack.

But we were well aware of a fowl in hand being better than two flying, and that is why we decided to opt for the LeNet family's most iconic offspring "LeNet-5", which, together with it's a bit skinnier brother LeNet-4, were specifically introduced to face the classification challenges with large datasets involved [5]. Moreover, the volume of the legendary dataset of handwritten digits MNIST counts around 60/70K square images, coming far too close to the volume of ours to neglect this promising coincidence. Further justification was the similar spatial composition (i.e. isolated units of more-less centered plant leaves vs isolated and well-centered handwritten digits).

Surely, just copying the LeNet-5 in every detail would have been a quite short-sighted decision, because necessarily needs to account for particularities of every single application case. And, certainly, we never fell short of these intuitive expectations.

4.1. Prevention of Overfitting

Overfitting is widely deemed a true and sadly intrinsic (unavoidable per se) malignant disease of training Machine Learning models – thus, no wonder why we made sure to undertake nearly all plausible and (in every respect) feasible measures at our disposal in order to combat it:

-1- via introducing a (less aggressive than L1) L2-regularization of weights, mostly for larger layers for a greater impact and at the same time avoiding overloading;

-2- via introducing Dropout layers with rates tailored to the scale of respective layers (i.e. using larger rates up to 0.5 between layers with more neurons);

-3- predominantly via introducing the augmentation layers (in this case, provided natively by Keras [6]) on the top of the LeNet-5 architecture, which equipped our fine-tuned version of LeNet-5 model with the steadiest and yet optimized armor against overfitting due to the following couple of considerations:

> -i- the randomized nature of augmentation layers secured the constant non-zero variance in the data with every single batch fed while training the model – in other and simpler words, the model was learning novel unseen features every single iteration within every single epoch;
>
> -ii- the great API flexibility of Tensorflow/Keras allowed a seemless incorporation of augmented images without any bit of extra disc (i.e. storage) load, and the added loaded on memory was kept as low as the freely choosable batch size.

-4- last and (frankly 😊) the least: via utilizing batch normalization layers to standardize the outputs of activation layers – although theoretically it would have been more promising to employ batch normalization before activation layers and not after (moreover, we've used the piecewise linear and yet still technically non-linear ReLU activation), the results we've obtained were cheering enough (sometimes we called them plainly excellent) to allow any reasonable objection and any need of readjustment. All in all, we feel the urge to stress that batch normalization is more used to facilitate speedier learning/training of the model [7] by its contribution to the stabilization of the underlying optimizer (i.e. so called "loss") function and (thus) allowing (relatively) larger learning rates for consistently effective reduction of error rate.

The three figures shown right below represent the visual and self-explanatory wrap-up (supported by respective captions) of the more than satisfying performance (topping 91% for the training and reaching 85% for the validation sets) of the augmentation-aided and fine-tuned LeNet-5 model that we've rebuilt literally from scratch to accommodate all the aforementioned optimization measures:

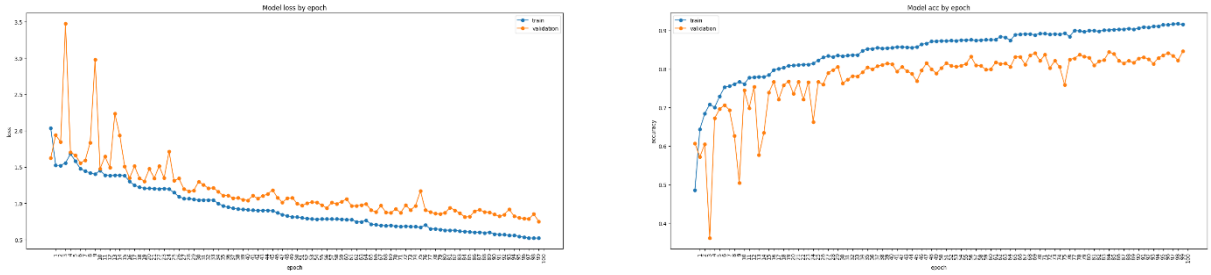

Figure 9: line plots showing the evolution of loss and accuracy rates over 100 epochs (batch size: 32)

