# Deep Neural Networks for Pattern Recognition

CS39440 Major Project Report

Author: Mihael Gubas (mig34@aber.ac.uk)

Supervisor: Chuan Lu (cul@aber.ac.uk)

5th May 2023

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree in Computer Science (G400)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

## Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name ……Mihael Gubas……

Date ……24th April 2023……

## Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name ……Mihael Gubas……

Date ……24th April 2023……

## Acknowledgements

I am grateful to my supervisor Dr Chuan Lu for overseeing my project and guiding me through the research and providing assistance whenever I needed.

I'd like to thank my second marker Dr Natthakan Iam-On for providing feedback on my mid-project demonstration which helped me in seeing which areas needed more development.

Lastly I'd like to thank my wife for her constant support throughout the making of this project.

## Abstract

There are over 380,000 species of vascular plants known to science, and an estimated 70,000 species of flowering plants not yet described. Out of those 70,000, half could already be collected and preserved in herbaria without yet being described, waiting an average of 35 years for detection and description from the date of collection. This combined with the facts that at least one in five vascular plants is threatened with extinction and the number of taxonomists is dwindling means that there is an urgent need to speed up this process.

Using deep learning models could greatly speed up this process. If a classifier is trained well, it could help taxonomists in classifying both new, and already existing plant species, and it would do so much faster than a human. This project focuses on developing a machine learning model able to classify different genera of the grass family as accurately as possible.

# Contents

# 1 Background

Prior to this project, the author had no experience building and training machine learning models. They had limited machine learning knowledge from a third-year module where the basics of neural networks and machine learning concepts were taught.
Although with no practical background, they have always been fascinated by neural networks and AI, especially computer vision. Ever since they were first introduced to the concept of AI and machine learning, its potential to change the way we think about problem-solving and decision-making has captivated them. This passion for AI and its impact on our world has driven the author to delve deeper into the field and contribute to its ongoing evolution.

## 1.1 Taxonomy and herbaria

Herbaria are enormous primary data repositories that have helped us preserve information about biodiversity for the last 500 years. If documented correctly, each specimen gives a breadth of information, such as phenotype, genotype, morphology, distribution and geographic occurrence etc. This context is invaluable for advancing our knowledge and supporting research in botany, biology, ecology and even medicine.

Plant taxonomy is the science of classifying and naming plants, and it is a very long and slow process. It currently depends on an ever-shrinking number of taxonomists with broad plant identification skills to recognize new species, who are in constant demand for their skills [1]. There are over 380,000 species of vascular plants known to science, and an estimated 70,000 species of flowering plants not yet described. Out of those 70,000, half could already be collected and preserved in herbaria without yet being described, waiting an average of 35 years for detection and description from the date of collection [2]. This combined with the facts that at least one in five vascular plants is threatened with extinction and the number of taxonomists is dwindling means that there is an urgent need to speed up this process.

Machine learning can play a significant role in plant taxonomy by automating and improving the classification process. A machine learning model can be trained on a large dataset of plant images and could then help with the classification of those plant species. It can analyse an enormous amount of data in a short amount of time whereas it would take a human significantly longer to do the same. A machine learning model can also automatically detect and quantify important features in plant images such as leaf shape, the colour of the flower or the branching patterns. This feature extraction is the heart of classification and it can be used to not only classify plant species, but also detect different types of plant diseases, or be used in further research [3].

## 1.2 Research

Preparation for this project included lots of documentation and research reading. Pytorch website was also used extensively as there is a vast amount of resources on the website from model architectures to data processing and model building. The following areas were researched:

- Python + Pandas and Matplotlib
- Pytorch

- Neural networks and convolutional neural networks
- Network architectures
- Kaggle
- Data processing

### 1.2.1 Python and Matplotlib

Although the author already possessed some knowledge of basic Python, they hadn't used Python in a while, so more practice was required for this project. They had to relearn Pandas and NumPy, creating classes, handling lists, tuples, arrays etc.

Matplotlib was also used to visualize data, plot results and provide a visual representation of various charts and models' performances. Most of matplotlib was learned during later sprints when the data had to be plotted and visualized.

### 1.2.2 Pytorch

PyTorch is an open-source library, used extensively in machine learning, offering a versatile and proficient toolkit for deep learning, scientific computing, and artificial intelligence research. Mainly designed for Python, PyTorch also extends support for C++ and several other programming languages.

Pytorch's website proved to be an excellent source of information due to its comprehensive documentation about different models, tensor operations, optimization algorithms and various others. Moreover, the website houses an extensive array of tutorials and guides for the development of machine learning models. It was used to gain a broad understanding of data processing and augmentation, model building, evaluating etc.

### 1.2.3 Neural networks and convolutional neural networks

An artificial neural network (ANN) is a computing system which is inspired by the biological neural networks in human and animal brains. An ANN is based on a collection of connected nodes called artificial neurons that are connected with each other.

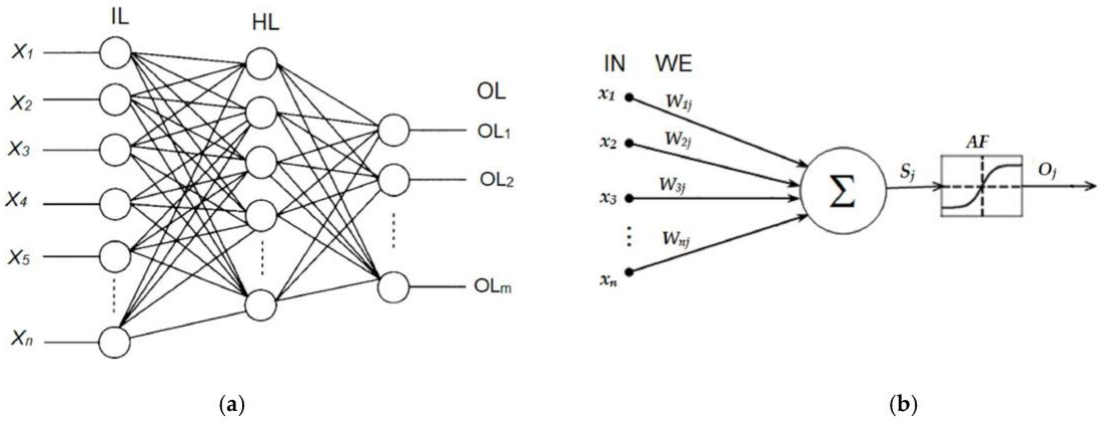


*Figure 1 - A representation of an artificial neural network and a single neuron*

The input layer consists of a number of nodes where each node represents a single feature or data point from the input. The number of nodes depends directly on the dimensionality and the number of features of the input data. For example, if a coloured image was desired to be used as input of the network, of the size 20x20 pixels, that model's input layer would consist of 20x20x3 nodes. We multiply it by 3 because each pixel has three colour channels: red, green and blue.

The hidden layers are the layers between the input and the output layers. An ANN can have one or more hidden layers depending on the complexity and the size of the model. Each node in the hidden layer performs a weighted sum of its inputs, applies a bias and then passes the result to the activation function such as sigmoid, hyperbolic tangent or Rectified Linear Unit (also called the ReLU) function. This non-linearity introduced by the activation functions allows the model to learn complex patterns and relationships in the data.

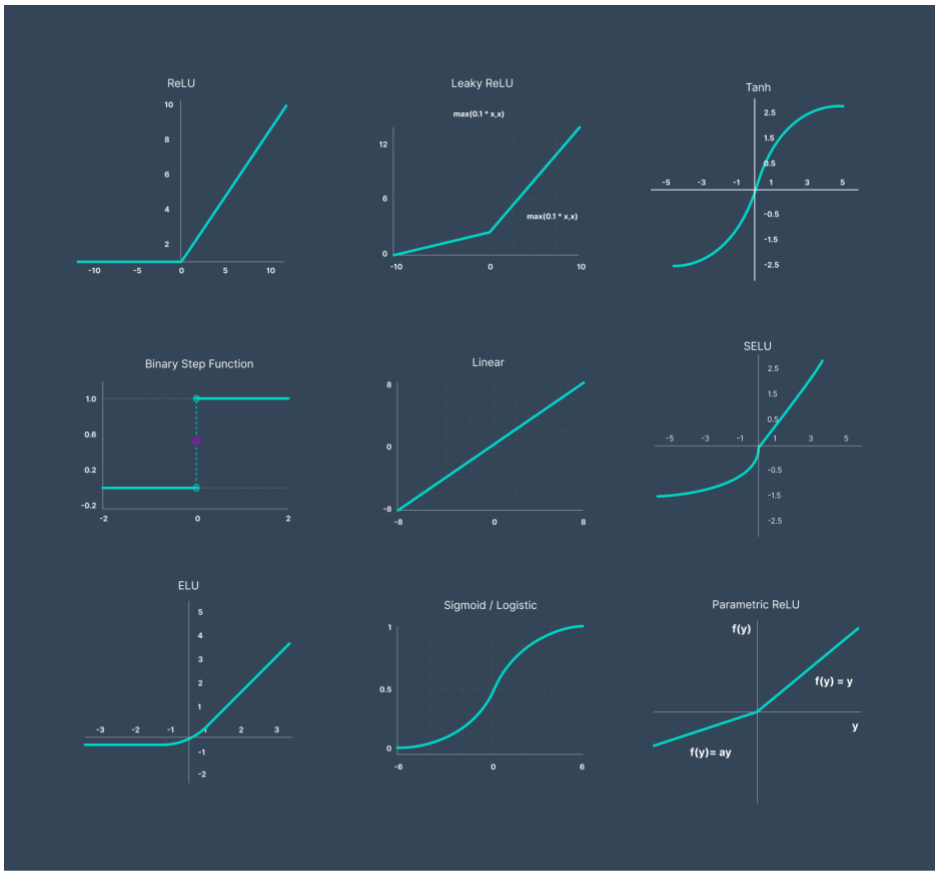


*Figure 2 - Different types of activation functions*

The output layer is the final layer and it produces the network's prediction or classification. The number of nodes in the final layer is determined by the number of classes in our problem. For example, a binary classification model will have 2 output nodes, and in the case of this project, the model had 158 output nodes due to the 158 different classes. The output layer usually uses a different activation function compared to the hidden layers, most commonly, the Softmax function [4].

A convolutional neural network is a neural network that uses an operation called convolution instead of matrix multiplication in at least one of the layers [5]. More on specific CNN architectures will be discussed later.

