Methodology & Design

Pre-Design Experimentation

Prior to officially starting design work on the project’s code, various experiments were conducted to gain understanding and familiarity with tools and libraries essential to the nature of the project. The code was formed using an amalgamation of snippets gathered from various sources, including tutorials, videos and articles. These pre-design experiments utilised a novel dataset of leaves gathered from a local park. This dataset consisted of 5 species, of which 10 leaves were collected. These 50 leaves were then photographed in 5 different environments to expand the dataset to 250 images. Additionally, in order to comprehend the possibility of expanding datasets further, data augmentation experiments were conducted, with 5 augmentations applied to each image, including rotation, vertical and horizontal reflection, translation, and colour adjustment. By applying these augmentations, the dataset was expanded to 1250 images.

There was no pressure on these initial experiments to produce meaningful results, which is fortunate as the results were not great. This was partly due to the poor quality of the input dataset, a result of insufficient data pre-processing. The original images were too high-resolution, the photography environments too busy, and the data augmentation too intense. However, various important lessons were learned regarding the significance of preparing the input dataset to give the model every advantage during its learning process.

Another reason for the lack of success of these pre-design experiments was the code responsible for defining and training the model. Since the code used was an amalgamation gathered from various sources, it is certain that the different elements were not fine-tuned to work smoothly with one another. This revealed another important lesson, that the code itself must be optimised in order to produce a successful, robust model.

Following this pre-design experimentation stage, the lessons learned provide sufficient direction to begin the design process of the deep learning program.

Management Methodology

Prior to beginning the design stage, consideration went into which management methodology would be most appropriate for a project of this nature. One option, *Kanban*, stood out as the obvious choice for many reasons. Firstly, the *Kanban* board itself offers great benefit as it clearly visualises the project’s tasks and their place in the completion process, ensuring organisation and structure are maintained during the project’s development. Secondly, *Kanban* is a flexible framework, which suits the iterative, evolutionary development process of a project of this nature, whereby an initially unpredicted implementation may be required. Such implementations are difficult in other more rigid management methodologies; however, *Kanban* supports them with its fluidity, allowing them to be performed without disturbing the flow of the project. Finally, Kanban is.

Initial Design

The initial version of the program will be designed using the knowledge gained from a variety of sources during the learning and research phase of the project, with the intention of starting with simplicity and expanding as is needed. Based upon this research [1], the assembly of a successful model requires four key components: importation of the required libraries and tools, data preparation/preprocessing, model development, and performance analysis.

The program’s operation will start by importing the essential libraries: *TensorFlow* (including the *Keras* API), *MatPlotLib*, and *CSV*. Next, some important data preprocessing will be performed, including the establishment of constants such as the dataset directory and image size, the splitting of the dataset into separate training and validation sets, and their normalisation. Next, model’s architecture will defined, compiled, and trained, and finally the training and validation results will be visualised and stored for further analysis.

This simple and efficient blueprint will form the foundation upon which future implementations can be developed. Containing each of the key stages listed above, these four components will grow and expand as necessary, with the importation of new libraries, enhancement of the preprocessing, expansion of the model’s architecture, and inclusion of more analytics. Many features will be added, and many tests will be conducted, but this initial program will be a solid foundation on which to develop the rest of the project.

Tools & Libraries

As stated above, the first iteration of this project’s code will require three essential libraries to perform its objective.

The most essential library is *TensorFlow*, responsible for providing the utilities and framework for creating, assembling, training and testing the deep learning classification model. One such utility is image\_dataset\_from\_directory, which loads a dataset from a specified directory and converts its images into a *TensorFlow* dataset.

One of two vital packages supplied by *TensorFlow* (via *Keras*) is models, a collection of elements made for developing the model’s architecture, including models.Sequential, used to define the model as having sequentially stacked layers; model.compile for configuring the optimiser and loss function; and model.fit which controls the training process.

The second crucial package used within the initial program is layers, containing components such as layers.Conv2D, which specifies the inclusion of convolutional layers used for feature extraction; layers.MaxPooling2D, used to reduce the spatial dimensions of the feature maps while preserving the most significant values, and layers.Rescaling, which normalises pixel values within the range of 0 to 1.

To facilitate the storage and visualisation of the results within this initial version, *CSV* and *MatPlotLib* will be required. The role of the former will be to record the training accuracy, training loss, validation accuracy and validation loss for each epoch, while the role of the latter will be to plot visualisations of this resultant data to enhance the analytical process.

|  |
| --- |
|  |

Hyperparameters

The development of the model and the improvement and its performance will involve the manipulation of many aspects, such as the layers, nodes and functions of the architecture itself, or the size, quality and balance of the input dataset. However, the primary method of experimentation for the initial implementation phase will focus on manipulating hyperparameters, including the learning rate, batch size, dropout rate, and the number of epochs.

The learning rate controls how much the model updates weights in each step. A small learning rate causes slow but precise updates, while a large learning rate trains the model quicker, but can overshoot the optimal values. The batch size is the number of samples processed by the model before the weights are updated. A larger batch size speeds up the training process, but can be more computationally expensive, while a smaller batch size updates the weights more regularly, and can lead to better generalisation, but will increase the execution time. Dropout rate refers to the percentage of neurons which are disabled during training. By including a dropout rate, the model can have better generalisation and therefore higher accuracy, however if the value is too high the model may struggle to learn its data. The number of epochs refers to how many times the entire dataset is passed through the model during training. In general, increasing the number of epochs allows the model to learn the data better, but increasing this value too high can lead to overfitting.

