Università degli Studi di Milano

Algorithms for Massive Datasets

Plants leaves recognizer, a Convolutional Neural Network application

Author:
David A. Fernandez *(988346)*

Lecturer:
prof. dr. Dario Malchiodi

June 11, 2023

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Disclaimer

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

Contents

1 Introduction

The following report describes an implementation of a Convolutional Neural Network (CNN) algorithm to a dataset composed of images of leaves of different species. The dataset is available for download from the platform Kaggle in the following link https://www.kaggle.com/datasets/csafrit2/plant-leaves-for-image-classification. The dataset is published under the name 'Plants Leaves for Image Classification'.

This dataset is originally set up to classify healthy leaves from diseased leaves on 11 different species: Alstonia Scholaris, Arjun, Basil, Chinar, Gauva, Jamun, Jatropha, Lemon, Mango, Pomegranate, and Pongamia Pinnata. For the purpose of this application the classification goal will be only toward the species of the plants, disregarding the health status of the leave. The dataset is originally composed of four folders: train, validation, test, and images to predict, but only the first three will be used to select the best model among three proposals; the images to predict folder will be ignored in this application.

As previously mentioned, three different CNN algorithms will be tested on the Plants dataset. For this purpose, the algorithms will be trained on the training dataset and validated using the validation dataset. Model selection will be performed based on validation loss scores, and further analysis will be provided on the winner model .

The remainder of this paper is organized as follows: Section 2 describes how the data has been organized and the pre-processing techniques, Section 3 describes the algorithms and their implementations as well as a description on how this solutions scales up with data size, finally, in Section 4 the three different algorithms are described and their results discusses as well as some suggestions for future improvement.

2 Data Pre-Processing

Before starting pre-processing the data, it is common practice to visualize what kind of images we are working with. In **Figure 1** a sample of random images from the training folder is displayed. Also, gathering a notion of how many images are in each of the folders helps to understand what the algorithms will be dealing with. For this purpose, **Table 1** describes folders size.

Figure 1: Sample images from the training set of both healthy and diseased leaves

Folder	**No. of images**
Train	4,274
Validate	110
Test	110

Table 1: Number of images in the train, valid and test folders.

The data pre-processing phase starts by merging the folders 'healthy' and 'diseased' in the train, valid and test folders for every plant species available. This is, reducing 22 folders to 11 folders. Once the folders are merged and there's a single folder for each species, all images are resized to be 200x200 pixels using the *imread* library. It is worth highlighting that the network requires all input images to be of the same size, as the number of nodes in the input layer is fixed, hence the resizing. **Table 2** describes the number of images in each of the merged folders.

As it is clearly seen, the dataset is unbalanced, meaning that there are more images of certain categories than others. This may affect the performance of the network, as it will not learn the features of certain species as good as other ones.

Folder (labels)	**Train**	**Valid**	**Test**
Alstonia Scholaris	412	10	10
Arjun	432	10	10
Basil	244	10	10
Chinar	203	10	10
Gauva	398	10	10
Jamun	603	10	10
Jatropha	237	10	10
Lemon	216	10	10
Mango	414	10	10
Pomegranate	538	10	10
Pongamia Pinnata	577	10	10
Total	**4,274**	**110**	**110**

Table 2: No. of images per folder per species

The *imread* library is also used to read the pixels of each image and transform them into RGB values to be stored as arrays. In this sense, each image is then converted into a list of 200 matrices, each one of size (200x3), with each column displaying the values of each of the three RGB channels.

The collection of matrices of the train folder is then split in two objects: the first one, containing 4,274 elements called X_{train} which contains only the features of each image and the second one, y_{train}, a sparse matrix of (4,274 x 11) which contains the labels of each image. The y_{train} matrix is obtained through one-hot-encoding, i.e. is a [0,1] with value [1] in the column index of the corresponding label of each image and value [0] elsewhere. Finally, to simplify calculations in the network, X_{train} values are normalized by a constant of 255 (the maximum of RGB pixels). This process is repeated for the valid and test folders as well.

3 Convolutional Neural Networks

CNN as we know them nowadays can be traced back as far as the 80's, when Fukushima published the paper *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position* (Fukushima and Miyake, 1982). Further research has been done since then, but the main idea prevails: add Convolutional layers to a traditional Neural Network (NN) architecture to improve performance on computer vision tasks. The notable difference is that CNN allow us to encode image-specific features into the architecture whilst further reducing the parameters required to set up the model (O'Shea and Nash, 2015).

