University of Zimbabwe

Deep Learning for Improved Plant Classification

A thesis submitted to the cluster Computer Science Education in Partial fulfillment of the requirements for the award of a

Master of Science Degree
(Computer Science)

By

Innocent T. Kwangware

Supervisor: Professor Serestina Viriri

July 2018

DEDICATION

This work is dedicated to God, All Mighty
And
To my wife Priviledge and our children, Emmanuel and Providence.

ACKNOWLEDGEMENTS

I am very grateful to my supervisor Professor Serestina Viriri for the help and the support I received from him during my research. I would like to extend my gratitude to my beloved family and all friends for the provision of support and for their encouragements.

ABSTRACT

Plants are crucial in the ecosystem: they enable life by providing oxygen, they have medicinal properties, among many other uses, and hence their classification is vital. Automating plant identification has remained a challenging task. Machine learning does not automate feature engineering, making its application to the problem an arduous task. Deep learning automates featuring engineering, but requires large datasets, presenting another challenge. This research proposes an improved and fast plant classification model for plants using deep learning on small datasets. The objectives of this research are to explore techniques that are state-of-the-art in the classification of plants using leaves, to improve the classification accuracy scores, and to model a framework for the classification. A convolutional neural network was designed and applied on a small leaf dataset with fine-tuning. An accuracy score of 94.99% was achieved. High overfitting was noted since the dataset was small. Data augmentation was applied to the dataset with the images augmented before being input into the model, differing from the many cases where augmentation is applied on-the-fly. This increased the success rate to 99.99% with reduced overfitting. This showed that augmentation applied in this way improves the performance of the model. Transfer learning with the same augmentation method was applied, resulting in a 100% test accuracy and stable results with low overfitting. The proposed methodology provided good results on the Flavia leaf data set used in the experiments. Finally, a deep learning framework for improved plant classification using leaves is outlined.

KEY WORDS

Deep Learning	Machine Learning	Convolutional Neural Networks
Transfer Learning	Augmentation	Plant Classification
Feature Engineering	Feature Extraction	

TABLE OF CONTENTS

TABLES

FIGURES

ABBREVIATIONS

- **CNN** Convolutional Neural Networks.
- **PNN** Probabilistic Neural Networks
- **HSV** Hue-Saturation-Value
- **HIS** Hue-Saturation-Intensity
- **RBPNN** Radial Based Probabilistic Neural Network
- **GLCM** Gray-Level Co-Occurrence
- **PCA** Principal Component Analysis
- **SCED** Simple Canny Edge Detection
- **SIFT** Scale Invariant Feature Transform
- **CNN** Convolutional Neural Network
- **ReLU** Rectified Linear Unit
- **ANN** Artificial Neural Network
- **FC** Fully Connected
- **CM** Color Moments

CHAPTER 1: GENERAL INTRODUCTION

1.1. Introduction

Plants serve crucial roles in the ecosystem. All life is anchored on plants, and without them there would be no sustenance of life on earth (Chaki & Parekh, 2011). It is a true statement that many types of plants are at the danger of disappearance moving into the future (Chaki & Parekh, 2011). Plant recognition is important as it helps us to preserve the endangered plant species, however it remains a challenging task. We need to recognize plants to manufacture medicines and, also, plants provide alternative sources of energy, for example bio-fuel. Plant classification is also pivotal for environmental protection, agriculture, remote sensing, geographic information system, education and museum catalogues. Work has been done to develop systems that help in plant recognition through the use of leaf images (Kumar1 et al, n.d.). Computer vision and pattern recognition techniques have been applied by researchers in the recent past to try to automate plant recognition.

Classification of plants based on images of leaves supersedes other methods like cell and molecule biology, hence efforts in automating this classification has mainly focused on leaf recognition. It is noticeable that taking leaf samples and imaging them for classification are both convenient and comes at a lower cost (Ehsanirad, 2010). Singh et al (2010) noted that automating plant classification through computer vision remains a very challenging task due to lack of proper architectures and frameworks for such tasks. In 2012 convolutional neural networks or CNNs came to the lime light, as they were used to a great success, improving dramatically over the previous state-of-the-arts in the ImageNet computer vision competition (Krizhevsky, et al., 2012). From that time, there has been an upsurge in the number of researchers using convolutional neural networks in trying to classify plants.

1.2. Motivation

Plant identification requires in-depth knowledge; it is very complex that even professional botanists need a lot of time to be subject matter experts in this respect. Linnaeus identified

and categorized over 300000 plant species by applying the systematic plant classification he designed (Linnaeus, 1967). Although his work has been outstanding, it still demands a lot to a trained botanist to use Linnaeus classification model because of the number of rules and the fact that these rules require strong observation capabilities from the botanists. Here comes computer image processing techniques to help in the recognition process of plant species, by automating the analysis of plant parts such as leaves and stem, to improve the success rates on image recognition and also to economize on the processing time (Kumar et al, 2012). Hence, there is need for a proper model of acquiring images, pre-processing images, extracting image features, selecting features and finally leaf classification, during which the features extracted are combined into a single vector for classification. Therefore, an appropriate methodology with feature extraction and the classification algorithm(s) is required. Fundamental to this automation is the requirement that the features be generic and simple enough, so that they can be computed with relative easiness, and be applied to a variety leaf images. Applications of the automated process of image processing and recognition range from weeds identification, plant taxonomy, species discovery up to natural reserve park management to mention but a few. (Silva et al, 2013).

1.3. Problem Statement

Although many efforts employing complex leaf feature extraction techniques , computer vision and machine learning algorithms have been made, for example as in research papers by Beghin et al (2010), Kadir et al (2010) , Charters et al (2014) and Kalyoncu and Toygar (n.d.), automated plant identification is unarguably still an open and challenging problem. A major challenge is because plants in nature have very similar characteristics that is, shape and color representations, among others. Plant recognition using leaf images has been a challenging task and problem owing to the wide-ranging diversity of plant types coupled by the limitation of the leaf image two-dimension (2D) used to represent a plant leaf which is a three-dimension (3D) structure by nature (Wilkin et al, 2012). Hence, although different approaches have been applied in trying to automate the recognition of plant species and progress noted, this area remains an ongoing research topic as there is need for further improvements. This research focuses on the application of deep learning in plant classification using leaf recognition, in a

quest to improve and achieve higher accuracy rates in plant classification. This research also attempts to address the problem of using deep learning with small leaf datasets, as deep learning requires large datasets.

1.4. Dissertation Objectives

This research aims to propose a more accurate and fast plant classification model using leaf images by using Convolutional Neural Networks. This research has the following objectives:

1. To explore state-of-the-art or advanced methods for classification of plants based on leaf recognition.
2. To improve the classification of plants through recognition of their leaves using Convolutional Neural Networks.
3. To model a framework for accurate classification of plants based on leaf recognition

1.5. Dissertation Contributions

This dissertation explores the use of deep learning, specifically the Convolutional Neural Network, in plant classification using image recognition. The deep learning model has been fine tuned to produce higher accuracy scores and to minimise the cost or loss function.

Deep learning, in this case, has been applied on small datasets, achieving high results. Use of small datasets is a known problem with deep learning. This has been achieved by varying the methodology of using data augmentation, where this has been applied in a pre-cast form, differing with the many cases where it has been applied on-the-fly.

Data augmentation has also been tailored to be usable with transfer learning on the VGG16 pre-trained model for feature extraction, which resulted in higher and more stable classification accuracies. Normally, using this pre-trained model would require high computational power, but the methodology used allows its effective use with lower computing resources.

Modern techniques aimed at the classification of plants through leaf recognition by means of machine learning have been explored in this research. The conclusion drawn is that deep learning produces higher and more accurate results as compared to these contemporary methods.

Finally, an outline of a framework for accurate plant classification using leaf recognition is presented in this dissertation. The deep learning framework has been developed with Python, Open CV, and Anaconda with Jupyter Notebook and Keras Deep learning model implementing Tensorflow in the backend.

1.6. Outline of the Dissertation

This dissertation is then arranged in the following order: In Chapter 2 there is discussion of the work of other researchers, focusing mainly on the state-of-the-art approaches being applied for plant classification using leaf recognition. It starts by briefly describing the anatomy and taxonomy of plant leaves. Different methods in literature for automating plant classification using leaf recognition are then explored. Chapter 3 explores the background of deep learning itself in the context of application to this research, with focus on Convolutional Neural Networks. Chapter 4 explores how data augmentation and transfer learning have been modelled and how these have been applied in this research to achieve higher classification results. Chapter 5 discusses results obtained in the previous chapter. Chapter 6 presents the conclusion and some recommendations for future work.

CHAPTER 2: LITERATURE REVIEW

2.1. Introduction

This chapter first briefly describes the anatomy and taxonomy of leaves as these are the fundamentals used for plant classification in this dissertation. It then explores state-of-the-art techniques in literature for leaf analysis to automate plant classification. A number of researchers employed various techniques in trying to automate plant classification using plant leaves in the past. The most common plant leaf features that have been used are shape, texture, vein structure and color. Various scientific methodologies have been applied in trying to extract these features, including complicated procedures and calculations, after which the features would be fed into some classification algorithm like support vector machines (SVMs) for final classification. Kulkarni et al (2013) suggested an outline which identified and recognized plants by means of their texture, venation, shape and color, and these descriptors were joined with Zernike movements. Employing the radial-basis-probabilistic-neural-network (RBPNN) as the classifier, the proposed technique achieved a test accuracy score of 93.82% on the Flavia leaf dataset. In a comparable research, Kadir et al (2013) combined leaf texture features, leaf shape, color and venation to classify leaf images. They employed the use of a Probabilistic Neural Network (PNN) as a classifier and scored a 93.75% test accurate rate using the Flavia dataset.