### 1.2.4 Network architectures

There are many different types of CNNs, and all of them have their own unique designs, architectures, and features. Researching different types helped in coming to the choice for the models that will be used for testing.

### 1.2.5 Data processing and augmentation

Before the model could be built, the data required some organization. For that, the author had to familiarize themselves with Pandas NumPy and tensors. The data had to be sorted, filtered, displayed etc. Data augmentation was also used for testing and the Pytorch library proved useful for this.

## 1.3 Problem analysis

Prioritizing an initial research phase was necessary, given the limited experience with this type of project. Substantial effort was devoted to gaining a comprehensive understanding of the entire model development process which involves aspects such as data processing, model selection, training, tuning and evaluation. This first step contributed to the development of a more informed and effective approach to addressing the problem at hand.

After initial analysis and research about model development, four main areas were discovered to be the main focus of the development process. They are listed in the order of development:

- Exploratory data analysis
- Data processing
- Model selection
- Evaluation metric selection
- Training and evaluation
- Fine-tuning
- Testing

Once the main areas of development were defined, they were tackled mostly in order of the model development.

### 1.3.1 Labels

During analysis, labels came to be understood as well. This dataset has 3 levels of labels. The top level is the family, the second level is the genus and the third level is category. This means that a family has multiple genera, while a genus has multiple categories. It is important to understand this structure to be able to understand the results better.

## 1.4 Project objective

The goal of this project at first was to develop a model capable of classifying the whole herbarium 2022 plant dataset as accurately as possible. However, as the development moved forward, the dataset was instead shrunk down to one family of plants, the Poaceae.

## 1.5 Process

The beginning of the project was mostly characterized by research, note-taking, and experimenting with the code, learning through trial and error. This is because the whole scope and extent of the software elements required for the project were not known from the start, combined with a lack of previous knowledge of deep learning or experience with similar projects.

As the beginning was mostly reading documents, researching, and going through tutorials, it was important to keep track of the new things learned. For that, a notebook was used and kept as a reference point for things that were learned. Through trial and error, a more structured way of approach was embraced, one that uses certain aspects of Scrum and Kanban.

### 1.5.1 Kanban board

A Kanban board creates a visual workflow of the tasks at hand. Rather than a Kanban program or website such as Trello, a Kanban-like board was used with Post-it notes that helped keep track of current tasks at hand. The board consisted of three columns with one being the current task with a supporting column next to it where relevant knowledge and information relating to the task was put. The third column consisted of tasks that were meant for future work but were be discovered sooner.

### 1.5.2 Scrum

Scrum is an iterative and incremental Agile project management framework. The workflow in Scrum revolves around timed iterations - sprints, which typically last between 1 to 4 weeks. The sprint duration for this project was 1 week. Each week a set of tasks was prioritized aiming to be finished in the sprint. There was a review once in the middle of the sprint and once at the end of the sprint. These reviews acted like breaks where progress would be reviewed and decided if the tasks or the approach should shift towards something else.

### 1.5.3 Why this process was chosen

This approach was chosen for several key reasons. Firstly, it ensured that the current task at hand was always well-defined and organized, resulting in a clear focus on priorities. Secondly, it prevented the risk of becoming excessively absorbed in learning new concepts without making tangible progress towards the project's objectives. Finally, by incorporating aspects of both methodologies, it reduced the likelihood of developing tunnel vision by maintaining a broader perspective on the project's overall goals and milestones. This agile, combined approach allowed for continuous improvement and adaptation to any emerging challenges encountered throughout the project's development.

# 2 Experiment Methods

## 2.1 Hardware

The most significant hardware component for this project was GPU memory.

When training a deep learning model, the model learns to identify patterns and make predictions by adjusting its internal parameters with the given training data. This involves using mathematical operations and transformations on the data. The main parts of the training process are forward propagation, loss computation, backpropagation, and optimization. During training, a large number of mathematical operations, such as matrix multiplications, convolutions, and activation functions are performed.

When training a model using a CPU, the Central Processing Unit executes the code and performs the required computations sequentially. While CPUs are designed to handle a wide range of tasks and can execute complex instructions, they typically have a limited number of cores that can execute tasks simultaneously. As a result, the training process can take much longer, especially for large deep-learning models and datasets.

A GPU (Graphics Processing Unit) provides significant advantages over a CPU due to its architecture and capabilities. The main benefits of using a GPU during model development are faster training and better data handling.

GPUs are designed for parallel processing, meaning they can perform multiple mathematical operations simultaneously. They contain a large number of smaller cores that work together to process large amounts of data quickly [7]. This proves incredibly useful when training deep-learning models because of the repetitive nature of operations performed. Because of this, a GPU will speed up the training process several-fold, giving more time to focus on experimenting, fine-tuning etc. Also thanks to their higher memory bandwidth, a GPU can take a lot of data from memory simultaneously, making the data transfer between the GPU and processing units faster.

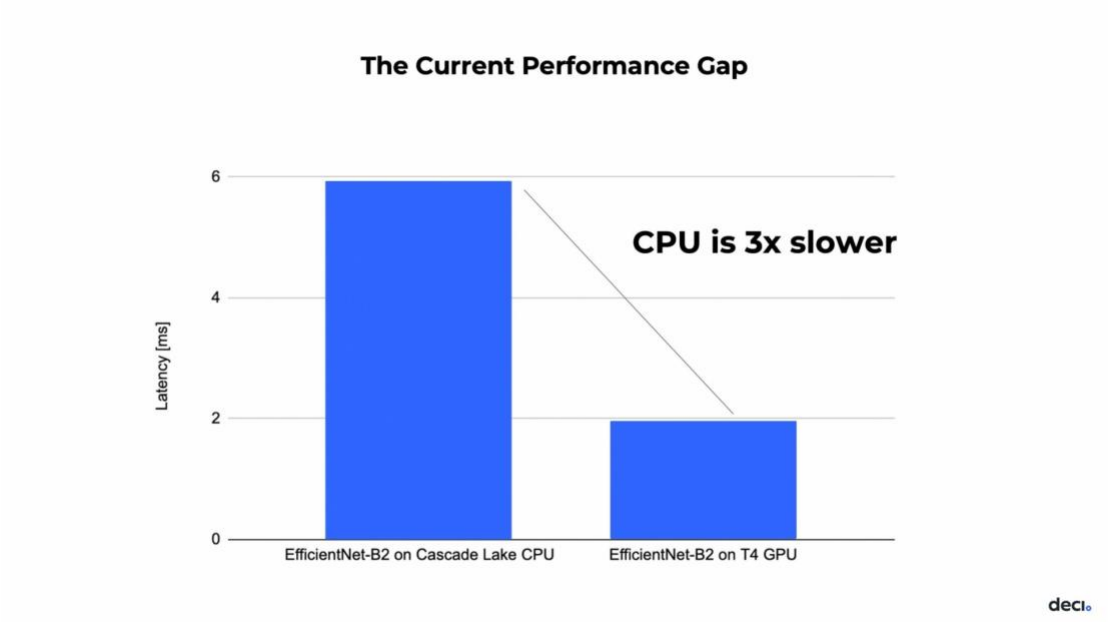


*Figure 3 - EfficientNet-B2 forward pass speed on a CPU vs GPU*

## 2.2 Software

### 2.2.1 PyTorch

Choosing a machine learning library that the rest of the project will be built on was one of the first steps. There was a choice between the most widely used frameworks: TensorFlow, PyTorch and Keras [9].

Pytorch strengths:
- Dynamic computation graph (eager execution), making it easier to debug and develop models with complex architectures.
- Strong support from the research community, which means state-of-the-art models and techniques are often available in PyTorch first.
- Extensive documentation and a large number of tutorials, making it beginner-friendly.
- Tight integration with other libraries in the PyTorch ecosystem, such as torchvision, torchaudio, and torchtext, simplifying tasks like data loading and preprocessing.

Pytorch weaknesses:
- Less mature ecosystem compared to TensorFlow. This means that some specialized tools are not available

TensorFlow strengths:
- Static computation graph (although eager execution is now available), which can lead to better performance optimizations for some use cases.
- Large industry adoption, making it more likely to encounter TensorFlow-based models and tools in commercial settings.
- Strong support for deployment in various environments, including web, mobile, and embedded systems, through TensorFlow Lite, TensorFlow.js, and TensorFlow Serving.
- Integration with Keras as the high-level API, making it easy to build and experiment with models.

TensorFlow weaknesses:
- The static computation graph can make debugging and developing complex models more difficult.
- Steeper learning curve compared to PyTorch, especially for beginners.

Keras strengths:
- High-level API that simplifies model building and training, making it an excellent choice for beginners and for rapid prototyping.
- Built on top of TensorFlow (and previously supported Theano and CNTK), allowing users to take advantage of TensorFlow's capabilities and tools.
- Large number of pre-built layers and components, reducing the amount of code needed to build custom models.

Keras weaknesses:
- Limited flexibility and control compared to lower-level frameworks like PyTorch and TensorFlow, making it more challenging to build complex or highly customized models.
- Being a high-level API, it may not provide the latest cutting-edge features as quickly as lower-level frameworks.

Taking all of these points into account, Pytorch was chosen as the machine learning framework for producing the model. Taking the characteristics of all the frameworks into consideration, the reason PyTorch was chosen was due to the pure abundance of documentation, resources, and support available for it. As this was the first project of its kind for the author, these were deemed the most important.

Having torchvision as a part of its library was also a big advantage as torchvision provides lots of useful features such as the transform method and direct imports of models.

### 2.2.2 Kaggle

Kaggle is an online platform for machine learning, data science and research. The platform hosts lots of datasets that users can freely access for their projects. It also hosts machine learning competitions for teams and individuals to tackle different machine learning tasks. Additionally, Kaggle provides an environment for users to write code in Python using Jupyter Notebooks, which are used for developing machine learning models. This project was also written in one of those notebooks. Kaggle also provides the user with free GPU resources that can be used on model training, speeding up the process severalfold.

Next to Kaggle, there was also a choice to use Google Colab as the hosting platform for the model. Next to Kaggle, Google Colab has one advantage which is the ability to purchase extra GPU resources, providing a longer usage than Kaggle's free 30 hours of GPU time per week.

However, to use Google Colab, one needs to first upload the dataset to the notebook, whereas with Kaggle, the data is already hosted on the website. This would mean that the data would have to be uploaded to the notebook every time a session is started. It could potentially be uploaded to Google Drive and imported from there, but when this was attempted, Google Colab wouldn't read the data properly and there would be missing images, making Kaggle the obvious choice for the project.