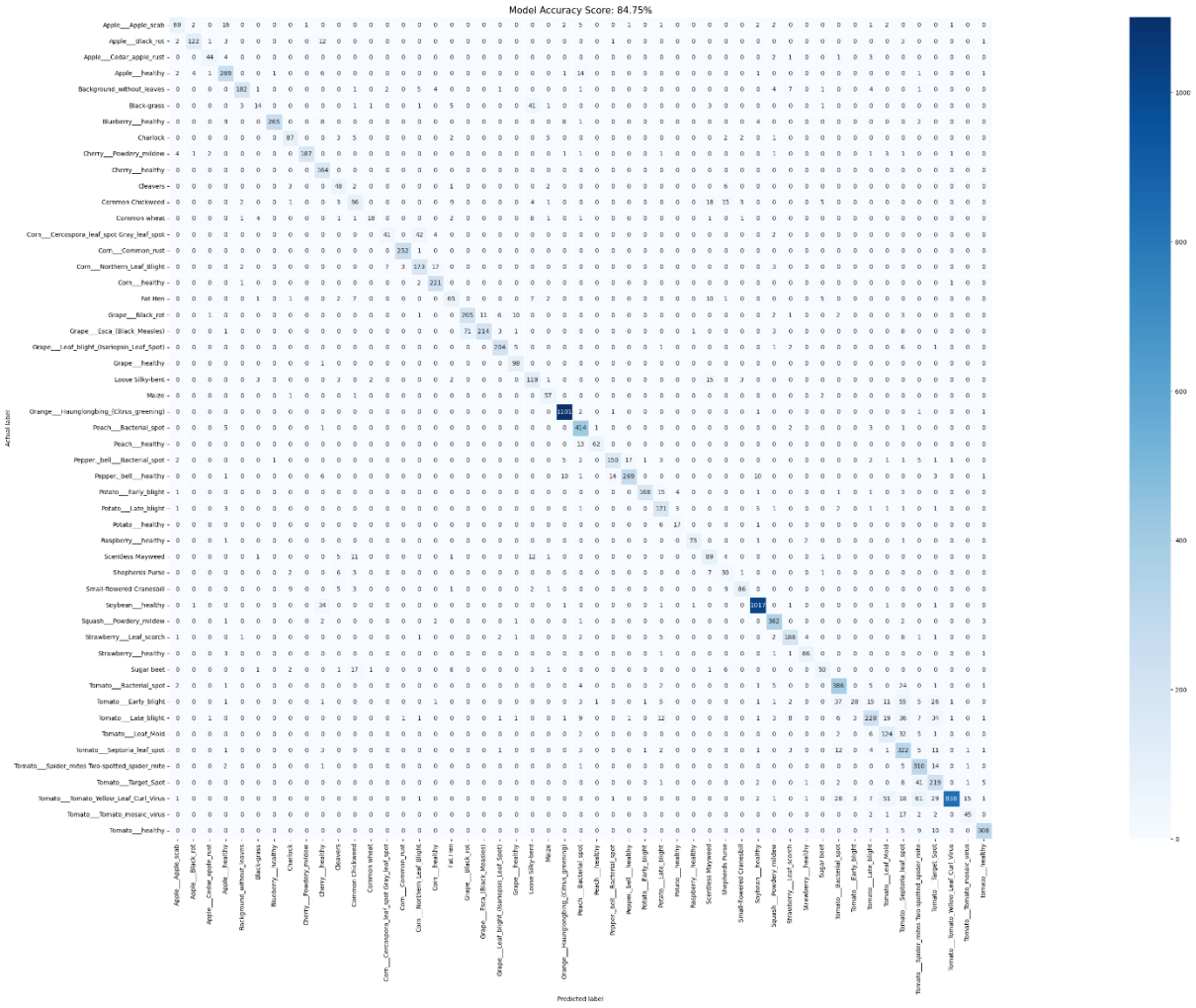

Figure 10: heatmapped confusion matrix clearly displaying the well-accurate performance of the trained model on the test set of around 12K images over pretty much the entire spectrum of 51 classification labels