Following the initial implementation period, other methods of manipulation will be integrated, however by starting the experimentations with these aforementioned hyperparameters, substantial insight should be gained, translating into improved accuracy and reduced loss as the project continues.

Evaluation Metrics

The initial version of this program will start with a minimal set of evaluation metrics and expand as the project complexifies. For this reason, the first iteration will focus on accuracy and loss as the primary performance indicators for the training and validation sets during each epoch.

* Accuracy represents the percentage of correctly classified examples out of the total predictions. It provides a simple yet useful measure of the model’s overall performance.
* Loss quantifies the difference between the predicted probabilities and the actual labels. Unlike accuracy, which gives a general correctness percentage, loss provides a more nuanced measure of how confident and accurate the model’s predictions are. A lower loss value indicates that the model’s predicted probabilities are closer to the true labels.

While many other evaluation metrics can be introduced in future iterations, accuracy and loss offer sufficient insight at this early stage to identify patterns and suggest initial improvements. However, research has been conducted into additional metrics that may be beneficial in later stages of development.

One such method is the confusion matrix, which is a table that compares the model’s predicted labels against the true labels for each class, providing a clear visual representation of classification performance.

In an ideal scenario, most values in the confusion matrix should align along the diagonal (from the top left to the bottom right). These represent correct classifications—whether true positives (TP) for a given class or true negatives (TN) for other classes. However, real-world models often make mistakes, and off-diagonal values indicate misclassifications:

* False positives (FP) – When the model incorrectly predicts a sample as belonging to a class when it does not.
* False negatives (FN) – When the model incorrectly predicts a sample as not belonging to a class when it actually does.

By examining which classes the model struggles with, confusion matrices help identify specific areas for improvement. Misclassifications can highlight model weaknesses, enabling focused debugging and data adjustments to enhance performance.

A graph of a number

AI-generated content may be incorrect.

*Figure 6 Confusion Matrix Example*

Another way to evaluate the performance of the model would be the generation of a classification report. These provide detailed into the performance of the model regarding each class within the dataset. Typical metrics found within classification reports (in addition to the already-covered accuracy and loss) include precision, recall, F1-score, and support.

Precision measures, for each class, how many of the predictions made for that class were correct. This is done by dividing the number of true positives (correct predictions) by the total number of predictions for that specific class. This concept is easier to understand by examining the equation below. A high precision value indicates that the model was successful at labelling the given class, and that most of its classifications were true positives, with few false positive predictions.

Recall is the measurement of how many positive samples were classified correctly. In other words, recall examines, for each class, the number of correctly classified positive labels versus those which were incorrectly classified as negatives. Again, this concept is easier to understand through the equation below. Recall is useful for identifying how many positive samples were incorrectly labelled as negative.

F1-score is a value which balances both precision and recall, and can otherwise be defined as their harmonic mean. This generally makes it more versatile and useful metric, as it considers both false positives and false negatives. F1-scores fall on a scale between 0 and 1, and the higher the score, the better the model is at avoiding false positives and false negatives. The F1-score equation can be seen below:

While typically found within classification reports, the support value is not a performance metric, and therefore alone does not provide insightful analysis like the other three aforementioned values. Instead, support simply indicates how many samples are contained within each class, which is still a useful piece of information, as in combination with these other metrics, it can confirm whether or not the dataset is balanced.

Dataset & Preprocessing

The dataset to be used within this initial program version is a small, modified subset of 15 lab-taken image classes from the *LeafSnap30* dataset, which itself is a subset of the much larger, well-known *LeafSnap* dataset [2]. Remaining consistent with the initial design’s ethos of simplicity, the dataset component will be kept simple to facilitate a smooth initial development period, with intent to implement larger, more complex datasets as the project progresses.

Development Plans

In addition to the aforementioned initial methods of improving the model’s performance, plans are in place to further develop the model and its classification capabilities.

One such plan relates to the input dataset. For example, several relevant datasets have been identified which could not only increase the volume of input data, but also introduce more complexity, therefore increasing the model’s ability to generalise, and hopefully leading to a more robust and accurate model. Another idea involves utilising data augmentation as a means of balancing unequal classes, both internally during program execution, and externally by creating a dataset with even classes.

In relation to the increasing the challenge provided by the input data, another element of the development plan involves increasing the size and complexity of the model’s architecture. Initially this will be done by adding more layers and nodes, and by manipulating their functions. However, once this experimentation has been exhausted, pre-trained models will be explored and implemented for their superior architecture and performance. During the research stage, many viable options were encountered including *MobileNet*, *EfficientNet*, *ResNet* and *VGG*. These will be experimented with and tested, with the most viable option likely being utilised within the latter versions of the model development program.

By implementing these plans after completing the initial experimentational period, the options for expanding the model’s classificational capabilities will be vast, and will hopefully lead to the successful development of a robust and accurate classification system.