### 2.2.3 GitHub

In order to keep track of all the updates and changes to the software, a version control tool was necessary. For this project, GitHub was selected.

GitHub is a free, online software development platform. It is used for tracking, storing, and collaborating on projects [11]. Using GitHub has allowed to view all the changes made to several notebooks used during this project and has proved extremely useful in many situations when previous versions had to be checked.

## 2.3 Experiment overview

The experiment overview entails a high-level summary of the model development process, focusing on the key steps involved in selecting, testing, and fine-tuning the models. The primary objective was to develop a deep learning model with the best possible performance in terms of accuracy and generalization.

Firstly, a model was selected. The choice was made based on several factors. Once a candidate model was identified, it was tested using a set of predefined hyperparameters to measure its initial performance.

Following the initial testing, the model was fine-tuned by iteratively adjusting its hyperparameters, with the aim of improving its performance. This fine-tuning process involved systematically exploring various combinations of hyperparameters to identify the optimal configuration.

## 2.4 Model development

### 2.4.1 Exploratory data analysis

Exploratory data analysis is the analysis of data with the goal to investigate the dataset and summarize its main characteristics [6]. EDA was crucial in gaining insight into the data and setting a baseline of things to focus on.

One of the first things to note is the high imbalance of the dataset, as can be seen in Figure 4. The list of genera on the bottom isn't very clear but it gives the idea of the present imbalance.

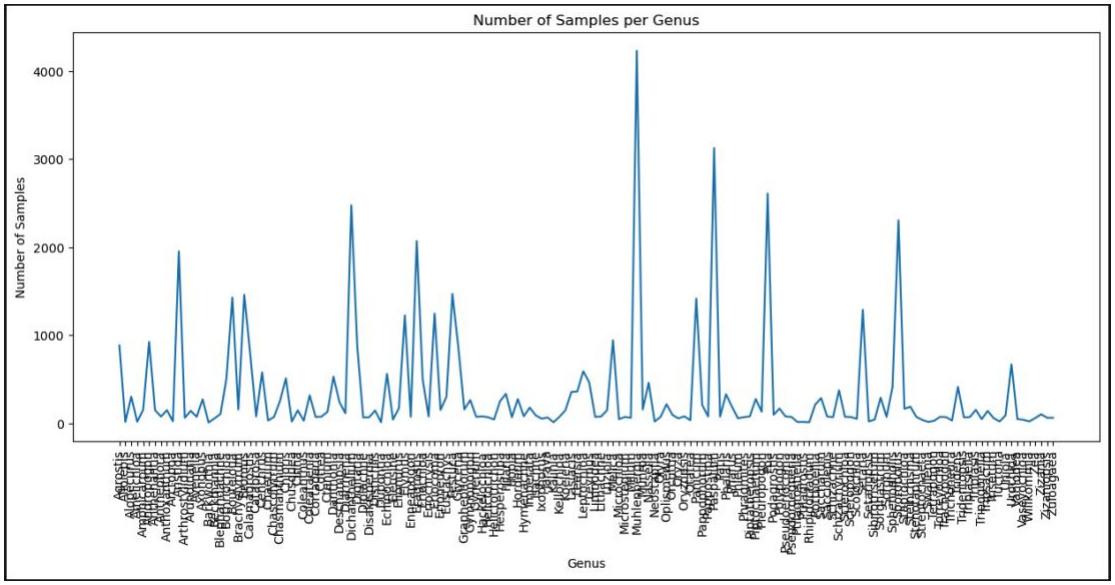


*Figure 4 - A graph showing the great imbalance of the data*

The second thing to note is that the variation within the same taxon (and even species) is very high. There are many plants of the same species or genus that look vastly different, and also plants of different species that look strikingly similar. This especially true for species rather than genera as species have all much subtler differences than genera.

### 2.4.2 Data pre-processing

In the case of this project, the data was already available so the collection of data wasn't necessary. That made it possible to go into data processing directly. Once imported, the data was:
**resized** – depending on the model being tested, but this was usually the size 256x256
**normalized to standard values** – all the images had to be normalized to a consistent range of pixel values. For that, Pytorch's recommended mean and standard deviation values were used
**transformed into tensors** – all the images need to be transformed to tensors to feed into the model
**augmented** – in some of the experiments, data augmentation was applied to the images. Augmentations such as horizontal and vertical flips, rotations, turned to greyscale, colour jitter etc.

The above data processing was achieved using a single transform method in the first stages of model development:

$$Transform = transforms.Compose([$$
$$transforms.Resize((224,224)),$$
$$transforms.ToTensor(),$$
$$transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]),$$
$$])$$

The data also had to be split into training, evaluation, and testing sets. The ratio of 80/20 was chosen as a split between the training and the test to make sure that all the classes are fairly represented in the test set since there were some classes with just a few samples. After this, the remaining of training data was further split 80/20 one more time with 20% going to the validation set.

### 2.4.3 Model selection

There are a very large number of possible models that could be tested, and within the timeframe of this project, it was not possible to test them all. That mean there had to be some kind of criteria for which models are going to be tested.
Since the start, the emphasis placed on those model architectures that had demonstrated success in prior research with similar data [10]. Following this, the following architectures were selected: ResNet, DenseNet and ResNeSt.

### 2.4.4 Training, validation and fine-tuning

When a model was selected, the next step was training. The training consisted of two parts: training and validation. The data had previously been split into three sets, namely: training, validation and testing sets. The model is first trained on the training dataset, and after each epoch, it is evaluated with the validation set.

Fine-tuning was one of the most important steps of the training process. After each training session of several epochs, the model's hyperparameters would be fine-tuned. Hyperparameters such as learning rate, batch size, model depth, early stopping etc. Fine-tuning is an iterative process, built upon trial and error. It is also true that not every single combination of hyperparameters can be tested in a reasonable amount of time, so, when an increase in the F1 score was observed after a change in a hyperparameter, the previous value was not taken into consideration anymore.

### 2.4.5 Evaluation metrics

For evaluating the model's performance, the F1 score was chosen as a metric.

$$F1 = 2\frac{Pre \times Rec}{Pre + Rec}$$

Where Pre means precision and Rec means Recall.

Precision is the ratio of true positive predictions to the total number of positive predictions made by the model.
Recall is the ratio of true positive predictions to the number of actual positive instances in the dataset.

F1 was chosen as the evaluation metric because it is particularly useful when there is an unequal distribution of classes in question, and when false negatives and false positives are

more important [8]. In the case of this project, a false negative could potentially classify a new plant species as an already existing one, which could be detrimental to the whole purpose of using deep learning for assisting in taxonomy.

#### 2.4.6 Testing

After a satisfactory validation f1 score was reached, the model was tested on a separate test set with data it has not yet seen. The score was obseerved and compared to the previous combination of hyperparameters.

# 3 Software Design, Implementation and Testing

## 3.1 Software building process

As mentioned previously, the approach to this project was firstly purely based upon research, and then it developed into a Scrum-like system. After the initial research phase, several sprints were used to develop the code to support the model development.

### 3.1.1 Sprint 1 – Sprint 2

The main focus of the first two sprints were pandas. Getting the data imported and turned into a dataframe was the first step. Learning to manipulate and represent the data properly helped to gain an understanding of what type of data will be used throughout the project. Dataframe manipulation, including adding columns, formatting the dataframe etc.

Throughout this process, the complexity of the tasks increased, and a more extensive knowledge of Python became necessary. To address this, various educational resources such as online videos and tutorials were utilized to reinforce and expand Python programming skills

### 3.1.2 Sprint 3

In another phase of the project, the focus shifted towards acquiring proficiency in the use of matplotlib, a powerful visualization library in Python. This skill was essential for effectively displaying the data, plotting the various classes, and visualizing the distribution of images across all classes. By learning to create meaningful visual representations, a more in-depth understanding of the underlying dataset was achieved. Additionally, the ability to display images using matplotlib proved valuable, as it provided a direct window into the data the model will be classifying.

Along with plotting, this is the sprint where the data was made ready for the first tests. All the required values were there for each image and what was left was to develop the model.

### 3.1.3 Sprint 4

For this sprint, the primary focus was on selecting the most suitable machine learning framework for the model development. After a thorough analysis of available options, PyTorch was selected and with that, the first parts of model development started.

With PyTorch selected, the attention shifted to learning how to manipulate and prepare the data effectively using the "transform" function, and familiarizing with different transformations. This step enabled the proper preprocessing and transformation of the input data to meet the requirements of the models being tested.

Furthermore, a GetData class was created, which served as a custom data handling class responsible for returning both the image and its corresponding label to the model. This class enabled the process of feeding data to the model, ensuring efficient and accurate training and evaluation of the chosen models.

### 3.1.4 Sprint 5

This sprint focused on designing the training and evaluation functions.
The training loop was designed to iterate through the dataset, feeding input data to the model and computing gradients to update the model's parameters

Simultaneously, the evaluation method was developed to assess the model's performance on a separate validation set. This allowed for monitoring the model's generalization capabilities and making informed decisions about fine-tuning and model selection. Metrics, such as accuracy, loss, and F1-score, were employed to measure the model's effectiveness during both training and validation.

This sprint also marked the beginning of testing the models on the datasets, with the first set of tests being done on a small dataset with only two genera.

From this point forward, not much regarding software was developed. The rest of the work went towards testing and experimenting which will be discussed in the following sections.

#### 3.1.5 Sprint 6 and after

Once the training and evaluation functions were functional, the focus shifted towards training and fine-tuning various models to improve their performance on the dataset. During this period, several model architectures were explored and fine-tuned to identify the most effective combinations. The objective was to develop a deeper understanding of the models' behaviours and to iteratively refine the model based on the insights gained from each experiment.

The final, and most important measuring variable – the testing loop, wasn't implemented until much later in the project. This was due to an oversight in the development and an insufficient understanding of the training process. However, an overview of its functions will still be given here.

##### 3.1.5.1 Train and evaluation functions

The first inspiration for the training function was found on PyTorch's own website describing a tutorial on how to train a classifier [12]. As previously stated, Kaggle hosts competitions on their website, and participants can upload their work to compete. Some make this code publicly accessible for others to use. A lot of time was spent studying already existing training and evaluation methods to gain a better understanding.