Simple CNN may look something like the display in **Figure 2**. Of course, in reality the number of convolution and hidden layers may vary depending on the task at hand, as well as their parameters. This opens up a vast and large universe of possibilities to build CNN models (no without increasing it's complexity and challenges).

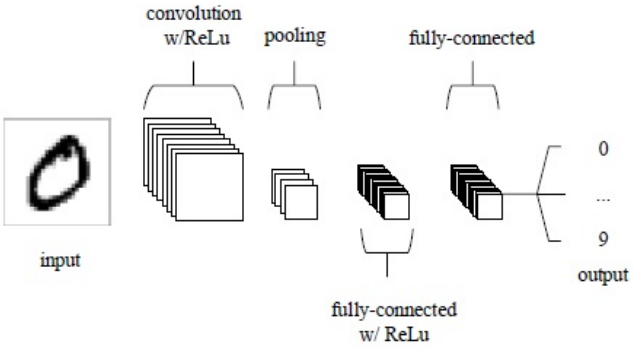

Figure 2: A simple CNN architecture, comprised of five layers (O'Shea and Nash, 2015)

It is worth mentioning that CNN architectures have won several different image recognition contests such as ImageNet, ICPR (Mitosis Detection in Breast Cancer), MICCAI (Medical Image Computing and Computer Assisted Interventions), IJCNN Traffic Sign Recognition Competition, among others, paving the road for further improvement as more and more real case applications are developed using these algorithms.

But how does this algorithm scales up with data size? Well, data size is actually where its power rests upon. These kind of algorithms are designed to be trained with thousands if not million of images, and it is imperative to do so because the more training data it has, the best performance it will yield; data quality is paramount as well, there's no much value in using high volumes of poor data.

The 'heavy' phase of the algorithm is then the training, because during this step that the network learns it's parameters, i.e. the weights of each node in each layer. State of the art algorithms can take a couple of days or even weeks to train, as seen in the implementation details of the VGG CNN paper (Simonyan and Zisserman, 2014); it is worth noticing that the training times may be parallelized by using several GPUs and thus being substantially reduced.

Once learning is done, the model is saved and ready to be executed, and it can start classifying images at once. The burden of the classification depends, on how many images the algorithm may have to process at the same time, but usually this is not costly, as the CNN is not learning any more (unless configured to do so). In most cases it can be considered that a well-trained network processed more images at a given time than the ones it will process while being productive.

4 Experiments and Results

Three different architectures will be trained and tested on the data. General strategy is to start with an architecture similar to the one proposed in (Wu and Qian, 2021) and combine it with some elements from (Simonyan and Zisserman, 2014) for **Model 1**, then different settings will be tested for **Model 2** and **Model 3**. Model selection will be performed based on validation loss score, then a risk estimate will be provided using the test set, as well as precision and recall scores.

Common ground for all models are the activation function of the output layer: the softmax function; loss will be calculated using categorical crossentropy; batch size is 32, number of epochs is 50 and early stopping is set with a patience of 10 epochs after no improvement on validation accuracy.

4.1 Architectures

Model 1

Hyperparameters for Model 1 are displayed in **Table 3** and architecture can be seen in **Figure 3**.

Hyperparameters	**Value**
Conv1 kernels	16
Conv2 kernels	32
Conv3 kernels	64
Kernel size	3x3
Activation	ReLU
Kernel Initializer	He uniform
Padding	Zero padding
Max Pooling	2x2
FC1 neurons	32
FC2 neurons	16
Optimizer	SGD*