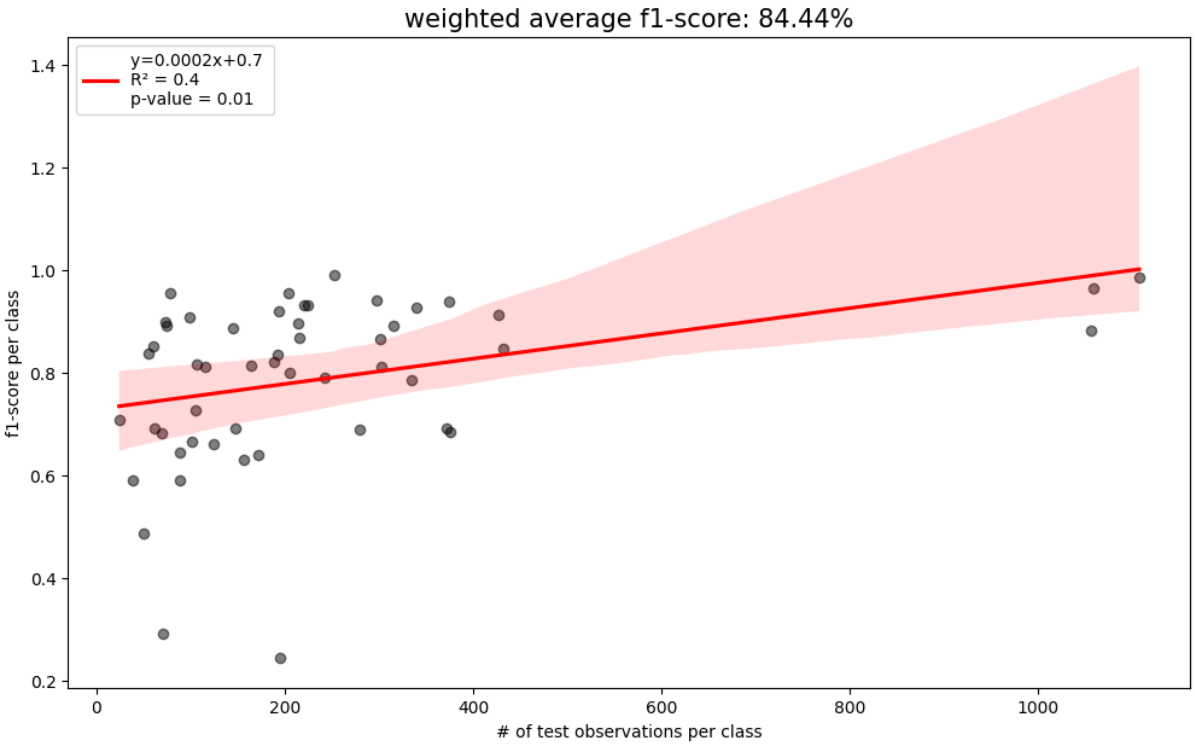

Figure 11: the major derivative of the scikit-learn's classification report: although the R^2 amounting to 0.4 is considered to be at the edge of high-to-low correlation, the steadily decent f1-score across the entire range of supports is evident (the shaded area represents the 99% confidence interval)

The mascot of our modification of LeNet-5 was undoubtedly the effectively built-in randomized augmentation composed of fine-tuned transformations like zooming in, flipping, altering brightness and contrast as well as injecting Gaussian noise (we permissibly refrained from the application of rotation as well as from zooming out, because, firstly, our augmentation was well-rich even without these addenda, and, secondly and foremostly, due to the inevitable injection of systematic edge artifacts that might have caused building of unwanted background features as classification criteria for foreground what would have been an unacceptable risk – we want to use this contextual opportunity and emphasize the reader's attention to the fact that, unlike MNIST, our dataset is still subject to a fine-grained segmentation, an effective background removal and object recognition, which are still pending and most probably not in the scope of the current project development stage).