2.2. The Anatomy of a Leaf

The leaf has a blade, which is flat, and the petiole holds that blade to the plant. Occasionally leaves can be separated into sections or leaflets, which may be more than two. If the blade is unbroken, it is referred to as simple while those with separated blades are termed compound. The cuticle is a tinny waxy layering the exterior surface of a leaf. The essential function of cuticle is to inhibit loss of water in the leaf meaning that plants that exist submerged in water do not have this layer. Right beneath the cuticle, there is a layer of cells named the epidermis. There are two nerve tissues that are located inside the leaf veins, and these are phloem and the xylem. Veins are structures that keep running from leaves to tips of the roots as far as possible up to the edges of the leaves. The external layer of the vein is made of cells called package

sheath cells, and they make a hover around the xylem and the phloem. Figure 1 below shows the leaf internal structure, xylem is marked in blue color while phloem is in red. Food or sugar is carried around through phloem while xylem is responsible for moving water around (Muskopf, 2018).

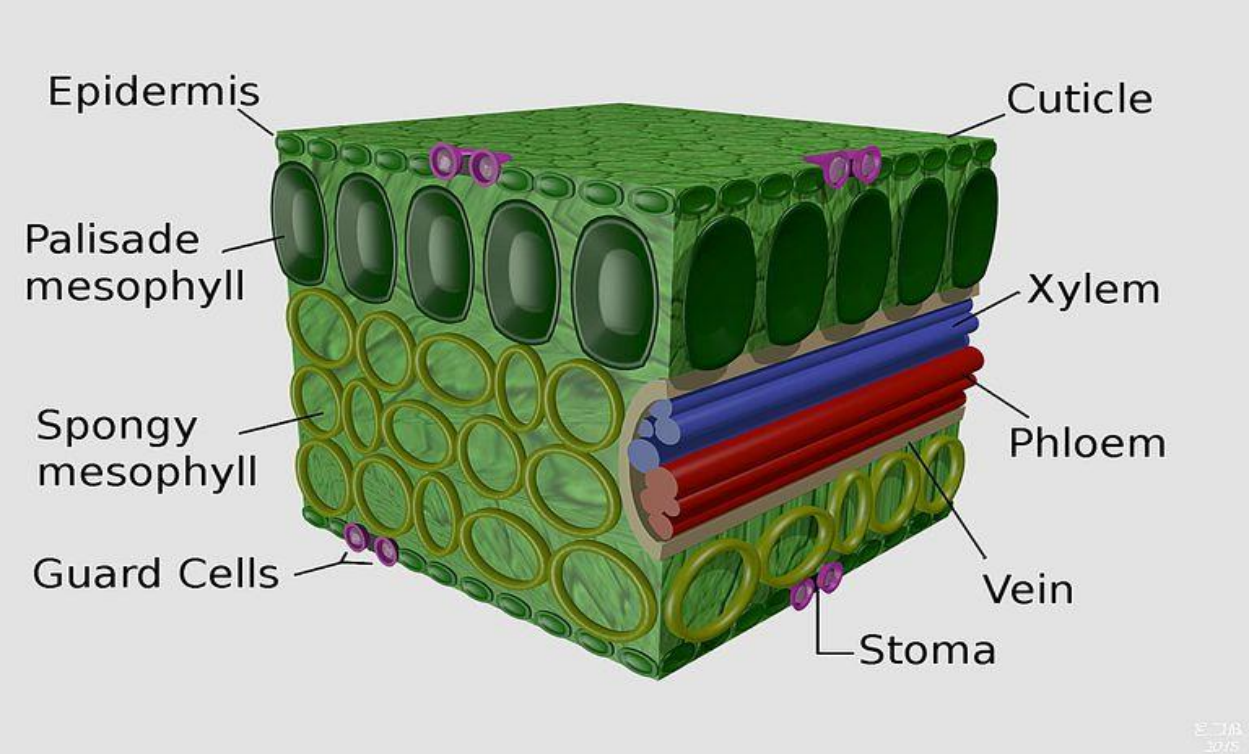

Figure 1: Leaf internal structure. Adopted from Vogel (2018)

The mesophyll is a layer found inside the leaf structure. There are two divisions of this layer, which are the palisade and spongy layers. Palisade cells are more segment like, and lie simply under the epidermis, while the supple cells are all the more approximately stuffed and lie between the palisade layer and the lower epidermis. There are spaces of air existing between the spongy cells, which are necessary for exchanging gas. Mesophyll cells are pressed with chloroplasts, and this is the place photosynthesis really happens. Epidermis additionally lines the lower region of the leaf (as does the fingernail skin). The leaf additionally includes small openings inside the epidermis called stomata. Specific cells, that are called guard cells encompass the stoma and are molded like two measured hands. Changes inside water weight cause the stoma to open or close. Figure 2 below shows the external structure of the leaf:

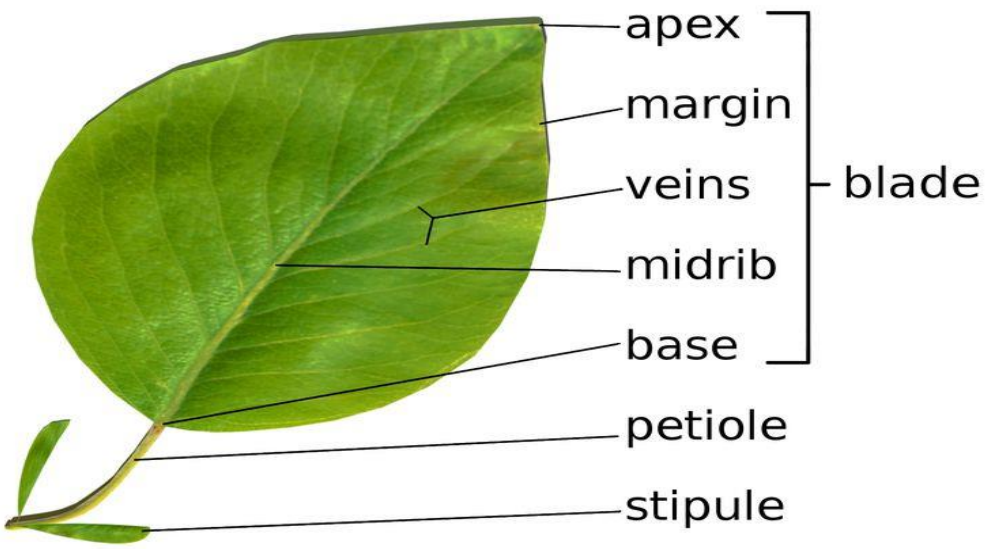

Figure 2: Leaf external structure. Adopted from Vogel (2018)

2.3. Leaf Taxonomy

Leaves vary from plant to plant. Figure 3 below shows common types of leaves:

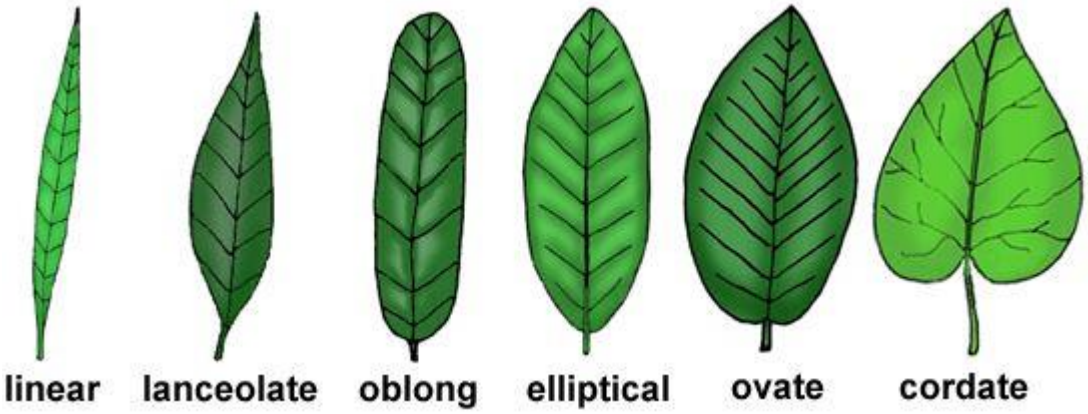

Figure 3: Common leaf types. Adopted from E. M. Armstrong 2002

Although there are variations, leaves can be classified into certain groupings by noting certain traits inherent in them. The Botanical website (Botanical-Online, 2018) categorizes leaves according to the: petiole, blade, edge, shape of the blade, veins, arrangement along the stem.

We shall discuss below some leaf features which are commonly used in literature to classify plants.

Classification using petiole

Petiolated leaves have a petiole, whose lengths differ from plant to plant. Figure 4 below shows how leaves can be classified according to their petiole:

Figure 4: Classification by petiole. Picture adopted from Botanical-Online (2018)

According to the blade

Simple leaves have an unbroken blade. In the few cases where they may have partitions, these will not stretch to the midrib. On the other hand, compound leaves will show a separated blade, with partitions going as far as the midrib. Occasionally, one more of these partitions will be alike a single leaf. These are named leaflets. Figure 5 shows the classification according to the blade:

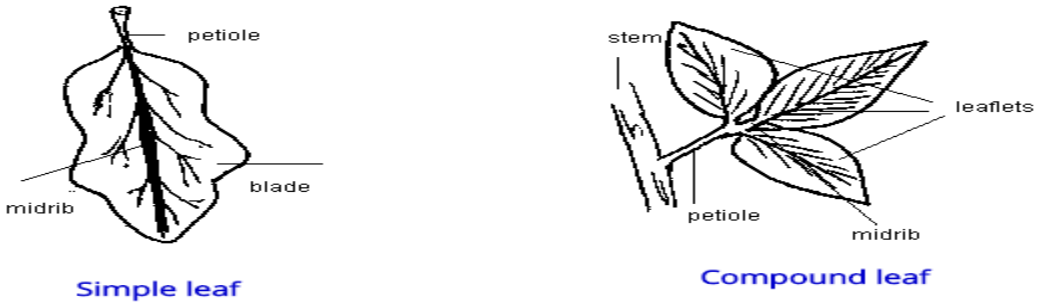

Figure 5: Classification according to blade. Picture adopted from (Botanical-Online, 2018)

Classification using the edge

In this case, the whole leaf will show a margin that is smooth. Those that will show some small curvatures which will be like waves are called sinuate leaves. Those that have bends similar to a saw are called serrate leaves. Those that have partitions that do not reach mid of half blade are called lobed leaves as in the Figure 6 below.