*lr=0.001, momentum=0.9

Table 3: Hyperparameters, Model 1

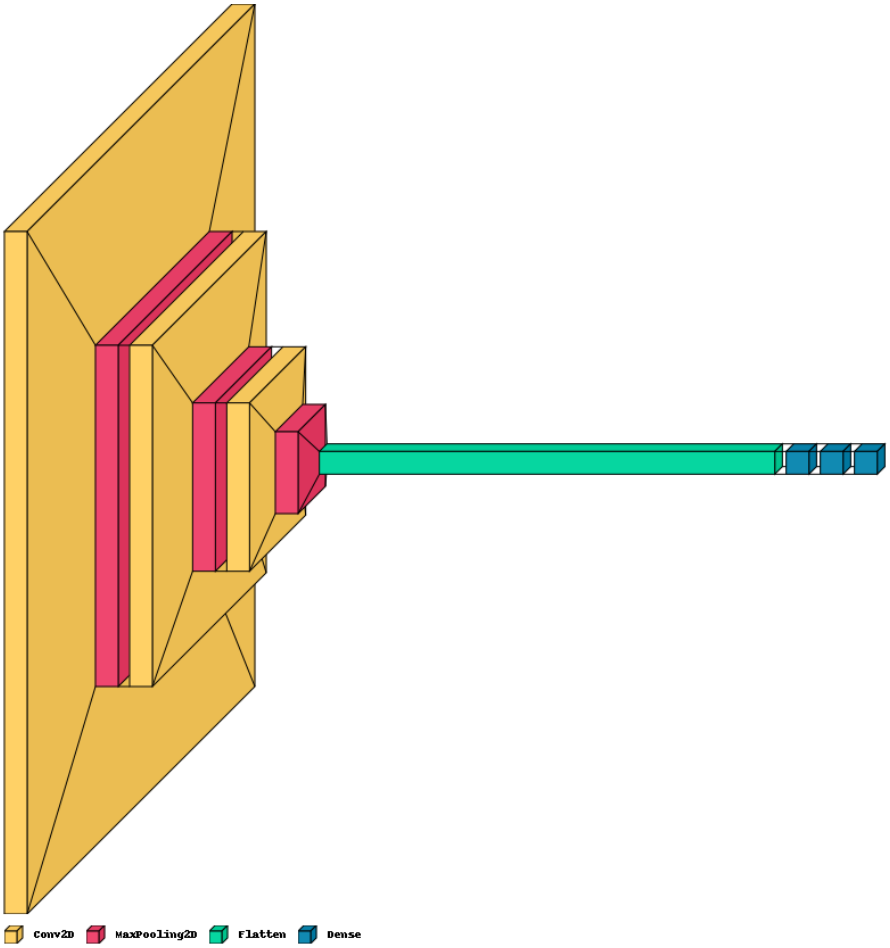

Figure 3: Architecture, Model 1

Model 2

Innovations for **Model 2** are the following: to test a different kernel initializer, the Glorot normal is used, padding is set to valid (no padding) and a Dropout layer of 0.2 is added after each max pooling layer. Motivation behind these minor changes is basically to test if something slightly different would achieve better results, specially with the optimizer, for which its parameters were chosen based on Keras website implementation. Hyperparameters are summarized in **Table** 4 and architecture is plotted in **Figure** 4.

Hyperparameters	Value
Conv1 kernels	16
Conv2 kernels	32
Conv3 kernels	64
Kernel size	3x3
Activation	ReLU
Kernel Initializer	Glorot normal
Padding	No padding
Max Pooling	2x2
Dropout (Conv)	0.2
FC1 neurons	32
FC2 neurons	16
Optimizer	SGD*