At first, a simple, basic training function was built. The variables for accumulating training accuracy and loss were initialized, and then the data loaders were iterated, moving the images and the labels to the device (in this case the GPU). The gradients were also reset to zero before each new batch to avoid gradient accumulation (although there are some benefits of implementing gradient accumulation, the main one being less memory usage). Following this, a forward pass is initialized, and the loss is calculated using cross-entropy loss. Once that was completed, the backwards pass was initialized and the accuracy of the batch was calculated.

As for the evaluation function, The function sets the model to evaluation mode, iterates through the validation dataset in batches, and computes the model's predictions for each batch without tracking gradients. It then calculates the accuracy and loss for each batch, updates the cumulative evaluation accuracy and loss, and stores the ground truth and predicted labels. Finally, it computes the per-class and macro-averaged F1 scores and returns the average accuracy, average loss, and F1 scores.

### 3.1.6 Testing loop

In the testing loop, the model is first transferred to the appropriate device (in this case, the GPU) and set to evaluation mode to ensure the correct behaviour of layers. The loop iterates through the test dataset, disabling gradient computation to prevent weight updates. For each batch of data, the images and labels are transferred to the device, and the model generates predictions.

The accuracy of the model's predictions is calculated by comparing the predicted labels against the ground truth labels, and this value is accumulated throughout the loop. Additionally, true labels and predicted labels are stored in lists to enable the later computation of the F1 score.

After completing the loop, the final test accuracy is calculated by dividing the accumulated accuracy by the number of batches in the test loader. A classification report, which includes F1 scores for all classes, is generated using the stored true and predicted labels. The report is printed, displaying the F1 scores for each class, and the average F1 score across all classes.

## 3.2 Design

### 3.2.1 A more detailed training function description

Through the development of the project, the training method evolved to involve some additional steps. In its final stage, the method included: mixed precision training, tqdm, and F1 scores for all the classes.

Mixed precision training is an approach that aims to optimize memory consumption during the training process by performing the forward pass using a lower precision number, typically a 16-bit float number (also called “half-precision” [20]) instead of a 32-bit. Subsequently, during the backward pass, the gradient computations are also carried out with reduced precision. However, to ensure the model's accuracy is preserved, the weight updates are conducted using the higher precision format, float32. This makes the training faster, reduces memory usage, and preserves model accuracy [13]. It is implemented with the following line of code:

$$with\ torch.cuda.amp.autocast(enabled = True):$$

The reason for the effectiveness of this method comes from the fact that modern GPUs can process lower-precision data types more quickly and with less memory usage than higher precision data types.

Tqdm is a Python library that provides a visual progress bar for loops and iterables. It is designed to give users real-time feedback on the progress of a loop, making it easier to monitor the execution of lengthy operations. This was used for both training and evaluation, and testing[14].

### 3.2.2 Training loop

In this code segment, the model is trained and evaluated using a loop that iterates over a predefined number of epochs. A gradient scaler is initialized to enable mixed precision training:

$$scaler = torch.cuda.amp.GradScaler(enabled = True)$$

Throughout the training process, training and validation F1 scores are collected and stored in their respective lists. For each epoch, the model is trained on the training dataset using the train function and subsequently evaluated on the validation dataset using the evaluate function. The training and validation F1 scores, both average and per class, are appended to their respective lists for future analysis.

Upon completing each epoch, the training and validation accuracy, loss, and F1 scores are displayed, providing insights into the model's performance at different stages of the training process. Furthermore, the F1 scores per class for both training and validation sets are printed, offering a more granular view of the model's performance across different classes.

### 3.2.3 Alternative design

There was a possibility to place all the code from the training function and the evaluation function into one large, outside loop which iterates over the epochs. Instead of this, the current design is made up of 2 functions: training and evaluation, and one loop outside of those functions which iterates over epochs and calls the functions, called the training loop (keep in mind, the training and evaluation functions both have loops inside of them too).

The reason this approach was selected was for several reasons:

- training and evaluation functions were learned about separately
- it allowed for both functions to be developed separately from each other
- the code is more readable

## 3.3 Model architecture

Out of the three model architectures discussed, ResNeSt, ResNet and DenseNet, the one that proved to be the best suited for this task was ResNeSt. In this section, the architecture of this model will be discussed, while the results and comparisons to other models will be shown in later sections.

### 3.3.1 Convolutional neural network

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing grid-like data structures, such as images, where local spatial correlations are of primary importance [17].

CNNs are characterized by their use of convolutional layers, which employ a series of learnable filters to perform local feature extraction. These filters are convolved across the input image, enabling the network to recognize patterns, such as edges, corners, and textures. The use of shared weights in the convolutional layers reduces the number of parameters in the network, making CNNs more computationally efficient than their fully connected counterparts.

In addition, CNNs frequently use pooling layers, which help shrink the size of feature maps by combining nearby information. This process makes the network more robust to small shifts in the input and lowers the computational demands of the model. At the end of the network, one or more fully connected layers are included to handle tasks like classification or regression [18].

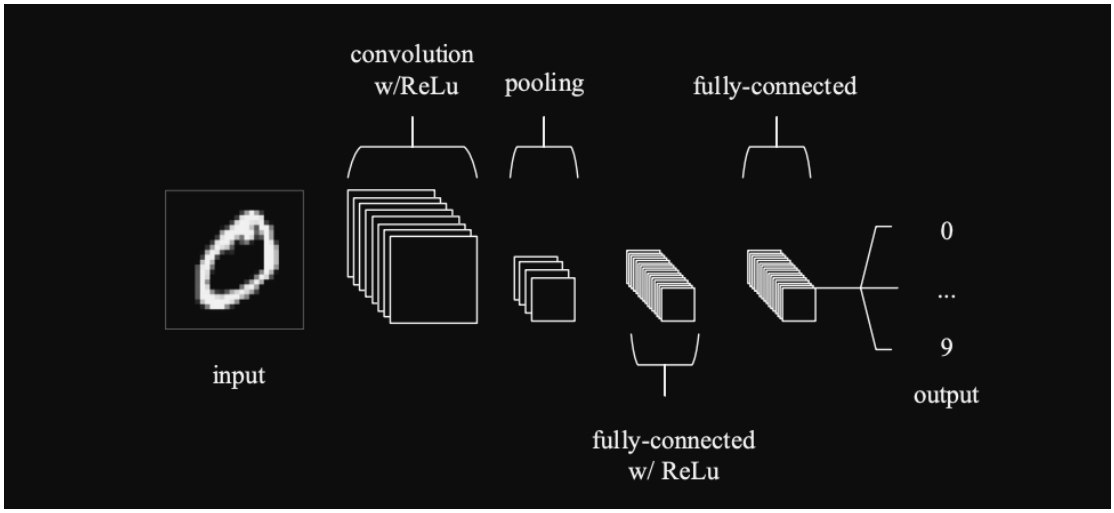


*Figure 5 - A simple CNN comprised of five layers*

The above image is interpreted as follows:

Firstly, the input image is passed to the network and it reaches the convolutional layer. There, convolutional filters are applied to the image to learn the patterns and a ReLU activation function is used to introduce non-linearity. Non-linearity enables the representation of complex and non-linear relationships between the input and the output. Without non-linearity, a CNN would only be able to model linear relationships, greatly limiting its capacity [19]. A representation of the ReLU function can be seen at the beginning of the paper in Figure 2.

Secondly, the pooling layer is responsible for reducing the dimensions of the previous layer by combining the outputs of neuron clusters into a single neuron in the next layer. This makes it less computationally heavy on the device, making the process faster. A representation of this can be seen in Figure 5.

The third layer, fully-connected layer with ReLU function flattens the output from the pooling layer and connects each neuron from the pooling layer to each neuron in this layer. The ReLU function is applied again to introduce non-linearity and learn complex features.

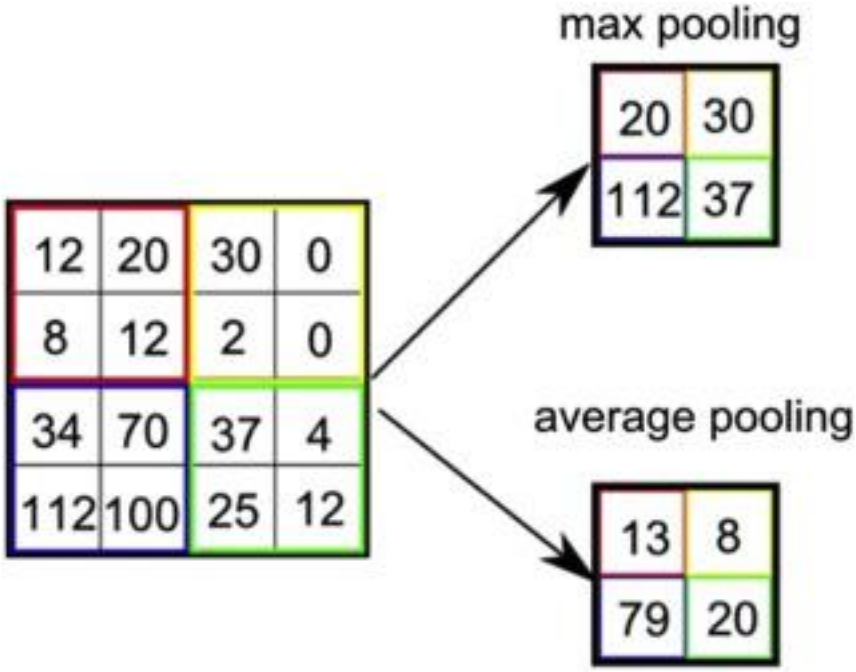


*Figure 6 - The pooling layer. Max pooling and average pooling*

The fourth layer is another fully connected layer which further processes features from the previous layer.

Finally the output layer, which has as many neurons as the number of classes in the classification task, has an activation function such as softmax applied to it to produce class probabilities.

### 3.3.2 Architecture overview

ResNeSt is a modified version of the ResNet architecture, which was designed to improve the performance of deep residual networks in large-scale image classification tasks. The key idea behind ResNeSt is to introduce a feature-map level "split-attention" mechanism in the residual blocks, which enables the network to learn more expressive feature representations [15]. ResNeSt builds upon the foundational principles of ResNet architecture, so it is important to understand ResNet first.

### 3.3.3 ResNet architecture

The Residual Network (ResNet) is a deep convolutional neural network architecture. ResNet addresses the problem of vanishing gradients and degradation in training accuracy as the network depth increases. It achieves this through the use of residual learning, where the network learns residual mappings in addition to the original functions.