As the plots clearly exhibit, there are no evident signs of any emerging overfitting and, quite on the contrary, probably (way) more epochs are needed to get the never-saturating learning with randomly and compositely augmented images to the consistently better performance rates, with no immitent danger of overfitting.

4.2. Augmentation-aided vs Augmentation-free CNN

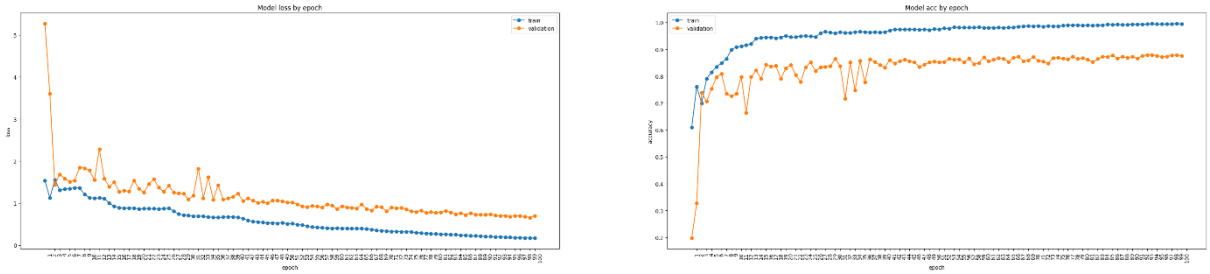

Figure 12: line plots showing the evolution of loss and accuracy rates over 100 epochs (batch size: 32) for the identically built and tuned LeNet-5 modification, but without any augmentation layers integrated in the architecture

Although the loss & accuracy evolution plots for both augmented and non-augmented versions of our custom-made LeNet-5 models might look similar at first glance, they are far from being indistinguishable and exhibit the following "threefold distinction" when taking a closer and somewhat quantitative look:

-1- the augmented model shows considerably more volatility (we strongly believe there is no need in delving into quantitative measurements of volatilities of plotted series – nitpickers are welcome to verify 😊), what is rather explicitly deducible from nature of involved augmentation

-2- the ballpark average estimate of relative proximity for both plotted "curves" is significantly greater (oxymoron-alert: "great proximity") for augmented model – e.g. for validation accuracy graphs, the (visually) saturated range of augmented model exhibits the rate of little over 80% with relative proximity of around 7.5% with respect to the training accuracy, whereas for the augmentation-free model the relative proximity is nearly doubled amounting to little over 12%

-3- non-augmented model reaches the stagnation considerably sooner/steeper, and Keras callbacks [8] integrated into the model's training phase actually fired/triggered for the augmentation-free model forcing it to stop execution and restoring the best weights of the previous (i.e. 99th) epoch, whereas augmented model remained vibrantly evolving to the very end and even hungered for more epochs.

5. Transfer Learning

The notebook that we've created is an amazing deep learning model that has been specifically designed to tackle image classification tasks. It's like having a super-smart assistant that can look at images and tell you what they are!

This model has been trained using a technique called transfer learning, which means it has learned from a pre-existing model called MobileNetV2. Think of it as building on the knowledge of a really smart friend who already knows a lot about images.

The reason of choosing this so far the latest version of Google-made MobileNet that comes equipped with Keras 3 API [9] was predominantly, if not exclusively, based on the considerations of limited resources at our disposal. Although being generally inferior in performance when compared to its big brother Xception as well as to the Microsoft's ResNet and the unbelievably beefy VGG from Oxford, MobileNet(V2) prioritizes computational efficiency without compromising performance which still keeps up with those of the named Big Three [X].