*lr=0.001, momentum=0.9

Table 4: Hyperparameters, Model 2

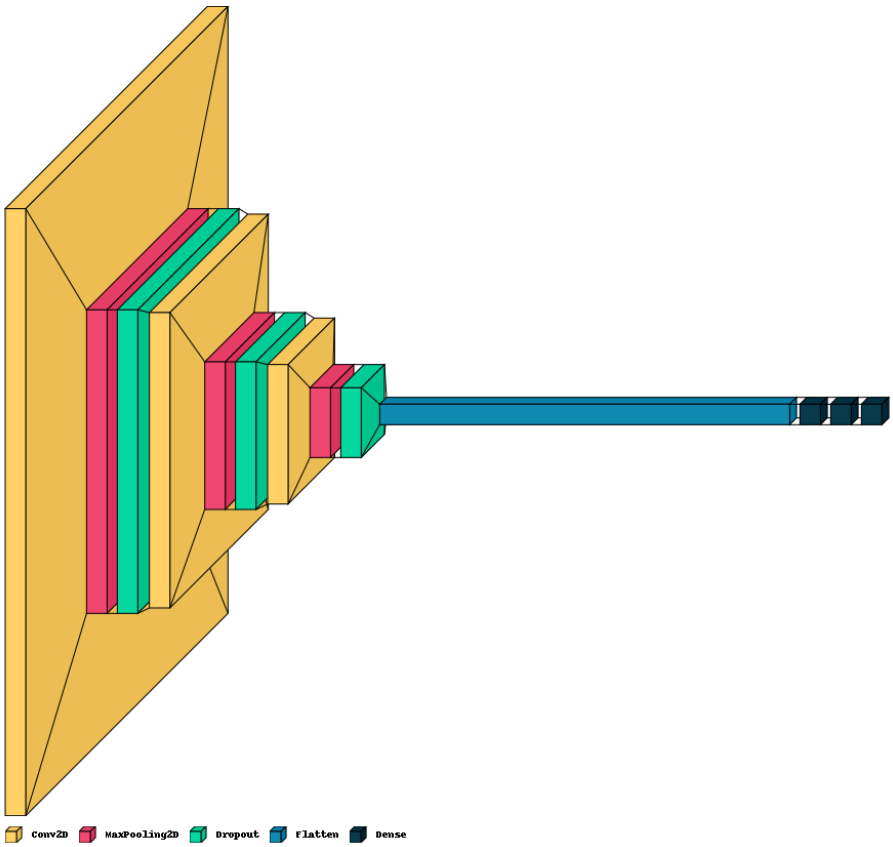

Figure 4: Architecture, Model 2

Model 3

Model 3 will not have any Dropout layer, based on the hypothesis that it may accelerate the training stage too much. This time, innovations are: optimizer is switched to Adam instead of SGD, padding is same (zero-padding) as in **Model 1** and a fourth convolutional layer is added before the fully connected layer.

Hyperparameters	**Value**
Conv1 kernels	16
Conv2 kernels	32
Conv3 kernels	64
Conv3 kernels	128
Kernel size	3x3
Activation	ReLU
Kernel Initializer	Glorot normal
Padding	Zero padding
Max Pooling	2x2
FC1 neurons	32
FC2 neurons	16
Optimizer	Adam*