Firstly, vanishing gradients occurs because as the depth of the network increases, gradients of the loss function can become very small as they are backpropagated through the network layers. This is due to the repeated application of the chain rule during backpropagation, causing the gradients to be multiplied by small numbers (less than 1). When the gradients become very small (i.e., they "vanish"), the weight updates become negligible, and the network stops learning or learns very slowly. Secondly, accuracy degradation refers to the phenomenon where the training accuracy of a deep neural network starts to degrade (i.e., get worse) as the network depth increases, even when there are no issues with overfitting. Intuitively, one might expect that deeper networks would be able to learn more complex

features and exhibit better performance. However, in practice, deeper networks can sometimes experience a decline in training accuracy due to optimization difficulties arising from the vanishing gradient [16].

ResNet addresses these issues by introducing skip connections, also known as residual connections, which allow the gradients to bypass one or more layers during backpropagation. This helps mitigate the vanishing gradient problem and improve the optimization of deep networks, enabling them to learn more complex features and achieve better accuracy without suffering from degradation [21].

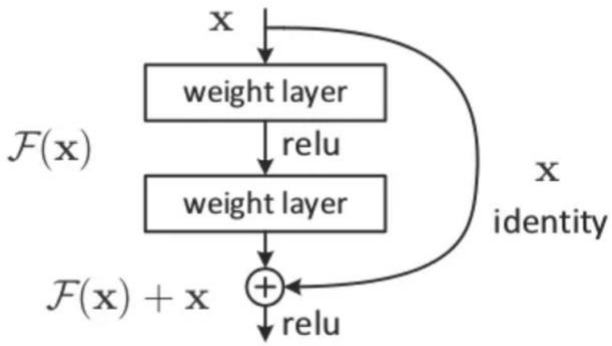


*Figure 7 - A residual block*

### 3.3.4 ResNeSt architecture

As mentioned before, ResNeSt is built upon the architecture of ResNet. It does so by introducing a split-attention mechanism in the residual block. To understand ResNeSt's architecture, one must first be familiar with featuremaps and multi-path representation.

A feature map is the output of a convolutional layer in a CNN. It represents the spatial arrangements of features learned by the network at that specific layer. It is created by applying a set of filters to the input image or the output of the previous layer. These filters learn to detect various features such as edges, textures and patterns, depending on the layer depth in the network. As the network becomes deeper, the features detected by the filters become more complex and high-level [22].

Multi-path representation is a concept in deep learning where the input feature maps are processed through multiple parallel pathways in a neural network. Each pathway focuses on learning different features or patterns from the input, thereby enabling the model to capture a diverse set of features. When the outputs of these parallel pathways are aggregated, the resulting representation is richer and more complex.

The core innovation in ResNeSt is the introduction of the split-attention block. This block consists of a channel-wise split, which divides the input feature maps into different groups, followed by attention mechanisms that operate within each group. The attention mechanism computes weights for different channels based on their importance for the specific input, enabling the network to focus on the most relevant features.

This method captures cross-channel feature correlations while preserving independent representations of the meta structure. This means that as different paths focus on different parts of the featuremap, this architecture can capture cross-path feature correlations while

still preserving the independent representations. A representation of a split-attention block can be seen in Figure 7.

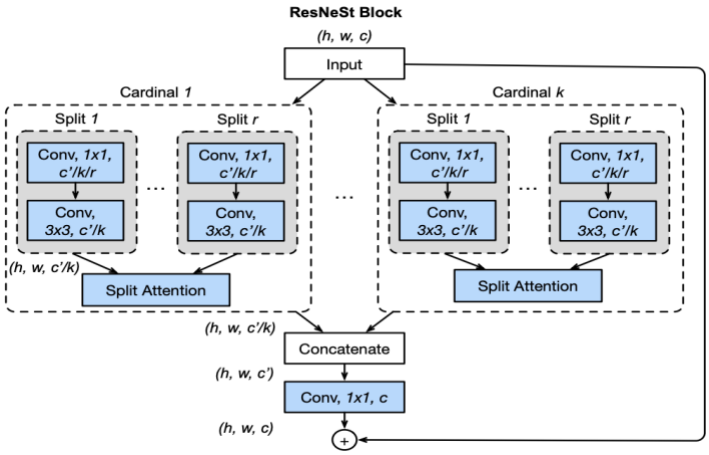


*Figure 8 - A split-attention block*

ResNeSt introduces two new hyperparameters: radix and cardinality. Radix determines the number of splits within each group in the Split Attention block, and cardinality defines the number of groups. By increasing the radix and cardinality values, the model can learn more complex and diverse feature representations, which can improve its performance.

## 3.4 Training and testing

In this section, the stages of model testing and development will be discussed. Bear in mind, these weren't predetermined stages, just the whole development separated into several sections based on the workflow and the turning points in testing.

### 3.4.1 First stage - 2 genera

The first stage of training the model revolved around using only 2 genera, mainly Paspalum and Muhlanbergia. The reason for this was that the experience was very limited at the beginning, and it was not known what to expect from these experiments, so it was decided to run the model with only 2 classes, making the issue a binary classification problem. At this stage, the binary cross entropy loss function was used. After this stage and for the rest of the project, it was cross-entropy loss.

Through EDA, it was discovered that the two genera, Muhlanbergia and Paspalum are the ones with the most samples in the Poaceae family, so they were chosen as a starting point. For this stage, a Densenet169 model was used to conduct the trainings.

#### 3.4.1.1 Issues

The issues in the first stage were mostly all of the same nature. It was usually mistakes in the code or faults in implementation. One of the first errors was a ValueError to do with the dimensionality of the tensors. The input tensor was two dimensional while the output tensor of the model was one dimensional.

Another very persistant error was an IndexError that kept signaling an out of bounds for a certain index. This happened because when the two genera were separated from the main

dataframe, their indices weren't reset. So while the size of the dataframe was 1101, some images had indices well above that number.

After a lot of trial and error, a F1 score of over 90% was reached for the two genera.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.94 | 0.93 | 0.93 | 634 |
| 1 | 0.90 | 0.91 | 0.91 | 467 |
| accuracy | | | 0.92 | 1101 |
| macro avg | 0.92 | 0.92 | 0.92 | 1101 |
| weighted avg | 0.92 | 0.92 | 0.92 | 1101 |

*Figure 9 - Muhlanbergia and Paspalum accuracy*

This score has been achieved with a DenseNet169 model after 10 epochs and a batch size of 128. The default learning rate of 1e-3 was also used. It is worth noting, that a test set was not used, rather, only training and validation sets.

In conclusion, these initial errors played a crucial role in establishing a solid foundation for the upcoming stages of model development. Addressing these issues not only resolved immediate concerns but also contributed to a deeper understanding of the underlying principles.

### 3.4.2 Second stage – Poaceae family and model comparisons

This stage of testing was by far the longest and it is here that three different architectures of models were being evaluated.

#### 3.4.2.1 Issues

The issues discovered during this stage should be mentioned first as they are major ones that have an impact on all of the following tests in this stage.

The first and largest issue was not using the .eval() method when evaluating the model.
When a model is initialized, it is by default put in a training state. This means that whatever data is pushed through the model, the model's parameters will change and the model will try to learn based on that data. This is exactly what is not supposed to happen during evaluation. The purpose of evaluation is to get an insight into the model's training and see if the model is overfitting to the training data or not [23]. If evaluation is done without calling the .eval() method on the model, the model will not act properly, namely batch normalization and dropout regularization.

As an example, let's take the ResNeSt model. ResNeSt uses a dropout layer which randomly sets a fraction of the neurons' outputs to zero at each forward pass, which helps the network to generalize better to unseen data. However, this layer needs to be turned off during classification, otherwise, it would again randomly set some neurons' outputs to zero, introducing randomness into classification and making less accurate predictions.

During most of this stage, all the models that were used had all but the last layer frozen. This was the largest reason for the small accuracy on most of the tests, as no model has gotten above 40% accuracy. The reason for this was a lack of proper understanding of how pre-trained models work and the concept of freezing and unfreezing layers while training a model. Once this was discovered, ResNeSt-101 was tested with more and more layers unfrozen in this order: 1, 2, 10, 15, 20, 50, 80, and all layers. Each time more layers were unfrozen, the accuracy and the F1 score kept growing, so eventually all layers were unfrozen.

Another issue was the lack of using testing sets. At this stage, models' performances were measured based on the validation F1 score and accuracy, and not that of a separate test set, until the last stages. This led to overconfident conclusions about the models' accuracies but did not have a major impact on the end result of the project.

#### 3.4.2.2 Testing

With the limited timeframe for the project and limited GPU resources, it was not possible to test every hyperparameter combination for all the models. Due to this fact, the approach was taken to evaluate the models by focusing on one hyperparameter at a time. When a reasonable range of values for a single hyperparameter was tested on the model (i.e.: a batch size of 32, 64, 128 and256 – typically used batch size values), the best-performing value was taken and the target hyperparameter changed (i.e. to learning rate). This process was then repeated for all three model architectures.

As these tests were conducted without the proper setup of model evaluation, and with no test sets, they will not be counted towards the conclusion of the project, however, due to the late discovery of this fault, the insight gained while performing these tests was taken and used towards creating the final version of the model.

All testing was conducted with the Adam optimizer and the cross entropy loss function. There was a choice between Adam and SHD (stochastic gradient descent) for the optimizer in this project, however, Adam was chosen for the project because it is on average a more optimal optimizer and it often converges faster, which was important with the constraints of the project [24].

It is also worth noting that the models used in tests were always pre-trained on the ImageNet dataset prior to being trained on the herbarium dataset. It was determined that using pre-trained models is best because it saves valuable time during training, which was limited, and in turn it saves the limited GPU resources.

### 3.4.3 Test results

The biggest takeaway from this stage were the hyperparameter values chosen to be used in the final model. This is not the best way to do so, because different hyperparameters will act differently with other models, however, the time did not allow to test more combinations for each model.

The hyperparameters tested at this stage were: batch size, learning rate and the number of epochs.

The model where these hyperparameters were decided on was ResNet-152. Once the best performing hyperparameters were found, the same were used on ResNeSt-101 and DenseNet-169.

The batch size values tested were: 32, 64, 128 and 254
The learning rates: 1e-2, 1e-3, and 1e-4
Epochs value: No more than 10 and usually 4 or 5

The optimal batch size was found to be 32. Every other batch size above 32 was unreliable. A batch size of 128 scored fairly close to 32, but every other time a model was to be trained with a batch size above 32, Kaggle threw a "CUDA out of memory" error which would only be

resolved by lowering the batch size. Based on other users' experiences, it is a common error that results in being unable to train the model unless the batch size is lowered.