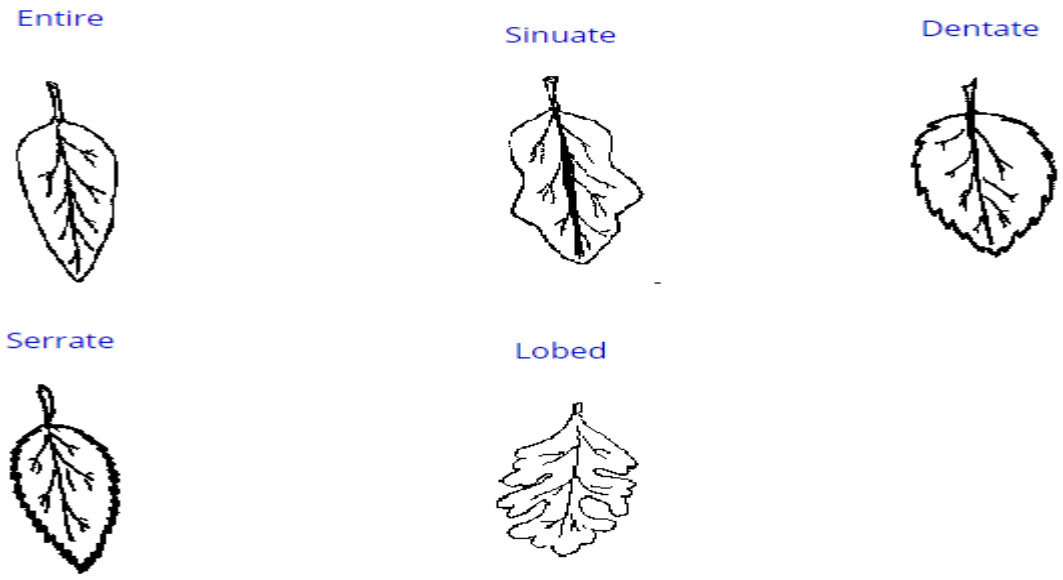

Figure 6: Classification by edge. Picture adopted from (Botanical-Online, 2018)

Classifying using the shape of the blade

Leaves that are strip-shaped are called linear. Those that have the shape of a heart are called cordate leaves. There are some that have the shape of the egg – ovate leaves. Their base is wider than their ovate counterparts. There are those that show three - partition style. There are also wider at the base and these partitions or leaflets are sharp at their edges. These are called hastate leaves. These have their length a number of times the size of their width. There are leaves that are shaped like a spear. These are called lanceolate leaves. The base gradually extends while it goes thinner at the apex. Acicular leaves have the shape of a needle, they are long than they are wide, and are sharp at the top end. Botanical-online (2018). Figure 7 below shows classification according to the blade shape:

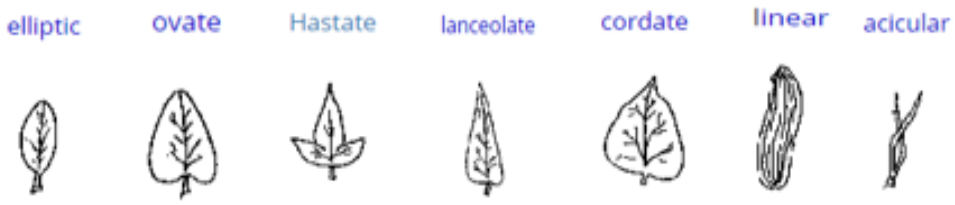

Figure 7: Classification according to blade shape. Picture adopted from (Botanical-Online, 2018)

According to the veins

Classification according to veins take into consideration the structure of the venation, where some have parallel veins, others have main nerve in the middle with branches extending from these main nerve, while others have diverging nerves. Those with parallel veins are referred to as parallel-veined. Those with a mid-main nerve are pinnate leaves, while the last ones are called palmate leaves. Figure 8 shows classification according to the veins:

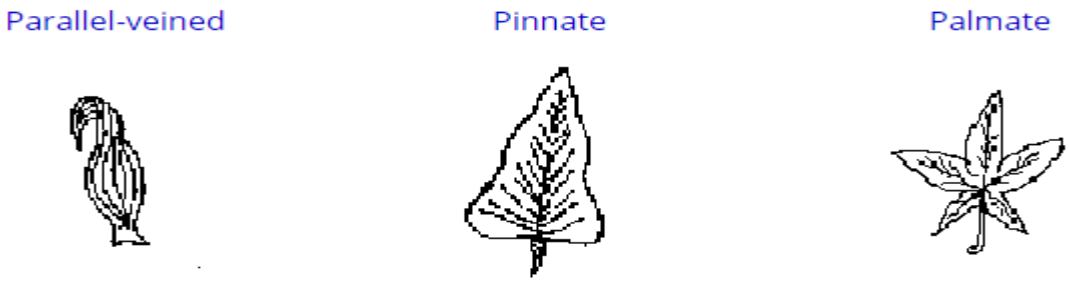

Figure 8: Classification according to veins. Picture adopted from (Botanical-Online, 2018)

2.4. Leaf Analysis Methods

A number of state-of-art methods have been applied in the recent past to classify plants according to their leaves. Much of the time, plants are recognized through their leaves. There are many species of plants in flora, and their value vary from being cash crops to being used for medicines in the medical industry. Therefore, the task of identifying plants is paramount for human beings. A variety of techniques and methodologies have been applied in the past to the subject of identifying plants. Features ranging from morphological to genetic have been used in the classification of a variety of leaves. The variedness of these characteristics across plants has presented major challenges and made the classification tasks an arduous exercise. Hence scientific methods have been employed to automate plant classification based on their leaves.

Waldchen and Mader (2016) did a systematic literature review of 120 peer reviewed papers selected from 2005 – 2015 that were done to automate plant classification. The result of the research showed that the most important feature in the classification of plants is shape. More than 50% of the studies conducted classification through shape. The shape of the flower was used by another 13 of the studies. Twenty-four (24) researchers applied the texture of the leaves in their research, whilst in another five (5) of the researches, flowers were investigated. Associated with color classification was flower analysis, counting to nine (9) of the researches. In five (5) studies, researches attempted to using color on classification of leaves themselves. Others went ahead to use some particular plant organs, for example venation, in

sixteen (16) of the studies. Lastly, the margin of the leaves were also used in another eight (8) of the works.

Identification using leaf shape

Human vision uses shape much of the time to recognize objects in the real world. Hence shape is a paramount feature as well in computer vision. Shape measurement follow the edges of an object, and determines structure of an object. In the real world, there are factors that may change around an object, such as translation, reflection, rotation or scaling. So the appropriate shape feature should remain constant to these variations. A research by Chaki and Parekh suggested an automatic technique for the classification and recognition of leaves using their shape. Two approaches were utilized here: one was using a prototypical called invariant-moments while the other one applied one centroid-radii. Appraisals between the two were made in relation to their classification scores. This methodology proved to be very suitable in the fast and effective classification of plants using leaf images. When compared with similar works in the research area, the accuracy of the method was within their range. Notably, this method has a clear advantage as it used the low-complexity data modeling scheme whereby feature vectors dimensionality were typically below 40. (Chaki & Parekh, 2011).

In another paper, the researchers exhibited three strategies of plants arrangement in view of their leaf shape; three models were employed in resolving many class issues – the Fourier Moment Technique, Support Vector Machine with Binary Decision Tree and Probabilistic Neural Network. A dataset of 1600 shapes of leaves taken from 32 various classes was used, and there were classes made of 50 samples of leaves of a like type. After a number of experiments with different algorithms, it was concluded that amongst the three models mentioned above, the Support Vector Machine with Binary Decision Tree scored higher than the two other methods. (Krishna , et al., 2010).

Although analyzing a leaf via shape is so popular, it does not come without its limitations. One of the challenges of using leaf shape as a feature emanates from the fact that some leaves

may have like margin traits but different shape or the opposite would be true. There normally exists high morphological variety crosswise over various species' leaves, and not just that - there is additionally frequently significant change among leaves of similar species. Results of the studies reveal that Simple and Morphological Shape Descriptors are excessively rearranged to separate leaves past those with vast contrasts, adequately and accordingly these are typically joined with different descriptors like texture and venation to produce better results. (Aakif & Khan , 2015) (Caballero & Aranda, 2010). Moreover, numerous single-value feature descriptions are exceptionally related with each other, making the exercise of picking adequately autonomous characteristics to recognize classifications of value particularly challenging. (Cope, et al., 2012).

Identification using Color of leaves

Color is a vital element of pictures. Color properties are characterized inside a specific shading space. There are a variety of color spaces that were used by different researchers for example hue-max-min-diff (HMMD) and hue-saturation-intensity (HSI). Others include hue-saturation-value (HSV) and red-green-blue (RGB). A variety of common color descriptions were suggested in the area of computer vision for classification of images according to Zhang et al (2012). These include color correlogram, color moments and color histograms. These are used as soon as the color space is defined where extraction of the features from the pictures happens. In a certain research, Kulkarni et al (2013) used color moments for classification of leaves. Principally, color moments are methods which can be successfully applied to separate pictures in light of their highlights of color. These are very useful to differentiate picture examination methods which use color according to Stricker and Orengo (1995). In the study, a score of 93.82% was achieved, when these color features were used with others like texture and venation. According to Waldchen and Mader (2016), the most researched was the Color Moments (CM) and this yielded results with high and reliable scores.

There are some disadvantages that come with color as a classification feature. A noteworthy test for color investigation is light varieties because of varied concentration and gloom of the

brightness coming from various edges. The adjustments in enlightenment may lead to shadowing impacts and light concentration changes. Pictures are taken under various conditions, and the variety in enlightenment can enormously influence the conclusion which come about. (Seeland, et al., 2016). In a certain research, it was realised that CM produced better results, when Yanikoglu et al (2014) examined the value of these color descriptions as features. Under the study, red-blue-green and CM won. The researchers additionally realised that color data did not add to the characterisation precision when joined with shape and surface descriptors. In another investigation too, utilising color data exclusively, without considering other features, demonstrated that color solely cannot be used to adequately classify flowers. (Nilsback & Zisserman, 2006). Drawing from the above notions, color as a feature is not strong enough, and needs to be augmented with other features.