*lr=0.001, fi1 = 0.9, fi2 = 0.999, ” = $1x10^{-7}$

Table 5: Hyperparameters, Model 3

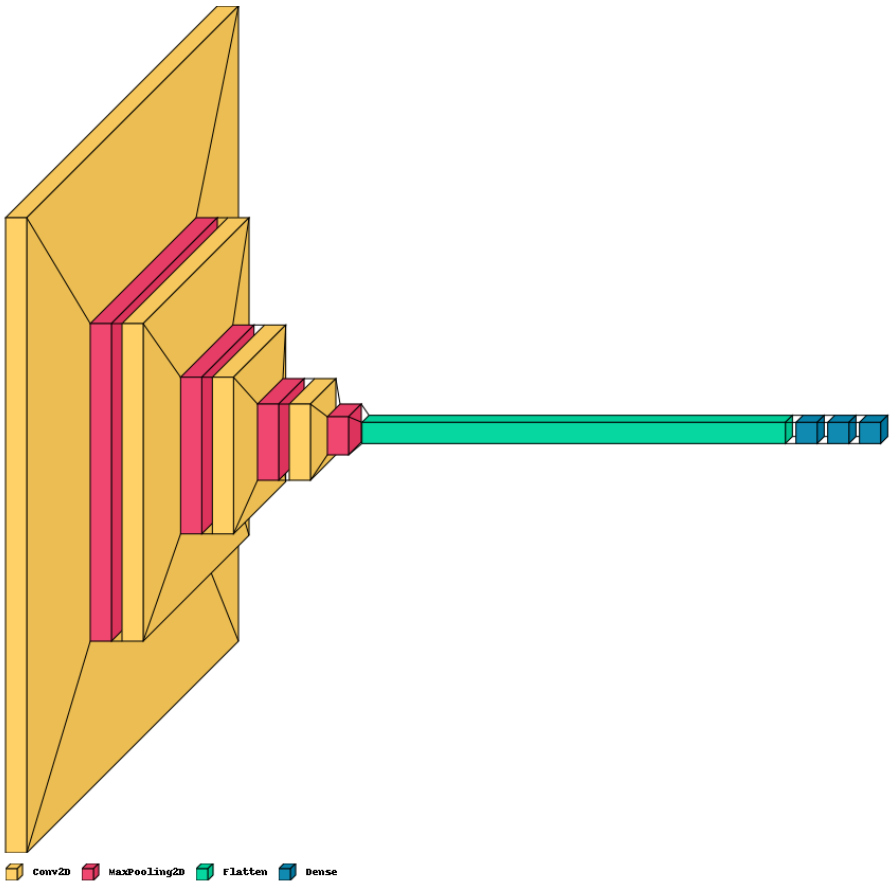

Figure 5: Architecture, Model 3

4.2 Training and Validation Losses

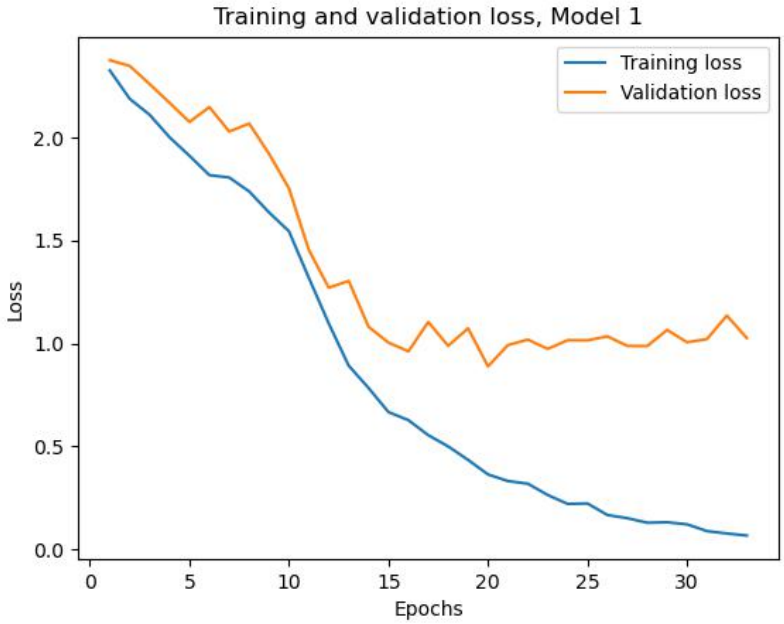

Figure 6: Training and Validation loss per epochs, Model 1

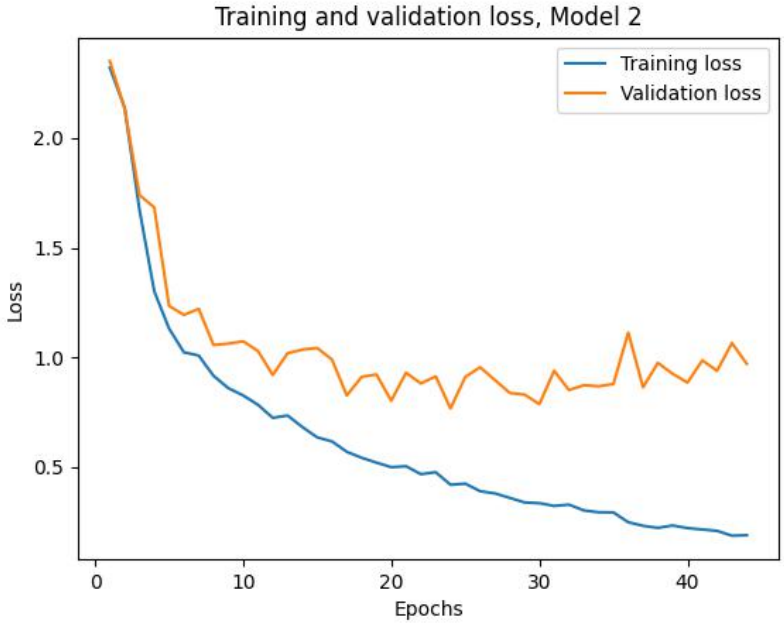

Figure 7: Training and Validation loss per epochs, Model 2

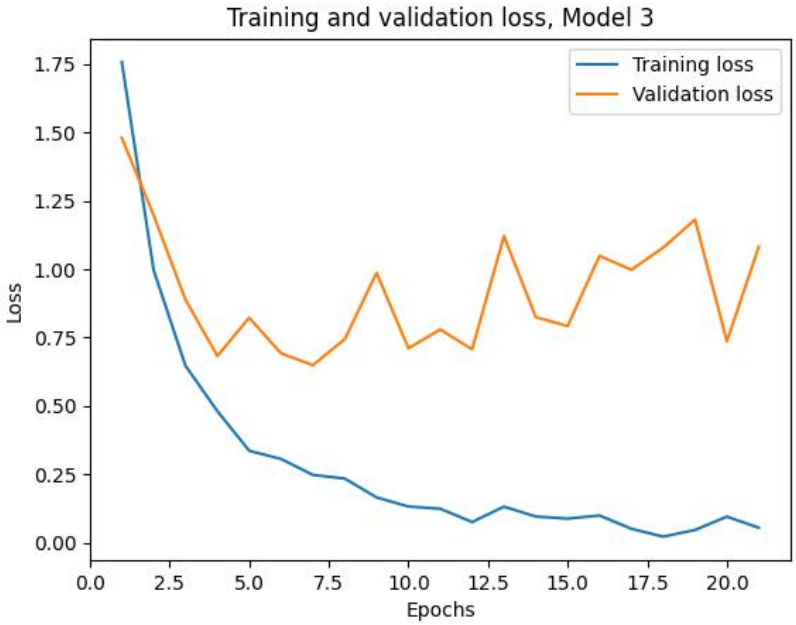

Figure 8: Training and Validation loss per epochs, Model 3

Data	Validation Loss
Model 1	1.037
Model 2	1.048
Model 3	0.914

Table 6: Average Validation Loss

As seen in **Table 6**, the model with the best performance on the validation set is **Model 3**. Next, a risk estimate and classification scores will be provided for the selected model.

4.3 Risk estimates for selected model

As seen below, the loss of the testing set is lower than in the validation set for **Model 3**, this is a puzzling result as both the test and validation sets have the same number of images, morover, they're balanced throughout the categories.

Data	Validation Loss
Train loss	0.295
Validation loss	0.914
Test loss	0.519

Table 7: Average Validation Loss

Table 8 shows precision and recall for **Model 3** are around 0.85 as well as its F1-score. The challenging classes for the model are *Pongamia Pinnata*, scoring 0.57, *Gauva* with a score of 0.73 on the same metric and *Jamun* with a score of 0.77. Besides these two categories, all the rest had a F1-score of at least 0.8.

Plant type	Precision	Recall	F1-score	Support
Alstonia Scholaris	0.89	0.80	0.84	10.00
Arjun	0.77	1.00	0.87	10.00
Basil	1.00	0.80	0.89	10.00
Chinar	0.91	1.00	0.95	10.00
Gauva	0.67	0.80	0.73	10.00
Jamun	0.62	1.00	0.77	10.00
Jatropha	1.00	0.90	0.95	10.00
Lemon	0.80	0.80	0.80	10.00
Mango	1.00	1.00	1.00	10.00
Pomegranate	1.00	0.80	0.89	10.00
Pongamia Pinnata	1.00	0.40	0.57	10.00
accuracy	0.85	0.85	0.85	0.85
macro avg	0.88	0.85	0.84	110.00
weighted avg	0.88	0.85	0.84	110.00

Table 8: Classification Report, Model 3

5 Conclusions