Afterwards, it was also found that some research indicates how smaller batch sizes, even though they take longer to train, they prove to be more accurate in general, with one research specifying the batch size of 32 as the most optimal out of the tested values [25].
As for the learning rate, the value of 1e-3 was found optimal, however, later both values 1e-3 and 1e-4 will be used in tests when developing the final model as they were both found to be useful in certain situations.

Epochs weren't strictly specified prior to testing, but rather the performance was measured after every epoch. Although, due to time and resource constraints, the epochs were never trained above 10, and mostly trained until 4 or 5. Epochs were stopped based on the difference between training and validation F1 scores. The choice was made for the training and validation F1 to not have more than a 10% difference. This is through reading several sources about overfitting [26]. This was later further reinforced when it was discovered that such a difference makes the model generalize well on the new unseen data.

Manually going through all the hyperparameter values was not the best solution to fine-tuning, as there were more optimal ways of finding those values. This will be discussed in the following sections.

#### 3.4.3.1 DenseNet-169 Results

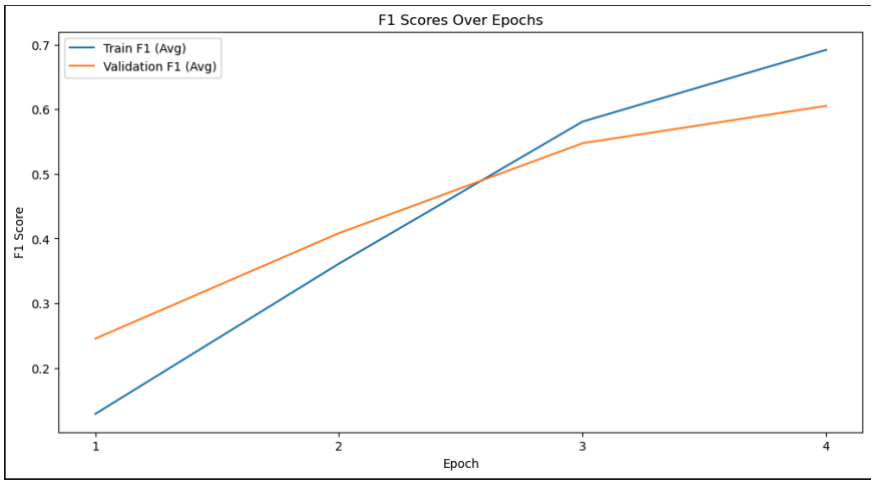


*Figure 10 - DenseNet-169 Training and validation F1*

The training was stopped after 4 epochs, when the average training F1 was 69.18% and the average validation F1 was 60.51.

The accuracy on the test set was 74.54% and the F1 score was **62.**

#### 3.4.3.2 ResNet-152 Results

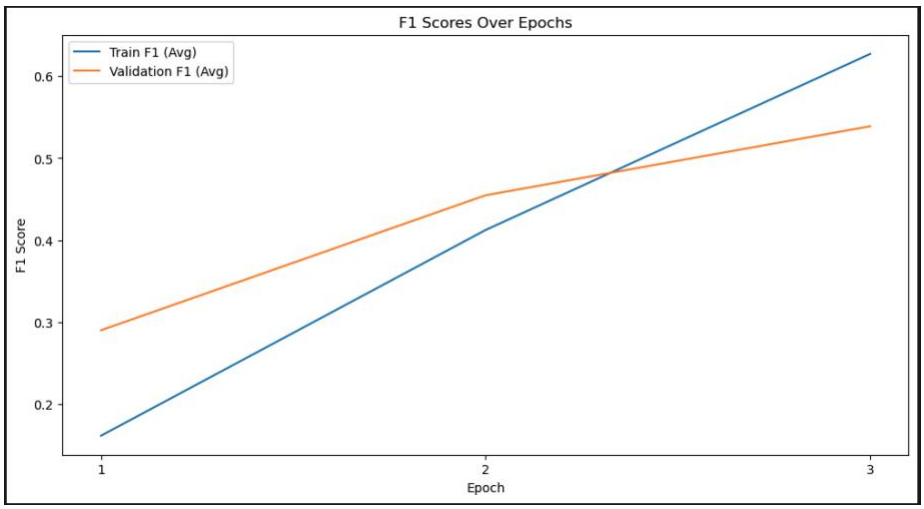


*Figure 11 - ResNet-152 Training and validation F1*

The training was stopped after 3 epochs when the average training F1 was 62.79% and the average validation F1 was 54.93%.

The accuracy on the test set was 74.14% and the F1 score was 54%

#### 3.4.3.3 ResNeSt-101 Results

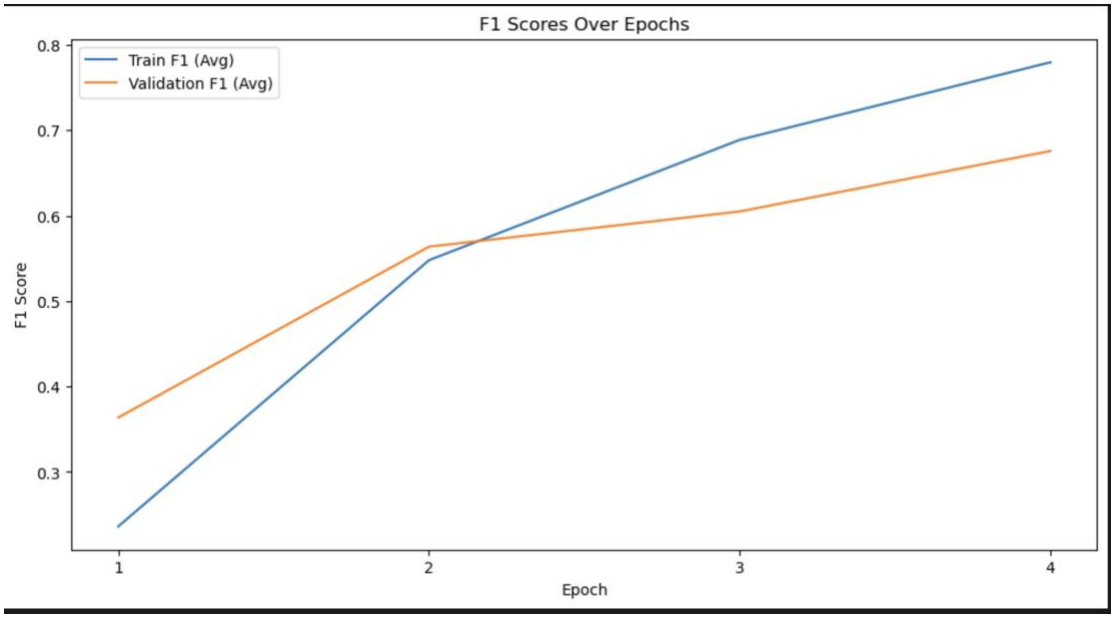


*Figure 12 - ResNeSt-101 Training and validation F1*

The training was stopped after 3 epochs when the average training F1 was 62.79% and the average validation F1 was 54.93%.

The accuracy on the test set was 81.4% and the F1 score was 70

## 3.5 Final model development

After the specified hyperparameters were tested and ResNeSt was selected as the final model, different strategies and tests were conducted to try to get the highest possible score, and even though the "best" hyperparameters were set in the previous stage, some were still experimented with.

ResNeSt has several sizes. ResNeSt-50, ResNeSt-101, ResNeSt-200 and ResNeSt-269. They were all tested, and ResNeSt-101 came as the one with the highest accuracy and F1 score for set hyperparameters. These are the results:

| | Test accuracy | Average F1 score |
|---|---|---|
| ResNeSt-269 (Val F1 = 62.15) | 75.40% | 62.6 |
| ResNeSt-200 (Val F1 = 59.9) | 75.02% | 62 |
| ResNeSt-101 (Val F1 = 69.50) | 81.89% | 71.2 |
| ResNeSt-50 (Val F1 = 62.85) | 79.48% | 66 |

### 3.5.1 Additional features

When the best-performing model was identified as ResNeSt-101, there were more attempts at making it more accurate. Rather than tuning the hyperparameters, other methods were used such as assigning weights to classes based on the number of samples, applying augmenting techniques to images, using cross-validation etc. Some of them will be discussed here.

#### 3.5.1.1 Weighted classes

Assigning weights to classes is a technique used in machine learning to handle imbalanced datasets, where the number of samples in different classes is not evenly distributed. It is important to balance the contribution of each class to the overall loss function to prevent the model from being biased towards the majority class [27].

This was done by first determining the number of samples of each class in the dataset. Afterwards, the classes with fewer samples were assigned heavier weights and the classes with more samples had lower weights.

This had such an effect on the dataset that the models F1 score in the first three epochs remained under 1%, as seen in Figure 12.

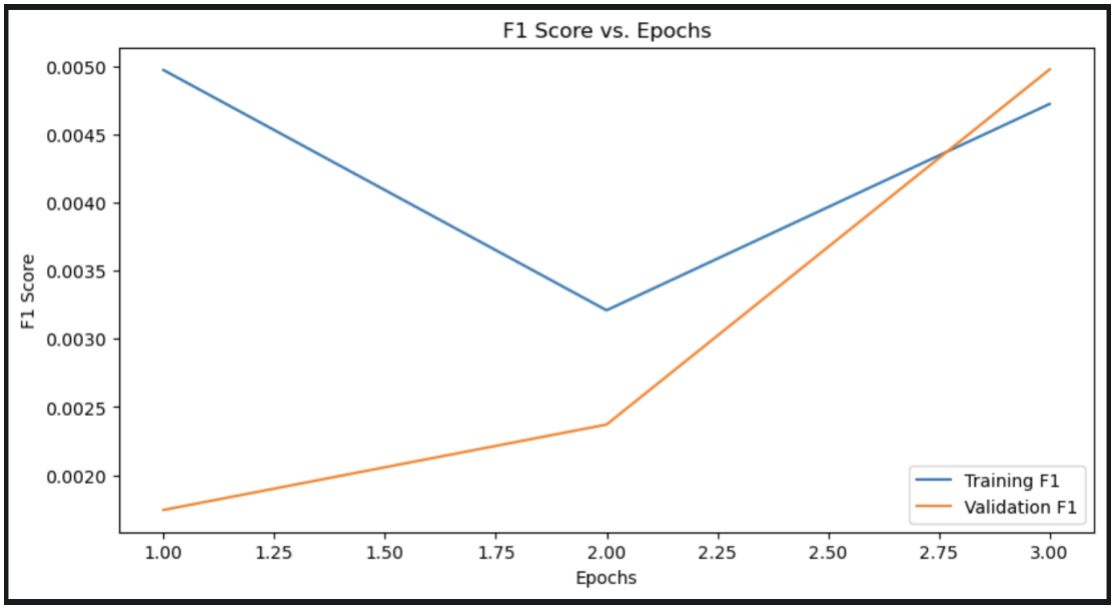


*Figure 13 - F1 scores with weighted classes implementation*

Due to the slim chances of this method working, a decision was made to stop it after 3 epochs to use the computational resources on other tests.