Identification by the Texture of leaves

The word texture is used to designate the outward part of an object and is without doubt a primary component utilized in the field of image analysis. (Wechsler , 1980). The fundamental point of texture analysis is to computationally speak to an instinctive impression of texture and furthermore to empower the programmed handling of the texture data for models identified with computer vision. After having done this process, we get features related to leaf texture. There are broadly four classifications of the methods used for analysis of texture: those based on models, statistical methods, signal processing methods and structural methods. Some of the examples falling in the four categories include Fourier Descriptors and Gabor Filters. Similar to shape analysis, the computational process for getting these texture features is called feature extraction. There are a number of texture analysis methods that have been used by researchers.

In one study, Ehsanirad employed image analysis methods so that he could classify plants using recognition of their leaves. He chose the algorithms: Gray-Level Co-occurrence matrix (GLCM) and also Principal Component Analysis (PCA) in the process. These algorithms were trained using 390 leaves, and were to be used to classify plants falling in three categories, and the dataset about 65 new or distorted leaves for testing purposes. The PCA method

outperformed the GLCM method, with the former hitting a 98% accuracy compared to 78% of the later approach. (Ehsanirad, 2010). This showed the effectiveness of texture as a classification feature.

Although leaf texture has been exploited as a feature for the automated classification of leaves, it needs to be combined with other features to achieve better results. It will not consistently produce effective results on itself. One of the weaknesses of this method stems from the fact that texture is more closely determined by the sense of touch, however for the purposes of computer vision, the texture has to be derived from the image analysis. Although a number of description features for texture for the analysis of leaves such as Gabor filter (GF) (Casanova, et al., 2009), fractal dimensions (FracDim) (Backes & Bruno, 2009) and the gray level co-occurrence -matrix (GLCM) (Chaki et al 2015) have been used by researchers, there are still gaps as normal surfaces like leaf surfaces do not demonstrate perceptible semi -occasional structures yet rather have arbitrary relentless examples.

Identification by Leaf Vein Structure

Veins furnish leaves with structure and a vehicle instrument for supplies of different substances such as water and food. There are various types of leaf veins, be they parallel, some are palmate or could be pinnate as we noted before on leaf taxonomy. The structure of the vein of a leaf is distinctive to types of plants, so settling on vein structure for the purpose of leaf characterisation becomes an attractive idea. Because of a high complexity contrasted with whatever is left of the leaf sharp edge, veins are frequently plainly obvious. The analysis of leaf vein structure was suggested in 16 studies reviewed by examining leaf vein structure. These papers were looked into by Waldchen and Mader (2016). Of these, four of them exclusively examined venation and did not consider other features of leaves for example texture. Another twelve researches examined the veins structure together with leaf shape, and two of the works examined veins structure together with the three: color, shape and leaf texture. Kulkarni et al (2013) explored venation features, with others as well. They used morphological processes executed on the gray-scale-image of the leaves to extract the structure of the veins (Kulkarni, et al., 2013). The features of veins can be computed in three (3) varying ways, as is illustrated below:

F1=P1/P (Khalil & Bayoumi, n.d.)

F2=P2/P (Yuan, 2009)

F3=P3/P (Xiaoyi Song & Li, 2008)

Where:

- **F1, F2,** and **F3** characterize the features of the vein and
- **P1, P2,** and **P3** signify the total pixels of the vein, and
- **P** denotes total pixels present on the leaf.

As with other features mentioned above, venation also presents its challenges. Wang et al (2011) used Simple Canny Edge Detection (SCED) and Scale Invariant Feature Transform (SIFT) extricated from shape and vein test focuses. Their realization was that the pattern of veins are not at all times very useful for classification using SCED. Their conclusion came to be because of the distortions on outputs when they did their experiments using SCED on vein extraction: the use of these venation patterns in the perspective of shape led to a performance scoring that was not stable. To consolidate their findings, Bruno et al (2008) contends that the dissection of the veins structure of leaves is a difficult exercise. This, according to him, is caused by the low disparity between the veins structure of a leaf and the structure of the blade of leaves.

Identification by Leaf Contours

GWO and WEI (2013) proposed a component extraction technique for leaf shapes or contours, which portrays the lines between the centroid and each form point on a picture. A long histogram is made to speak to the circulation of separations in the leaf shape or contour. From there on, a classifier is connected from a measurable model to figure the coordinating score of the layout and question leaf. Success scores of at least 92.7% were achieved. This has limitations, as shown by the success rate achieved of only 92.7%.

There are other features that were employed by researchers like leaf margin. As noted with the other ones which have been studied above, there are also various weaknesses that are inherent in these features.

2.5. Conclusion

From the discussions above, it shows that feature extraction, and the corresponding decision to select the most suitable features to use for a particular automated leaf classification task is not straight forward. Could there be possibility of actually automating these two pre-processes that is, feature extraction and feature selection, and of course ultimately, the classification task based on these features? Deep learning has emerged to try to answer this question. A number of researches have gone into automating the plant classification through deep learning models. These models try to act like how the human brain processes vision. Such deep learning software try to learn and build the intelligence that humans possess. Data is passed from layer to layer as the features are extracted by such deep learning networks. The important fact to be noted is that the network learns features from the images that are presented to it. The next chapter talks about deep learning and how it has been adapted for use in this study.

CHAPTER 3: BACKGROUND AND METHODOLOGY

3.1. Introduction

This chapter talks about deep learning and how it has been adapted for use in this research. Deep learning is a sort of machine learning in which a model figures out how to accomplish classification tasks in a direct way from pictures, content, or sound. Deep learning is generally executed utilising a neural network framework. The expression "deep" alludes to the number of layers in the architecture - the more the layers, the deeper the architecture. Conventional neural networks contain just a few layers, while deep architectures can have many. Goodfellow et al. (2016) concur that machine learning is the main suitable way to deal with building Artificial Intelligence frameworks that can work in complex situations. Deep learning is a type of machine learning that achieves remarkable power and adaptability by approaching world problems as stacked ideas, with every idea described in connection to less complex ones, hence being able to simplify complex situations.

Advanced tools and techniques have dramatically improved deep learning algorithms - to the point where they can outperform humans in some tasks (MathWorks, n.d.). Goodfellow, et al (2016) noted that researchers have since a long time ago longed for making machines that think as people do. For example, a botanist can be able to classify leaves into their various categories with very high accuracy. A man's regular daily existence requires a gigantic measure of learning about the world. A bigger part of this learning is subjective and instinctive, and therefore hard to express formally. Computers need to acquire this same knowledge with a specific end goal to act in an aptitude way. One of the key difficulties in artificial intelligence is the way to get this casual information into a machine.

3.2. Deep learning vs. Machine Learning

Deep learning can be viewed as the investigation of models that include a more prominent measure of organisation of either learned capacities or educated ideas than conventional machine learning does. (Goodfellow et al, 2016). If we draw a diagram demonstrating how

these ideas are based over each other, the chart is deep, with numerous layers. Hence this approach to Artificial Intelligence (AI) is called deep learning. In particular, it is a sort of machine learning, a method that empowers machines to improve with information or data. There are various inspirations for deep learning, and these come from the weaknesses postured by machine learning. Deep Architectures can be authentically effective. They have less computational units for the same purpose. Deep learning models are good for vision, sound and normal dialect handling, to mention but a few situations. Machine learning work well on a wide variety of issues. That mentioned, they have not been forthcoming with regards to taking care of focal issues in AI, for example, perceiving speech or perceiving objects. The rise of deep learning was fueled to a lesser extent by the failure of conventional algorithms to perform well on such AI assignments. Table 1 below compares machine learning with deep learning:

Table 1: Comparison of machine learning vs. deep learning. Adopted from Mathworks (n.d.)

Machine Learning	**Deep Learning**
+ Good results with small data sets	Requires very large data sets
+ Quick to train a model	Computationally intensive
Must try unlike features and classifiers to realise higher performance	+ Learns features and classifiers automatically
Accuracy plateaus	+ Accuracy is unlimited

3.3. Image pre-processing

This section provides details on the pre-processing techniques used during the experimentations in this research. Building an effective deep learning framework demands cautious thoughtfulness of the network model as well as the format of the images to be processed. The architecture will be discussed later in this dissertation. The widely used traits of the images employed as input parameters are the total number of images, the height of images, their breadth, the count of channels, and levels for each pixel. Typically we have three (3) channels of data corresponding to the colors that is Red, Green, and Blue (RGB). Pixel levels are usually [0,255].

Uniform aspect ratio

The first step was to make sure that the leaf images have the same size and aspect ratio. Most of the neural network models assume a square shape input image, meaning each image must be checked whether it is a square or not, and then cropped appropriately. The pictures can then be cropped so that we remain with square images, as shown below in Figure 10. The input images chosen from the Flavia dataset were reshaped to squares.

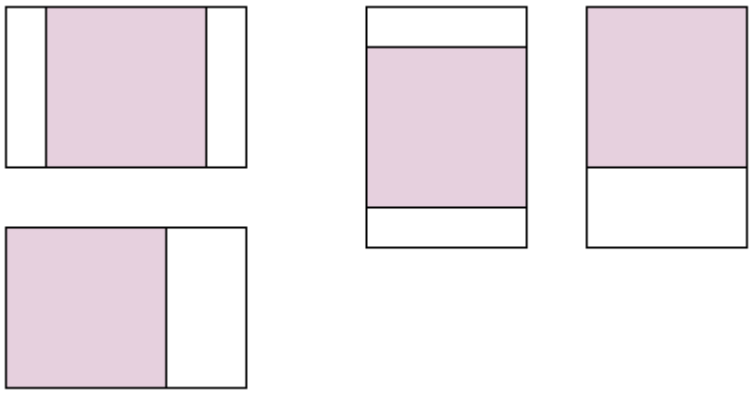

Figure 9: Image cropping into square images

Image Scaling

Deep learning models require images of certain dimensions to achieve better results. For example, a convolutional neural network may take as input tensors of shape (28, 28, and 3) representing image height, image width and image channels respectively. Once we have ensured that all images are square or have some predetermined aspect ratio, each image is then scaled appropriately. For the purposes of the experiments, images were transformed to dimensions of (150, 150, 3). Data-set images were converted into the described format before being fed to the deep learning model. There are a wide variety of up-scaling and down-scaling techniques and we usually use a library function to do this for us. OpenCV was also used for image scaling.