With the objective of testing an algorithm that could classify plant leaves accurately and could be capable to process thousands or even millions of inputs, a Convolutional Neural Network algorithm was proposed, and three different variants were tested. The core ideas behind the architecture come from the literature and the vast amount of resources available in the internet, while the parameter tunning strategy and fine details are inputs from the author.

As seen in the code and by the nature of Neural Networks, scaling these algorithms is as easy as saving more pictures of leaves in the corresponding folders for training, validation, and testing. The algorithm will run independently and does not need to be 'tweaked', unless a very big amount of data should be processed, for which case a parallelization method should be implemented, maximizing computing capacity available through a GPU.

Of course, this exercise opens several different paths for hypothesis and further testing. Improvements could come from adjustments to hyperparameters as obvious as the number of epochs or patience factor, or from slight changes in the learning rate of the optimizers. A first improvement should come from the input data, having a balanced dataset in all categories would ease the interpretability of results and would guarantee a 'fair' learning phase for the algorithm. Either way, room for imrpovement is large, and a sound strategy could be to create a network that learns slower, but steadier, starting from **Model 3**, which registered the best results of all tested algorithms.

Factors that seemed to have positive results are: zero padding, Adam optimizer and no dropout layer. It is not clear if kernel initializer should be He uniform or Glorot normal. Also, four convolutional layers seemed to improve results when compared to three convolutional layers architecture. A puzzling pattern seen during testing for **Model 3** was test loss score lower than the validation loss score,

Bibliography

References

Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets* (pp. 267–285). Springer.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Wu, P., & Qian, Z. (2021). Leaf classification based on convolutional neural network. *Journal of Physics: Conference Series*, *1820*(1), 012161.