5.1. Preparation

The notebook model has gone through a meticulous process of data collection and preparation. It has been trained on a dataset called the Fruits and Vegetables Image Recognition Dataset, which contains a wide variety of images of different plant's leaves. The model has been trained to recognize and categorize these images accurately.

To make the model even smarter, data augmentation techniques have been applied during the training phase. This means that the model has been exposed to different variations of the images, making it more robust and less prone to overfitting. It's like giving the model a crash course in handling all sorts of image variations.

5.2. Training

During the training process, the model has been closely monitored using validation loss and accuracy metrics. These metrics help us understand how well the model is performing and whether it's getting better at classifying images correctly. It's like keeping an eye on the model's progress and making sure it's on the right track.

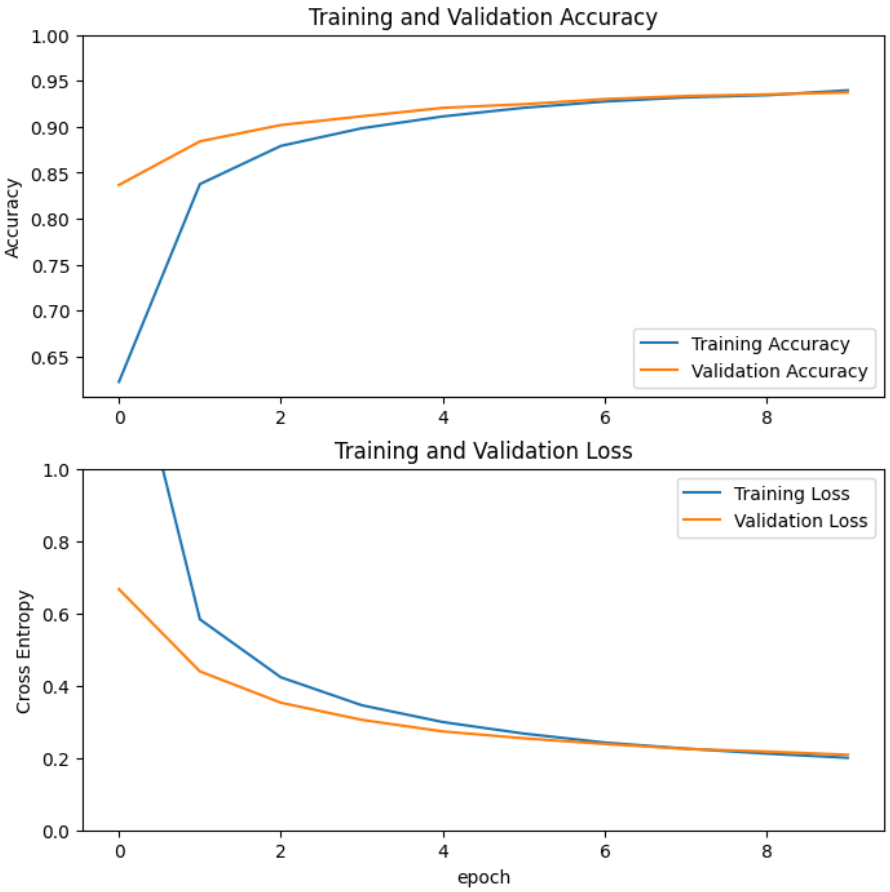

Figure 13: Accuracy and Loss of training and Validation

But the journey doesn't end there! If the model's performance falls short of expectations, there are strategies in place to fine-tune it. This could involve adjusting hyperparameters like the learning rate and optimizer or extending the training duration. It's all about making the model even better at what it does.