Normalizing image inputs

Data normalisation is a crucial procedure which makes sure that that every parameter input (or pixel, in the case of images) has a similar data distribution. This makes convergence faster while training the network. Data normalization is achieved by finding the difference of the mean from every pixel, and after that separating the outcome by the standard deviation. The distribution of this raw information would look like a Gaussian function aligned at zero. For image inputs we need the pixel numbers to be positive, so we choose to scale the normalized data in the range [0, 1] or [0, 255]. The [0, 1] was chosen as this is accepted by neural networks.

3.4. Feature Extraction

This section provides some details on how deep learning has automated feature extraction for leaf images classification. Deep learning enable machines to learn from experience and understand the world as a progressive system of ideas, with every idea characterized through its connection to less complex ideas (Stanford, 2017). Deep learning learns components and classifiers naturally not like machine learning, where there is requirement for feature engineering. Usually, it is impossible to realise what features ought to be taken, leading to the ineffectiveness of machine learning in such situations. With deep learning, accuracy is unlimited according to Bengio et al (2010). By gathering knowledge from experience, the methodology maintains a strategic distance from the requirement for people to formally determine all the learning that the computer requires. The chain of command of ideas empowers the computer to learn complex ideas by building them out of less complex ones. Figure 11 below shows that the lower levels of a convolutional neural network are responsible for feature extraction.

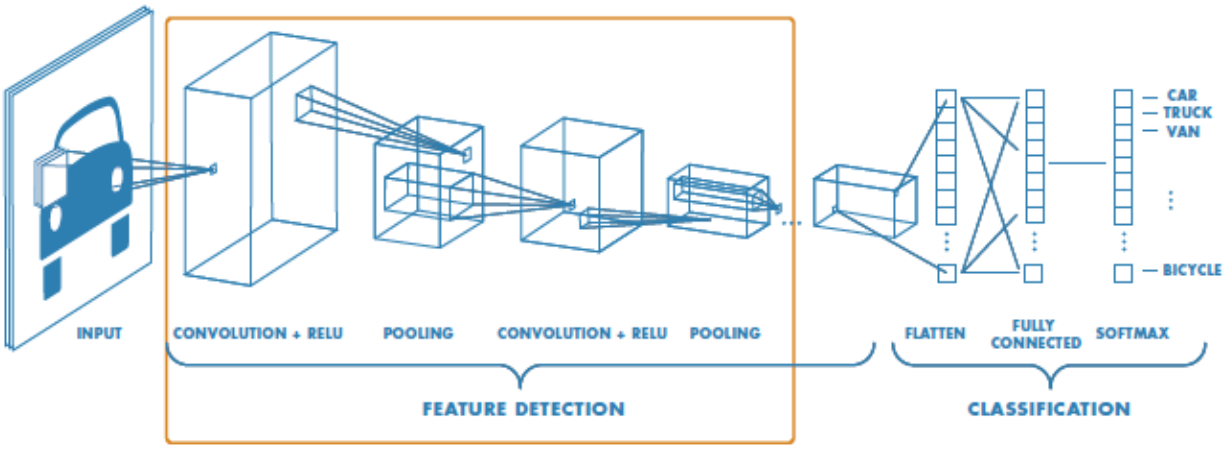

Figure 10: Feature detection layers of a CNN model (MathWorks, n.d.)

3.5. Convolutional Neural Networks

Convolutional neural networks (CNNs) are the quintessential deep learning models in computer vision and these models have been used in this research. Similar to neural networks, these have an input layer, an output layer, and also a number of hidden layers. Convolution networks are networks with linear operators, that is, confined convolution operatives employing certain core grid geometry. For example, consider the network whose **k**-th layer can be represented by the **m×m** grid:

Table 2: Grid representation of convolutional neural networks. Adopted from Goodfellow et al, (2016)

$h_{1,1}^{(k)}$	$h_{1,2}^{(k)}$	$\cdots$	$h_{1,m}^{(k)}$
$h_{2,1}^{(k)}$	$\cdots$		
$\cdots$		$\cdots$	
$h_{m,1}^{(k)}$			$h_{m,m}^{(k)}$

We then define the function: $h_{i,j}^{(k+1)}$ in layer $k+1$ by convolving over a *2×2* square in the layer below, and then applying the non-linear function *g*:

$$h_{i,j}^{(k+1)} = g\left(a^{(k)}h_{i,j}^{(k)} + b^{(k)}h_{i+1,j}^{(k)} + c^{(k)}h_{i,j+1}^{(k)} + d^{(k)}h_{i+1,j+1}^{(k)}\right)$$

The parameters a^k, b^k, c^k, d^k depend only on the layer, not on the particular square i, j. After convolving and applying $\boldsymbol{g}$ to obtain the grid-indexed functions,$h_{i,j}^{(k+1)}$, we replace these functions with the average or maximum of the functions in a neighborhood, which is called pooling. For example, setting:

$h_{i,j}^{-(k+1)} = \frac{1}{4}\left(h_{i,j}^{(k+1)} + h_{i+1,j}^{(k+1)} + h_{i,j+1}^{(k+1)} + h_{i+1,j+1}^{(k+1)}\right)$ (Goodfellow , et al., 2016)

CNNs are a particular sort of neural network for preparing information that has a known lattice like topology (LeCun, 1989). In general, these are neural networks that use the concept of convolution, replacing general matrix products in one or more layers. The neurons receive inputs, computes dot products and may follow it with an activation function. The network has one differentiable score function: image data is fed on one end and class scores are output. They have the objective function which needs to be minimised and all other elements of neural networks. The major variance is that these model explicitly assume that the inputs data are pictures. This enables practitioners and researchers to specify certain elements into the models. These then make the forward capacity more productive to actualize and limitlessly lessen the quantity of parameters in the network.

The layers of CNNs are particularly therefore organised in three (3) dimensions of width, height and depth. Each layer converts a three dimensional (3D) image volume to an output 3D image volume using some mathematical formula. There are basically three (3) main types of layers for CNNs, which are Convolutional**,** Pooling, and Fully-Connected Layer (Stanford, 2017).

3.4.1. Convolution

The convolution layer is the central component of a Convolutional Network as it does most of the computational hard work. Convolution moves the data through an arrangement of convolutional channels, every one of which enacts certain features from the depiction. The parameters at this layer comprise of an arrangement of learnable channels. Each channel stretches out through the full input volume. Each passage in the 3D yield could likewise be deciphered as a yield of a neuron that takes a gander at just a little area in the data and offers parameters with all neurons to one side and right.

There is Local Connectivity meaning every neuron is associated just to nearby area of the image volume, as opposed to being associated with all neurons in the past volume. The spatial degree of this availability is a hyper parameter called the open field of the neuron. The degree of the availability along the depth pivot is constantly equivalent to the depth of the input volume. Spatial course of action implies that three hyper parameters control the span of the yield volume i.e. the depth, stride, and zero-padding. The depth compares to the quantity of filters we might want to utilize, each learning to search for something else in the input. At the point when the stride is 1, we move the filters one pixel at any given moment. Figure 12 below shows the convolutional layers feeding into the fully connected layer of the CNN:

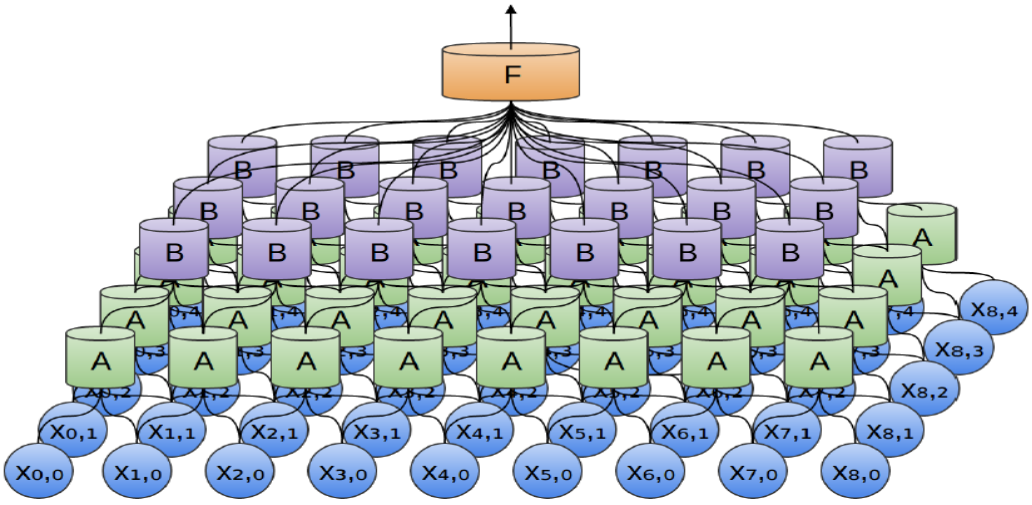

Figure 11: Convolutional layers (As, Bs) feeding to F – Fully connected layer. X is the input values

3.4.2. Pooling

Pooling streamlines the yield by performing nonlinear down examining, decreasing the quantity of parameters that the network needs to find out about. It is not uncommon to occasionally implant a Pooling layer amid progressive Convolutional layers in a CNN design. The goal is to dynamically reduce the spatial size of the portrayal to lessen the measure of parameters and calculation in the network, and consequently to likewise control over-fitting. The pooling layer works autonomously on each depth cut of the input and resizes it spatially, utilising the MAX task. The most widely recognized shape is a pooling layer with filters of size 2X2 connected with a stride of 2 down examples each depth cut in the input by 2 along both width and height, disposing of 75% of the actuations. Other than max pooling, there is additionally broad pooling, which can be normal pooling or even L2-standard pooling.