#### 3.5.1.2 Image augmentations

Image augmentation is a method used in deep learning to artificially expand and diversify the training dataset. This technique involves applying a range of transformations to the original images, including rotation, scaling, flipping, and noise addition, to name a few. These transformations create new, modified versions of the original images while maintaining their core meaning [28].

In this case, image augmentation was used to try to improve the model's generalization capabilities on new and unseen data. The following augmentation methods were used:

- RandomHorizontalFlip – randomly flips the input image
- RandomRotation(20) – randomly rotates the image by 20 degrees to the left or right
- ColorJitter – randomly changes the brightness, contrast, saturation and hue of the image

This has in turn lowered the overall F1 to **63**. It is not exactly understood why this happened, as the augmentations have been applied to all three sets, the training, validation and test set. However, might be due to several reasons, and a common one is wrong hyperparameters. Even though the hyperparameters work well enough for the model, it does not mean they are ideal. This could be further experimented with and would likely result in the augmentations providing better generalization and F1 score.

#### 3.5.1.3 Cross-validation (K-fold validation)

Cross-validation is used to evaluate the performance and generalizability of a model. It involves partitioning the available data into multiple subsets, training the model on some of these subsets (training data), and then validating the model's performance on the remaining subsets (validation data). This process is repeated multiple times, and the model's performance is averaged across all iterations to provide a more reliable estimate of its

generalization capabilities [29]. This greatly helps reduce the chance of overfitting as it trains and validates the model across the whole dataset.

Stratified K-fold is a form of K-fold validation

This approach was attempted. It was done by splitting the data into 5 folds with a goal of iterating over all 5 folds X times, where X is the epoch. However, with the limitations of Kaggle, which forces you to close the notebook after 10 hours of runtime as to deter users from really long training sessions, this was not possible, as it was calculated that the training would take at least 15 hours.

# 4 Results and Conclusions

## 4.1 Final model

After some additional hyperparameter tuning and testing, the final model was concieved. These are the results:

Training F1: 79.98
Validation F1: 69.82

Testing Accuracy: 81.14%
Average F1 score: 0.70

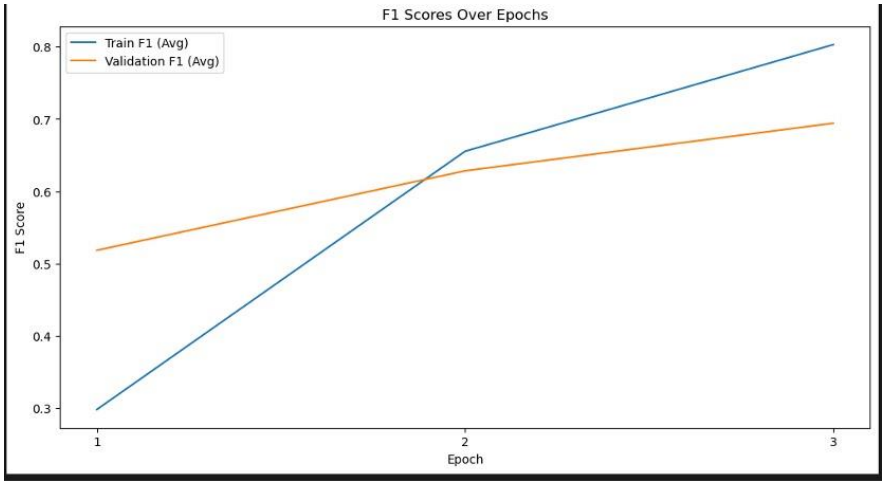


*Figure 14 - ResNeSt-101 final model performance*

To achieve this result, these were the hyperparameters used:

- A batch size of 32
- 3 epochs
- Input image size of 224
- Learning rate 1e-4
- Number of layers: 101

These results represent the model's performance on the Poaceae dataset, consisting of 158 classes. Some further tests were also done firstly adding another family Fabaceae, and also using "category" labels instead of the "genus" labels.

### 4.1.1 Results with two families

These are the results of testing when using Poaceae and Fabaceae families. Resulting in 275 classes.

Training F1: 82.06
Validation F1: 72.08

Testing Accuracy: 82.12%
Average F1 score: 0.71

Using ResNeSt-101 with two families gave a very similar result to using just one. This is not a surprise because of the structure of the labels. Two different families were used with genera that are quite different compared to using category labels.

### 4.1.2 Results using the category labels

Training F1: 64.80
Validation F1: 55.91

Testing Accuracy: 55.84%
Average F1 score: 0.52

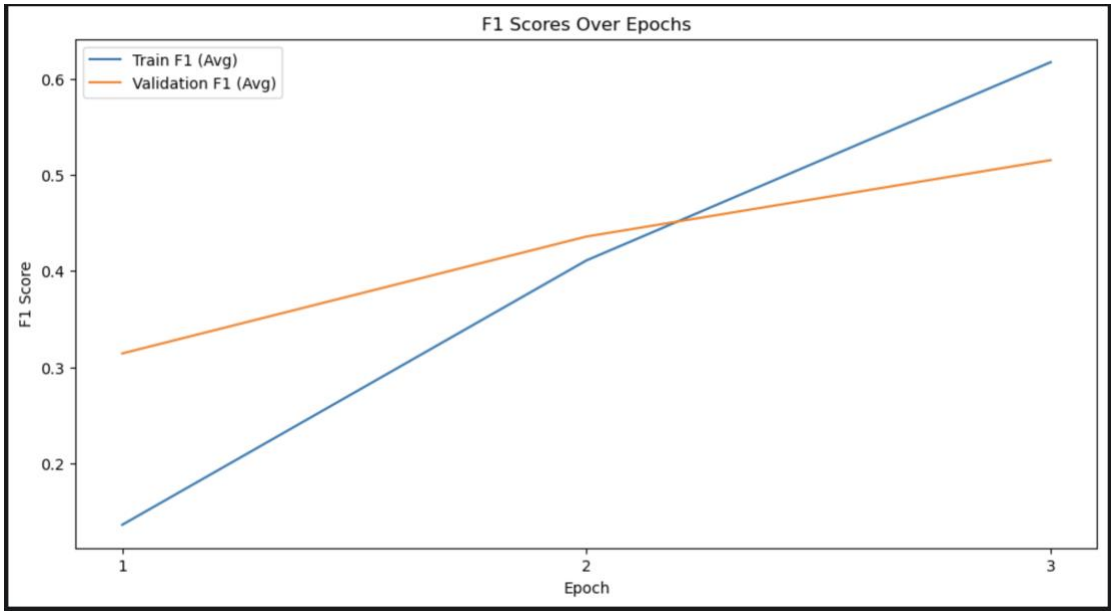


*Figure 15 - Results of ResNeSt-101 using category labels*

The reason for a much lower testing f1 score with categories used as labels compared to genus lies in the fact that images that fall under the same genus but different categories have much more similar features than images from different genera. Figure 16 shows just how similar plants from the same genus with different categories can be.

*Figure 16 - Striking similarity between different species of the same genus [2]*

## 4.2 Possible improvements

During model development and testing, some areas were found that could potentially achieve better results and become more efficient in terms of both training time and performance on unseen data.

### 4.2.1 Hyperparameter optimization

Automatic tuning of hyperparameters using search methods, such as grid search, random search, or more advanced techniques like Bayesian optimization. These methods systematically explore the hyperparameter space and help identify the best combination of parameters for the model, potentially leading to improved performance and reduced training time [30].

- Grid Search: In this method, the model's hyperparameters are adjusted over a predetermined range of values specified by the user. The grid search evaluates every possible combination of hyperparameter values within the defined grid to determine the optimal set for the model.
- Random Search: In this approach, the user defines a probability distribution for each hyperparameter, from which a designated number of samples are drawn. The model's performance is then assessed for each of these sampled combinations of hyperparameter values.
- Bayesian Optimization: This optimization technique takes into account the outcomes of previous evaluations when selecting the next set of hyperparameter values to be tested.

### 4.2.2 Learning rate scheduling

A learning rate scheduler is a part of the training process and is used to adjust the learning rate during the training of a model. The learning rate scheduler helps in optimizing the model's training by dynamically adapting the learning rate based on the number of epochs, the current performance, or other criteria. This can lead to more efficient training and, potentially, better model performance.

Implementing a learning scheduler has several benefits. It can reach the optimal solution quicker meaning less training time and more resources for further testing, a well-tuned scheduler can lead to better generalization performance by avoiding overshooting or getting stuck in local minima, it can help overcome plateaus or saddle points by temporarily increasing the learning rate, and this in turn can reduce the risk of overfitting [31].

### 4.2.3 Hierarchical labels

Hierarchical labels are a labeling system where classes or categories are organized in a tree-like structure, with multiple levels of granularity. In this system, categories are arranged in a hierarchy from more general to more specific, and a given class or label can have parent and child classes. Hierarchical labels are used to capture the relationships between different classes, making it easier to understand and navigate complex classification problems [32].

# 5 Critical Evaluation

Looking back at the whole project and the experience, I can confidently say that this project was the right choice. I have always been extremely interested in machine learning and computer vision and this project had both. Knowing how much I know now, there are certainly some things I would have done differently if starting over. I will go over the project phases and evaluate each of them.

## 5.1 Development phase

For the first two sprints, I definitely should have spent less time dealing with EDA. Looking back, a far better approach would have been to use third-party code and get the dataset ready to start model development and later on, testing. Spending too much time on EDA caused me to later lose valuable time that could have been spent experimenting with methods of improving the model's accuracy.