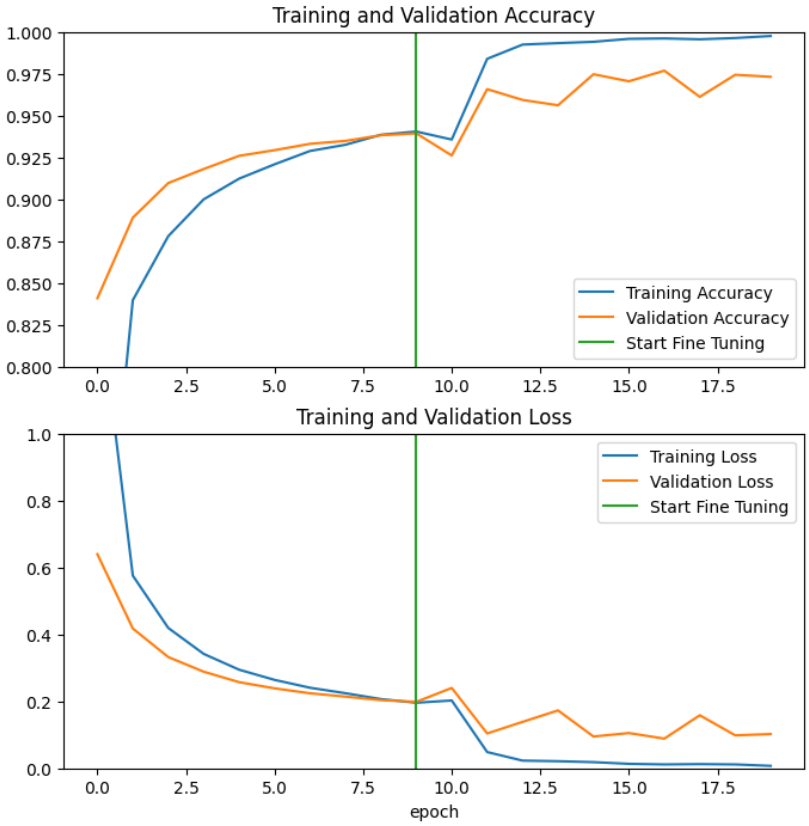

Figure 14: Accuracy and Loss of training and Validation after tuning the hyperparameters

The results are excellent for almost all of the classes as shown by the confusion matrix:

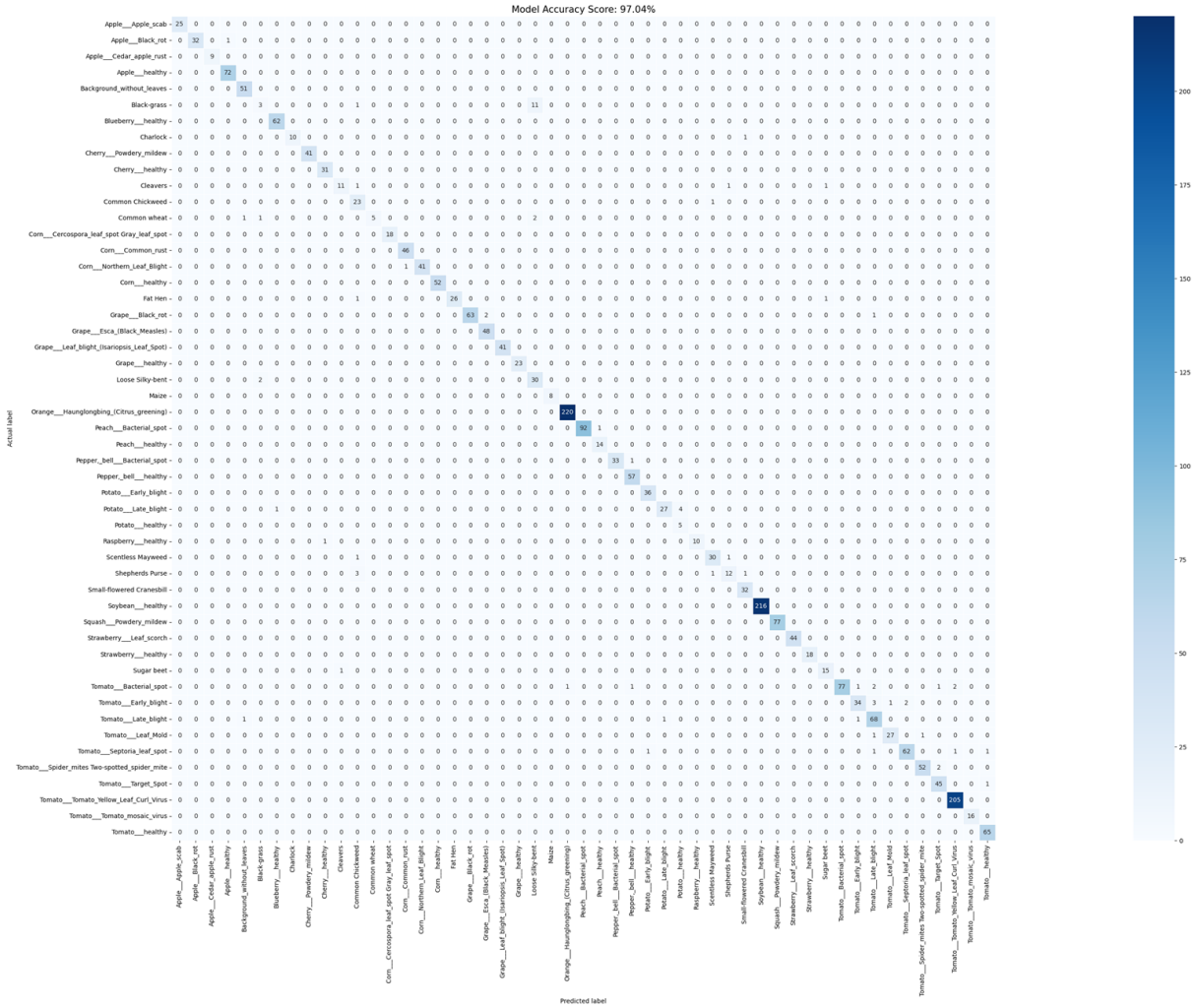

Figure 15: Confusion matrix after applying transfer learning

Once the model is trained and fine-tuned, it's ready to make predictions! It can take new images and tell you what category they belong to. It's like having a personal image classifier that can help you organize your inventory or identify different products based on their images.