Rectified Linear Unit Layer (ReLu): This takes into consideration quicker and more efficient training by mapping negative outcomes to zero and keeping up positive outcomes.

3.4.3. Flattening

Flattening is a method for lessening the measurements of the input data with the goal that it can fit to pass through the last layer, which acknowledges one-dimensional vectors only. The flattening step is required with the goal that one can make utilisation of fully connected layers after some convolutional layers. Fully-connected layers do not have a neighborhood restriction like convolutional layers. This implies one can consolidate all the discovered local features of the past convolutional layers. This makes sure that each component outline in the yield of a CNN layer is a "straightened" two-dimensional (2D) cluster made by including the summation of numerous 2D pieces (one for each direct in the input layer). This implies it needs an element vector. So the outcome of the convolutional part must be changed into a one-dimensional (1D) vector, to be utilized by the fully connected layer. It gets the output from the previous layers, levels all its structure to make a solitary long component vector to be utilized by the dense layer for the last layer.

3.4.4. Full Connection

The layer just before the classification layer is a fully connected layer that yields a vector of K measurements where K is the count of classes that the system will have the capacity to classify. The vector will have the probabilities for each one of the class of any picture being presented. Neurons layer have full links with all actuations in the past layer, as found in neural network architectures. Their actuations can therefore be calculated through a matrix product after which a bias is applied. The last layer of this deep learning model usually utilizes a Softmax function to give the classifications. Figure 13 shows the classification layer which comprises of flattening, full connection and a classifier function:

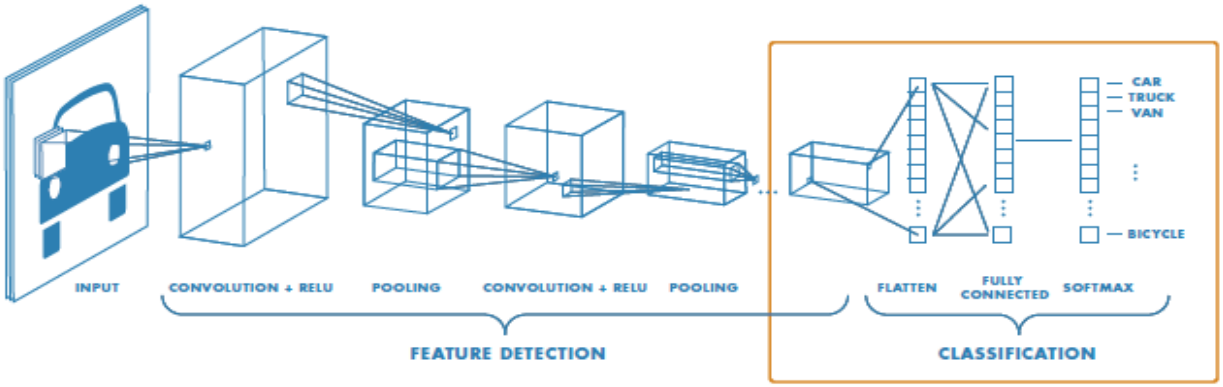

Figure 12: Classification layers (MathWorks, n.d.)

3.6. Conclusion

Deep learning has provided accuracy and reliability as an extension of machine learning. It has revolutionarised how machine learning is benefitting research and industry. One of the areas where deep learning clearly surpasses conventional machine learning is in the processing of unstructured data. The main headway of deep learning over all previous methods is its ability to automate the feature engineering step. In summary, it is an approach to machine learning that has benefitted insurmountably to our understanding of how the human brain operates, statistics and other knowledge bases. The years ahead are loaded with difficulties and chances to enhance profound adapting to deep learning and its furtherance. The next chapter is going to detail how deep learning has been applied specifically in the research.

CHAPTER 4: DATA AUGMENTATION AND TRANSFER LEARNING

4.1. Introduction

Convolutional neural networks were discussed in the previous chapter and that most commonly applied to analysing visual imagery. This chapter describes how they have been utilized in the identification of plants through leaf recognition. The challenge with deep learning is that it requires large amounts of data. A fundamental characteristic of deep learning is the models determine important features from the training data autonomously, so effectively eliminating hand-crafted feature engineering. This will only be achieved if there are lots of training examples, especially considering images, which are so much high dimensional. It is not possible to train a convolutional neural network to solve a complex problem with just a few tens of samples, but a few hundred can potentially suffice for a smaller architecture, and if the job at hand is easier. We are going to address the challenge of having too few samples and how to approach the problem with convolutional neural networks to achieve high performance rates in classifying leaf images.

4.2. Data Augmentation

In this research, one of the solutions that has been verified for improvement of plant recognition using plant leaves, in the context case of small datasets, is working on tweaking how data augmentation can be employed to improve test score. Data augmentation is a way of countering the challenge of having insufficient data on a convolutional neural network model. It is basically a way of synthetically modifying image data for example along orientation, width, length, color and other parameters. It is proved that the method can sufficiently overcome overfitting, verifying its usefulness in our context. The Flavia leaf dataset has got few examples per each category of images hence it becomes a perfect use case for this research on applicability of deep learning on small datasets. Even if one has a large dataset, augmentation still helps to increase the amount of relevant data in the dataset. This is related to the way in which neural networks learn and therefore applies to convolutional neural

networks. Figure 14 below shows how augmentation has been used in this research to produce six different copies from one leaf image:

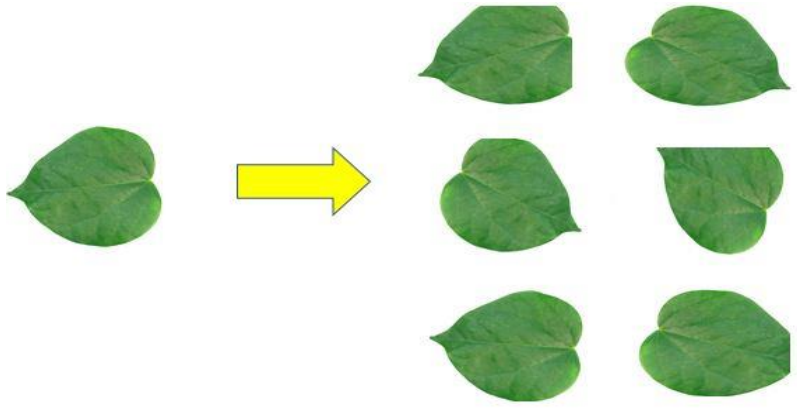

Figure 13: A leaf image augmented into its six (6) variations.

Insufficient data on deep learning causes overfitting. Overfitting is a phenomenon which happens when a model is too firmly fit to a restricted arrangement of input points. Overfitting the model for the most part appears as making an excessively complex model to clarify peculiarities in the information under investigation. In real world scenarios, the data under study will have some random distortions in it. Subsequently, trying to influence the model to adjust too nearly to somewhat off base information can contaminate the model with generous errors and diminish its discerning power. Overfitting is caused by having excessively few examples, making it impossible to gain from, rendering one unfit to prepare a framework that can derive important information from new data. If the data was unlimited, a framework would see every other aspect of the data, and overfitting would be impossible. During data augmentation, some transformations are systematically applied to training data to produce more of these samples. The target would be that the framework would not see an image more than once. This helps expose the model to more aspects of the data and generalize better.

To learn some real-world phenomenon means taking some examples of the phenomenon and selecting a model that describes them well. When such a model can also be used to describe