One thing I am glad for is the time I took during the model development itself, after EDA. I could have just copied and pasted training and evaluation loops and run them with my code, modifying them on the way, but that would not contribute to my understanding of the process. I took my time to fully understand the architectures of models and the training and validation process that my understanding is now on a very high level.

I am also happy with the Scrumban approach that I followed during the development. It helped me greatly in keeping track of relevant knowledge and resources. The bi-weekly reviews also put me back on the right track more than once.

## 5.2 Testing phase

During the time I was doing testing, rather than doing lots of tests for all the models, I should have instead picked one model based on research, and stuck with it until I have the model completely developed. Simultaneously trying to train and test several models took away from the limited time and resources I had. Focusing on one model and developing all the methods that I wasn't able to, such as hyperparameter optimization, learning rate scheduling, hierarchical labels and even the lengthy cross-validation would have been more beneficial in understanding deep learning and neural networks.

I am still satisfied that I managed to compare several models and run lots of tests, but if I had followed the steps like above, more could have been discovered.

## 5.3 Time management

The biggest setbacks came from work management rather than time management. I found myself very invested in the project when I was doing it, but that time and energy would have been better spent focusing on different tasks and a different development flow, as discussed above. I do believe I didn't use the Easter break as effectively as I wanted to and that also set me a little behind.

## 5.4 Review of the objective

At the start of the project, I was too ambitious. I set that aim to develop a model to be able to perform inference on the whole dataset with relatively good results. I have found along the way that this was not only over-ambitious but also technically infeasible with the programs and the hardware I had at my disposal. The Kaggle notebook shuts down automatically after the model has been running for 10 hours. And a single epoch of training for the whole dataset would take at least 8 or more hours. However, I do think that the revised objective of working with the Poaceae dataset was reasonably well met, although it could have been met even better if a different approach was taken in certain areas, as described above.

## 5.5 Overall experience

I can say for certain that if I could choose a project for this module again, I would choose the same one. I have always wanted to get into machine learning but it always seemed too daunting and difficult to even start. This project allowed me to dive into the ML world head first with proper guidance but also with great autonomy.

## 5.6 Future plans

I will continue developing the model and will for the time being stick with ResNeSt101. I will try to implement all the functionalities that I didn't have time to fully develop and keep track of the model's performance with them. The plan is to also switch from genera labels to category labels and make full use of hierarchical labeling as well as expanding the dataset to more families-

## A. Third-Party Code and Libraries

- **Pandas**: A data manipulation and analysis library for Python, providing data structures like DataFrames for efficient data handling.

- **NumPy**: A powerful numerical computing library for Python, enabling efficient operations on large arrays and matrices.

- **PyTorch**: An open-source machine learning framework that facilitates the development and training of neural networks.

- **Matplotlib**: A 2D plotting library for Python that produces publication-quality figures in various formats.

- **Seaborn**: A statistical data visualization library based on Matplotlib, providing an aesthetically pleasing and informative interface for statistical graphics.

- **JSON**: A lightweight data interchange format used for easy data exchange between systems.

- **Cv2** (OpenCV): Open Source Computer Vision Library, providing tools and functions for computer vision tasks like image and video processing.

- **Scikit-learn (sklearn):** A machine learning library for Python that provides simple and efficient tools for data analysis and modeling, including classification, regression, clustering, and more.

# 7 References

This final section should list all relevant resources that you have consulted in researching your project.

[1] Appendices Stefanaki, A., Porck, H., Grimaldi, I.M. and Thurn, N. (2019). Breaking the silence of the 500-year-old smiling garden of everlasting flowers: The En Tibi book hebarium. PLOS ONE, 14(6), p.e0217779. doi:https://doi.org/10.1371/journal.pone.0217779.

[2] Bebber, D.P., Carine, M.A., Wood, J.R.I., Wortley, A.H., Harris, D.J., Prance, G.T., Davidse, G., Paige, J., Pennington, T.D., Robson, N.K.B. and Scotland, R.W. (2010). Herbaria are a major frontier for species discovery. Proceedings of the National Academy of Sciences, 107(51), pp.22169–22171. doi:https://doi.org/10.1073/pnas.1011841108.

[3] Wu, D., Wu, D., Feng, H., Duan, L., Dai, G., Liu, X., Wang, K., Yang, P., Chen, G., Gay, A.P., Doonan, J.H., Niu, Z., Xiong, L. and Yang, W. (2021). A deep learning-integrated micro-CT image analysis pipeline for quantifying rice lodging resistance-related traits. Plant Communications, [online] 2(2), p.100165. doi:https://doi.org/10.1016/j.xplc.2021.100165.

[4] Brownlee, J. (2020). Softmax Activation Function with Python. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/softmax-activation-function-with-python/#:~:text=The%20softmax%20function%20is%20used%20as%20the%20activation%20function%20in [Accessed 4 May 2023].

[5] Goodfellow, I., Bengio, Y. and Courville, A. (2016). Deep Learning. [online] Deeplearningbook.org. Available at: https://www.deeplearningbook.org/.[Accessed 4 May 2023]

[6] www.ibm.com. (n.d.). What is Exploratory Data Analysis? | IBM. [online] Available at: https://www.ibm.com/topics/exploratory-data-analysis#:~:text=Exploratory%20data%20analysis%20(EDA)%20is [Accessed May 1AD].

[7] Dsouza, J. (2020). What is a GPU and do you need one in Deep Learning? [online] Medium. Available at: https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d [Accessed 4 May 2023].

[8] Huilgol, P. (2019). Accuracy vs. F1-Score. [online] Medium. Available at: https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2 [Accessed 4 May 2023].

[9] Terra, J. (2020). Keras vs Tensorflow vs Pytorch: Popular Deep Learning Frameworks. [online] Simplilearn.com. Available at: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article [Accessed 4 May 2023].

[10] De Lutio, R., Park, J.Y., Watson, K.A. and D'Aronco, S. (2022). The Herbarium 2021 Half–Earth Challenge Dataset and Machine Learning Competition [Accessed 4 May

2023]. [online] frontiersin.org. Available at: https://www.frontiersin.org/articles/10.3389/fpls.2021.787127/full#B2.

[11] Juviler, J. (2022). What Is GitHub? (And What Is It Used For?). [online] blog.hubspot.com. Available at: https://blog.hubspot.com/website/what-is-github-used-for [Accessed 4 May 2023].

[12] pytorch.org. (n.d.). Training a Classifier — PyTorch Tutorials 1.5.0 documentation. [online] Available at: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html [Accessed 4 May 2023].

[13] docs.nvidia.com. (n.d.). Train With Mixed Precision. [online] Available at: https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html [Accessed 4 May 2023].

[14] Costa-Luis, C. da (n.d.). tqdm documentation. [online] tqdm.github.io. Available at: https://tqdm.github.io/ [Accessed 4 May 2023].

[15] Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Lin, H., Zhang, Z., Sun, Y., He, T., Mueller, J., Manmatha, R., Li, M. and Smola, A. (2020). ResNeSt: Split-Attention Networks. [online] Available at: https://arxiv.org/pdf/2004.08955.pdf [Accessed 4 May 2023].

[16] He, K., Zhang, X., Ren, S. and Sun, J. (2015). Deep Residual Learning for Image Recognition. [online] arXiv.org. Available at: https://arxiv.org/abs/1512.03385v1 [Accessed 4 May 2023].

[17] IBM (n.d.). What are Convolutional Neural Networks? | IBM. [online] www.ibm.com. Available at: https://www.ibm.com/topics/convolutional-neural-networks [Accessed 4 May 2023].

[18] O'shea, K. and Nash, R. (2015). An Introduction to Convolutional Neural Networks. [online] Available at: https://arxiv.org/pdf/1511.08458.pdf [Accessed 4 May 2023].

[19] Jason Brownlee (2019). A Gentle Introduction to the Rectified Linear Unit (ReLU) for Deep Learning Neural Networks. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ [Accessed 4 May 2023].

[20] uk.mathworks.com. (n.d.). What is Half Precision? - MATLAB & Simulink - MathWorks United Kingdom. [online] Available at: https://uk.mathworks.com/help/coder/ug/what-is-half-precision.html [Accessed 4 May 2023].

[21] Chi-Feng Wang (2019). The Vanishing Gradient Problem. [online] Medium. Available at: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 [Accessed 4 May 2023].

[22] Baeldung. (2023). What is the Purpose of a Feature Map in a Convolutional Neural Network. [online] Available at: https://www.baeldung.com/cs/cnn-feature-map [Accessed 4 May 2023].

[23] Jordan, J. (2017). Evaluating a machine learning model. [online] Jeremy Jordan. Available at: https://www.jeremyjordan.me/evaluating-a-machine-learning-model/ [Accessed 4 May 2023].

[24] Jason Brownlee (2017). Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ [Accessed 4 May 2023].

[25] Thakur, A. (2020). What's the Optimal Batch Size to Train a Neural Network? [online] W&B. Available at: https://wandb.ai/ayush-thakur/dl-question-bank/reports/What-s-the-Optimal-Batch-Size-to-Train-a-Neural-Network---VmlldzoyMDkyNDU [Accessed 4 May 2023].

[26] Landup, D. (2021). Machine Learning: Overfitting Is Your Friend, Not Your Foe. [online] Stack Abuse. Available at: https://stackabuse.com/machine-learning-overfitting-is-your-friend-not-your-foe/ [Accessed 4 May 2023].

[27] Singh, K. (2020). How to Improve Class Imbalance using Class Weights in Machine Learning. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/ [Accessed 4 May 2023].

[28] Sharma, P. (2019). Image Augmentation | Pytorch Image Augmentation. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2019/12/image-augmentation-deep-learning-pytorch/ [Accessed 4 May 2023].

[29] Brownlee, J. (2018). A Gentle Introduction to k-fold Cross-Validation. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/k-fold-cross-validation/ [Accessed 4 May 2023].

[30] Sukumar, R. (2020). Introduction to Automatic Hyperparameter Optimization with Hyperopt. [online] Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/introduction-to-automatic-hyperparameter-optimization-with-hyperopt-e0b9c84d1059 [Accessed 4 May 2023].

[31] Suki Lau (2017). Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning. [online] Medium. Available at: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1 [Accessed 4 May 2023].

[32] Romero, M., Finke, J. and Rocha, C. (2022). A top-down supervised learning approach to hierarchical multi-label classification in networks. Applied Network Science, 7(1). doi:https://doi.org/10.1007/s41109-022-00445-3.