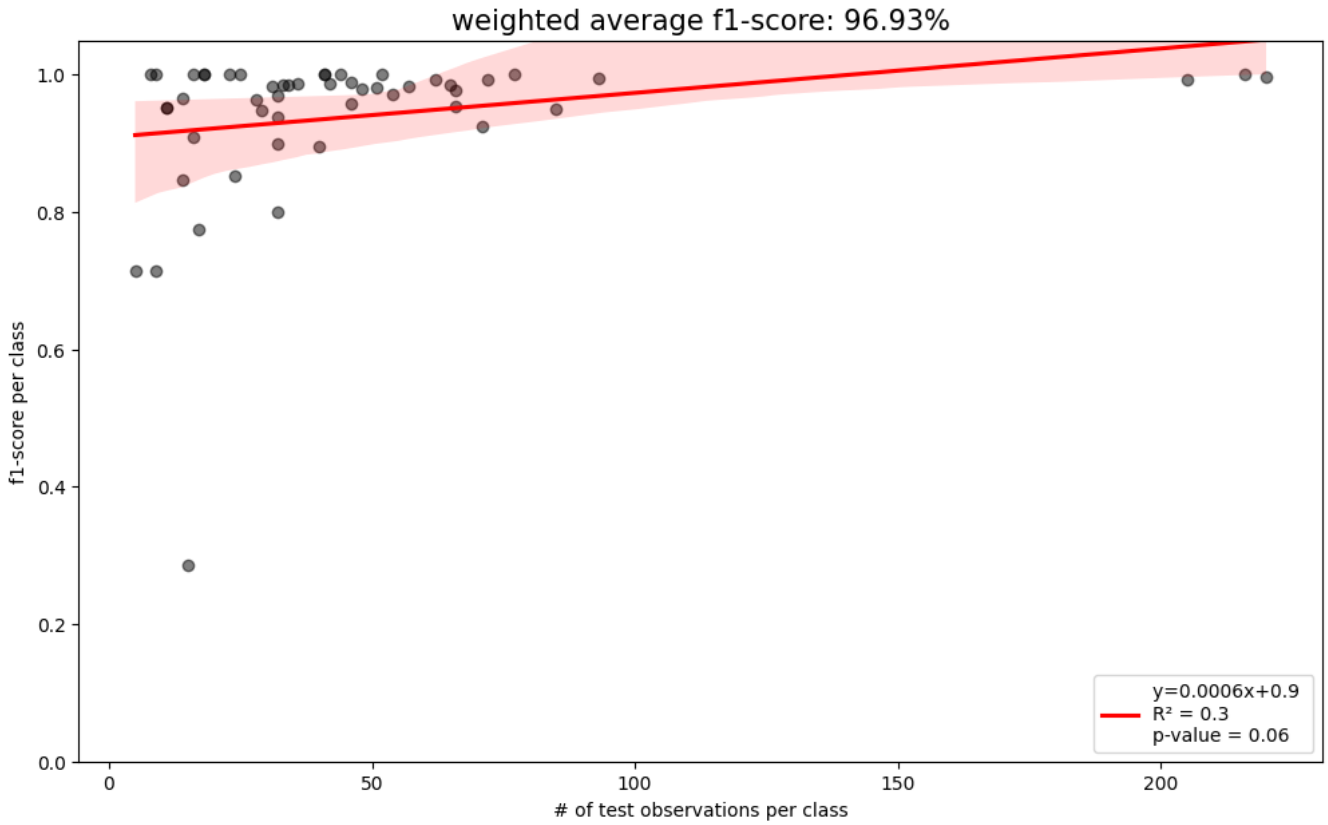

Figure 16: F1-Score vs Support

In conclusion, this notebook is a powerful tool that combines the magic of deep learning with the expertise of MobileNetV2. It's been trained on a diverse dataset, fine-tuned for optimal performance, and is now ready to impress with its accurate image classification abilities.

6. References

1. Bajaj, A, Performance Metrics in Machine Learning [Complete Guide]: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide
   Accessed on 12.06.2024

2. API Reference, classification_report: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
   Accessed on 12.06.2024

3. SIFT feature detector and descriptor: https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_sift.html
   Accessed on 12.06.2024

4. Fourier Transform and Wavlet Transform: https://www.kaggle.com/code/shuvoalok/special-technique-for-image-classification#classification-of-image-using-Fourier-Transform-and--Wavelet-Transform
   Accessed on 12.06.2024

5. LeCun, Yann, Bottou, Léon, Bengio, Yoshua and Haffner, Patrick. "Gradient-Based Learning Applied to Document Recognition." Paper presented at the meeting of the Proceedings of the IEEE, 1998.

6. Source code of Keras preprocessing layers, official GitHub Repository: https://github.com/keras-team/keras/tree/master/keras/src/layers/preprocessing
   Accessed on 12.06.2024

7. Shipra Saxena, Introduction to Batch Normalization: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/
   Accessed on 12.06.2024

8. Callbacks API, Keras 3 Documentation: https://keras.io/api/callbacks/
   Accessed on 12.06.2024

9. Keras Applications, Keras 3 API Documentation: https://keras.io/api/applications/
   Accessed on 14.06.2024

10. Tannaz Mostafid, Overview of VGG16, ResNet50, Xception and MobileNet Neural Networks: https://medium.com/@t.mostafid/overview-of-vgg16-xception-mobilenet-and-resnet50-neural-networks-c678e0c0ee85
    Accessed on 14.06.2024