instances of the same phenomenon that it was not trained on we say that it generalises well or that it has a small generalisation error. The task of a learning algorithm is to minimize this generalization error. Figure 15 below shows how augmentation can be implemented in Keras:

```
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

Figure 14: A Keras implementation used for data augmentation.

4.2.1 Pre-cast Augmentation

In a number of literature, augmentation has mostly been applied on-the-fly. In this research, we have explored the use of augmentation offline, which we can refer to as pre-cast augmentation. The first method involves an image being passed through a model, with augmentation happening at the same time. This method makes augmentation not usable in the case where we want to utilize the convolutional base of a pre-trained model (as we shall explain) for feature extraction only, where the model has to be run on the whole dataset once only, hence seeing each image exactly once. The methodology in this research first augments images to a directory via a Python script, after which the model will be run for the samples in the same directory. This differentiated methodology gives state-of-art performance results with a very low computational cost, being able to be run on a cheaper computer. The Figure 16 below shows a folder containing an image augmented into six forms.

Figure 15: Showing implementation of pre-cast augmentation. The original image is named catB2.

As pointed out, this type of augmentation allows us to use an inexpensive form of deep learning with convolutional neural networks, where we can extract features by using a pre-trained model's base-layers, and use the base layers to extract features, which can be fed into a densely connected classifier to categorize the images. On-the-fly augmentation would not be applicable in this case, as the base model must see an image once only as highlighted above. Using this approach has allowed us to get state-of-art performance using a small capacity CPU. Normally, deep learning is capital intensive in terms of computing resources, requiring Graphical Processing Units.

4.3. Transfer learning with pre-cast augmentation

In this research, transfer learning is also verified, with particular reference to small data sets which are quite different from those with which the models, like VGG16, would have been initially trained on. Transfer learning is an exploration issue in machine learning that spotlights on putting away information picked up while taking care of one issue and applying it to an alternate however related issue. For instance, information picked up while learning to perceive cars could apply when endeavoring to perceive trucks. While most machine learning algorithms are intended to address single issues, the advancement of algorithms that encourage transfer learning is a theme of continuous research for the machine-learning network.

Transfer learning may enhance learning in three ways. First: is the underlying performance achievable in the objective mission utilising just the exchanged knowledge, before any further learning is done, contrasted with the underlying performance of an oblivious agent. Second: is the measure of time it takes to completely take in the objective mission given the exchanged knowledge contrasted with the measure of time to take in it sans preparation. Third: is the last performance level achievable in the objective mission contrasted with the last level without transfer. Use of a pre-trained model is wide and effective across researchers and practitioners in the case of small datasets. Pre-trained networks are those trained on bigger datasets. Given

the dataset used for this pre-trained network was large and broad, it means such type of network can be applied on similar small datasets. These can be used as well even for different datasets. There are two ways in literature to use a pre-trained network: either we use it for feature extraction or we do fine-tuning.

4.3.1 Feature extraction using a pre-trained network

Feature extraction was used in this research, focusing on the small Flavia dataset, employing pre-cast image data augmentation. The primary reason deep learning took off so quickly is that it offered better performance on many problems. Deep learning simplifies the task by automating the extraction or engineering of features which would be fed in machine learning models. Previous machine-learning techniques, which we could refer as shallow learning, only involved transforming the input data into one or two successive representation spaces, by using modest conversions through the use of for example Support Vector Machines or other algorithms. But the refined representations required by complex problems generally cannot be attained by such techniques. In that capacity, people needed to put everything on the line to make the underlying input data more manageable to handling by these techniques: they needed to physically engineer great layers of portrayals for their data. This is called feature engineering. Deep learning, then again, totally computerizes this progression: with deep learning, you take in all features in a single pass instead of engineering them. This has incredibly improved machine-learning work processes, regularly supplanting complex multistage pipelines with a solitary, straightforward, end-to-end deep-learning model.

Feature extraction comprises of utilising the representations learned by a past system to separate fascinating features from new examples. These features are then run through a new classifier, which is trained from scratch. Convolutional Neural Network start with a series of pooling and convolution layers, and they end with a densely connected classifier. The lowest layer referred to as the convolutional base of the framework. In the case of convolutional neural networks, feature extraction consists of taking the convolutional base of a previously trained network, running the new data through it, and training a new classifier on top of the

output. Figure 17 shows the employment of a pre-trained for feature extraction. Only the classifier is change, whilst base is frozen.

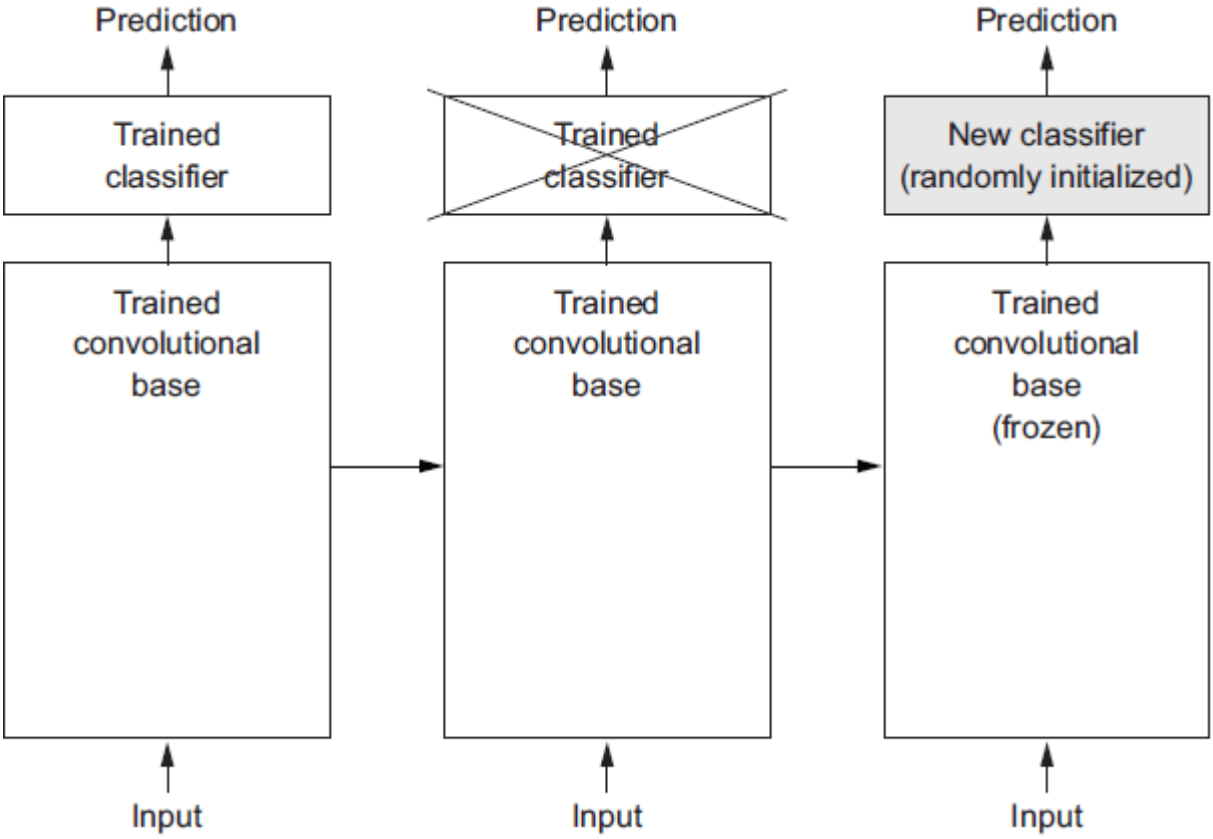

Figure 16: Swapping classifiers while keeping the same convolutional base. Adopted from (Chollet, 2018)

This is because the representations acquired this lower base will be more general and hence can be re-used: these maps taken from the features of a convolutional neural network are presence maps of generic concepts over a picture, which in most case, will be valuable for other computer vision problems encountered. Yet, the representations learned by the classifier will fundamentally be particular to the arrangement of classes on which the model was trained—they will just contain data about the nearness likelihood of either class in the whole picture.

The level of generality (and therefore reusability) of the representations extracted by specific convolution layers has direct relation with level of the layers. Layers that come earlier in the model extract local, highly generic feature maps (such as visual edges, colors, and textures), whereas layers that are higher up extract more-abstract concepts (such as “leaf venation”). So, if the new dataset differs a lot from the dataset on which the original model was trained, it will be better using only the first few layers of the model to do feature extraction, rather than using the entire convolutional base.

To re-use a pre-trained network which is computationally intensive, it was decided to run the convolutional base over the augmented dataset, recording its output, and then using this data as input to a standalone, densely connected classifier. This solution is fast and cheap to run, because it only requires running the convolutional base once for every input image, and the convolutional base is by far the most expensive part of the pipeline. Since this solution would run past one image once, hence the proposal to use pre-augmentation.

4.4 Conclusion

In this chapter, image data augmentation was reviewed, and an improvement to the way it is applied was verified. Transfer learning based on feature extraction was also discussed with a view to application on very small datasets, and its use with pre-cast augmentation was verified to be a methodology that can make deep learning work on small datasets, and on cheaper computers with Central Processing Units only. Deep learning for computer vision usually requires computers with Graphical Processing Units. The next chapter will describe how the experiments were conducted and will also discuss the results.

CHAPTER 5: RESULTS AND DISCUSSION

5.1. Introduction

The setting up and carrying out of the experiments, the results obtained and discussions are presented in this chapter. A Convolutional Neural Network is trained with selected data from the Flavia dataset. Succeeding experiments are done on the same dataset using data augmentation on the dataset. A method of using data augmentation first then using the convolutional base of a pre-trained network is then conducted, and is shown to yield higher, and more stable results overall.

5.2. Leaf Datasets for experimentation

Three leaf datasets, that is, the Flavia dataset, the UCI dataset and the LeafSnap dataset were investigated for use in the experiments. The Flavia and LeafSnap datasets were finally picked as they have a bigger number of samples per leaf category than the other two. Deep learning requires bigger datasets, so it was more appropriate to select Flavia and LeafSnap datasets with a view to augment it. LeafSnap has some categories which have above 100 samples per category. The experiments were also designed to have three separate classes for the data i.e. training, validation and test. Although validation and tests set appear to be similar, it is important to have a validation set in order to choose among different models. Below is the description of the datasets:

1. Flavia Database:

FLAVIA dataset contains more than 1600 leaf images from more than 32 different species of plants. This amounts to an average of 50 images per category. Each leaf image was captured with a high-resolution camera on a uniform white background. The plants used to create FLAVIA dataset are from the Nanjin University and the Yat-Sent arboretum. These plant species are common in the Yangtze Delta.

2. UCI Database:

UCI contains more than 400 leaf images from more than 32 different species of plants. This gives an average of 12 leaves per category, which number is very small for deep learning as we needed to test a Convolutional Neural Network from scratch. Each leaf image was captured with an IPad2 camera on a uniform background with one color. The 24-bit Red-Green-Blue images have a resolution of 720*920 pixels.

3. LeafSnap Database:

The LeafSnap dataset contains 185 species from the North-Eastern United States. LeafSnap is composed of 23147 Lab images and 7719 _eld images. The Lab images are pressed leaves, are of high quality, and they were obtained from the Smithsonian collection on controlled backlight. The _eld images are low quality images taken using mobile phones and are characterized by a varying amount of blur, noise, shadow and illumination which further complicates manual feature extraction process.

5.3. Experimental setup

5.3.1 System development environment

The deep learning system was implemented using Python Version 3.6.3 running on Anaconda Navigator implementing Jupyter Notebook. OpenCV image processing was also used to make some changes to the images. The computer had the following specifications:

- The processor: AMD E2-1800 APU with Radeon™ HD Graphics 1.70GHz
- The installed memory (RAM): 6.00 GB
- The system type: 64-bit Operating System, x64-based processor

The specification of the computer is of very limited use with deep learning models, as they are computationally intensive as to require Graphical Processing Units. Jupyter notebooks are a great way to run deep-learning experiments. They are extensively used in the data-science and machine-learning communities. A notebook is a file generated by the Jupyter Notebook application (https://jupyter.org), which can be edited in a browser. It mixes the ability to execute Python code with rich text-editing capabilities for annotating what one will be doing.

A notebook also allows to break up long experiments into smaller pieces that can be executed independently, which makes development interactive and means one does not have to re-run all of the previous code if something goes wrong late in an experiment.

5.3.2 Performance evaluation

Training accuracy, validation accuracy and test accuracies were used. Training accuracy is not relevant on its own but using it with validation accuracy will make it useful. The combination of the two can be used to detect overfitting, which has been fully described in the previous chapter.

The quintessential performance evaluation metric for deep learning models is the test accuracy. This is the percentage of the number of correctly classified units out of all classifications done, and usually this is given as a probability or percentage, and the highest score for an algorithm will be 100%. The common approach in measuring performance of artificial neural networks is splitting data into the training set and the test set and then training a neural network on the training set and using the test set for prediction.

5.3.3 Performance Assessment

For performance assessment, usually a record is done taking into account the false-positives and also the true-positives for each class predicted. The error rates are very important as they show how much our system can be trusted. The probabilities are useful to interpret the results if predicting one class is more vital than predicting another. First, one makes a prediction using the CNN and obtain the predicted class multinomial distribution:

$$\sum p_{class} = 1$$

If it is a top-1 score, then a check is made if the top class (the one having the highest score) is the same as the required label. If it is a top-5 score, a check is done if the anticipated value is one of the top 5 outcomes (the 5 ones with the highest scores). In all instances, the top score

is calculated computed at the times a predicted label matched the target label, divided by the number of data-points evaluated. Finally, when for example, 5-CNNs are used, we first average their predictions and follow the same procedure for calculating the top-1 and top-5 scores.

So, the classifier gives a probability for each class. For example, say we had leaf categories "category1", "category2", " category3", " category4" as classes (in this order). Then the classifier gives something like: 0.1; 0.2; 0.0; 0.7 as a result. The Top-1 class is “category 4". The top-2 classes are {category4, category2}. If the correct class was “category 2", it would be counted as "correct" for the Top-2 accuracy, but as wrong for the Top-1 accuracy. Hence, in a classification problem with k possible classes, every classifier has 100% top-k accuracy. The "normal" accuracy is top-1.

For the experiments done in this research, classification was limited to two categories per dataset.

5.3.4 Model choices

Although convolutional neural networks are so far the best models for classifying and recognizing images, one needs to be careful when selecting the parameters to use in a particular project. One has to carefully select the model to use, the loss function, the optimization function, the activation function, the classification functions to mention but a few. The below selections were made pertaining to the experiment at hand:

Deep learning model: Convolutional Neural Network using the Sequential Model.

Pre-trained network for transfer learning: VGG16 trained on Imagenet dataset

Loss function: Binary Cross Entropy was selected as two categories were chosen per dataset. This function describes the mechanism by which the model will be measuring its performance on the training data, and therefore how it will gravitate towards the right direction in terms of accuracy rates. The whole process takes the form of back propagation, where errors are propagated back to the network and weights are adjusted according using this loss function.

In information theory, the cross-entropy amid two probability distributions **a** and **b** over the same primary set of events measures the mean number of bits required to identify an event drawn from the set, if a coding scheme is used that is optimised for an "unnatural" probability distribution **b**, rather than the "true" distribution **a**.

This is defined as:

$$H(a,b) = E_a[-\log b] = H(a) + D_{KL}(a||b)$$

Where $H(a)$ is the entropy of a, and $D_{KL}(a||b)$ is the Kullback-Leibler divergence of **b** from **a**.

Optimiser: The mechanism by which the network updates itself based on the data it sees and its loss function. It specifies the exact way in which the gradient of the loss will be used to update parameters: for instance, it could be the RMSProp optimizer, SGD with momentum, and so on. The optimizer determines how the network will be updated based on the loss function. It implements a specific variant of stochastic gradient descent (SGD). RMSProp was chosen. RMSProp (for Root Mean Square Propagation) is a mechanism by which the learning rate is adjusted via the gradient descent, for the parameters. The notion is to partition the learning rate for a weight by a running normal of the extents of late gradients for that weight. Thus, first the running normal is computed as far as means square,

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)\big(\nabla Q_i(w)\big)^2$$

where, γ is the forgetting factor. The parameters are updated as:

$$w := w - \frac{n}{\sqrt{v(w,t)}}\nabla Q_i(w)$$

RMSProp has demonstrated great adjustment of learning rate in various applications. RMSProp can be viewed as a speculation of RMSProp and is skilled to work with scaled down groups too restricted to just full-batches.

Activation functions: ReLu and Sigmoid functions were used. These are shown in Figure 18 and Figure 19 below. In artificial neural networks, the rectifier is an activation function defined as the positive part of its argument: $f(x) = x^+ = \max(0, x)$ where *x* is the input to a neuron.

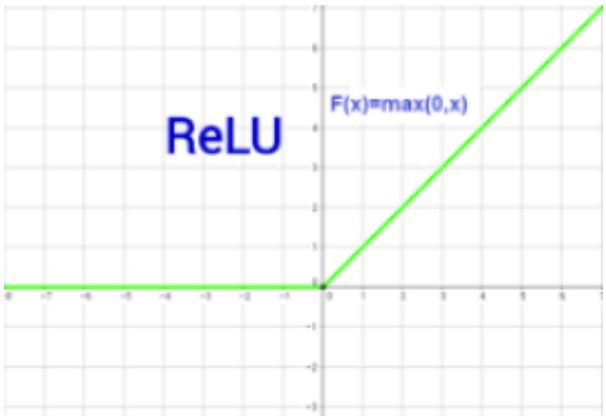

Figure 17: The ReLU Function

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve. It is defined as below: $S(x) = \frac{1}{1+e^{-x}} = \frac{e\hat{}x}{e^x+1}$

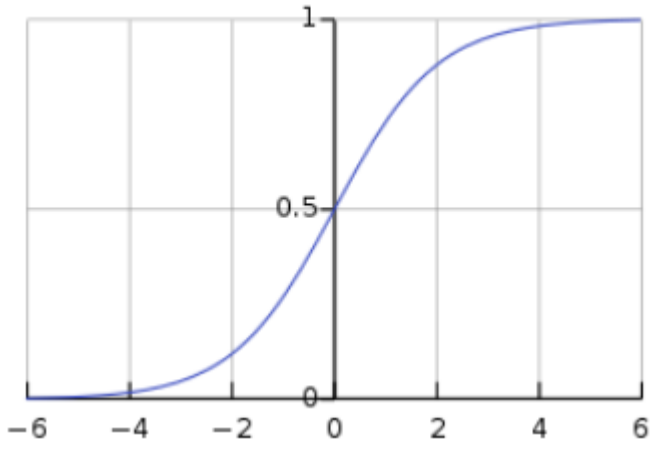

Figure 18: The Sigmoid function

5.4 Experimental Results

The experiments were all done on personal computer.

First experiments: Training the convolutional neural network from scratch

Two categories of leaf images from the Flavia Dataset of leaves were selected and were saved in a single base directory. A script was then used to split the data into Training, Validation, and Test directories, creating these directories, and then transferring the images to the directories. Each of these directories had two sub-directories for the two categories. Figure 20 below shows the directory structure of the input images:

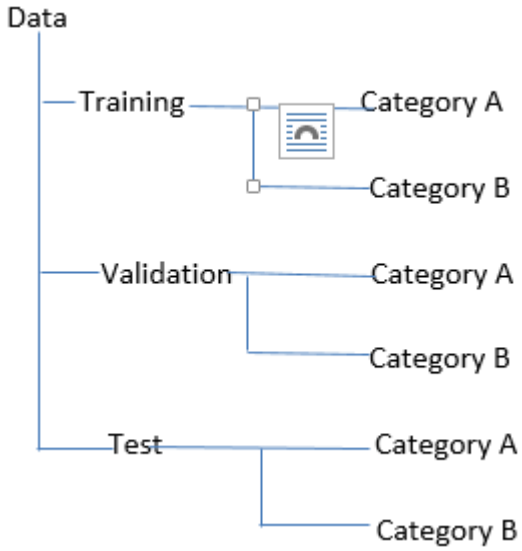

Figure 19: Directory structure of input images

Category A images – Figure 20 below shows the training set for category A leaves:

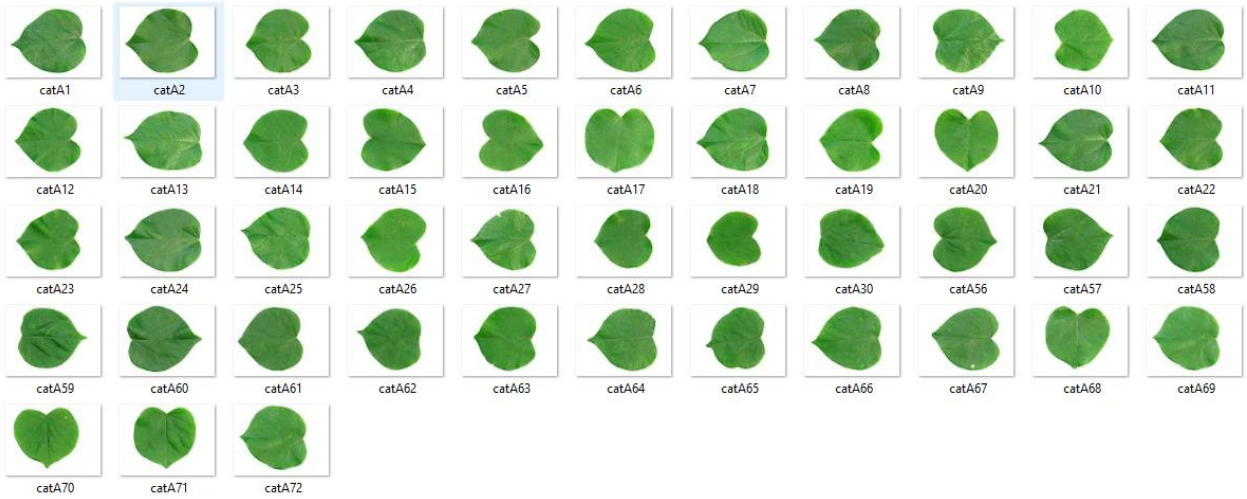

Figure 20: Training set for the first category of images from Flavia dataset

Category B images – Figure 21 below shows the training set for category B leaves

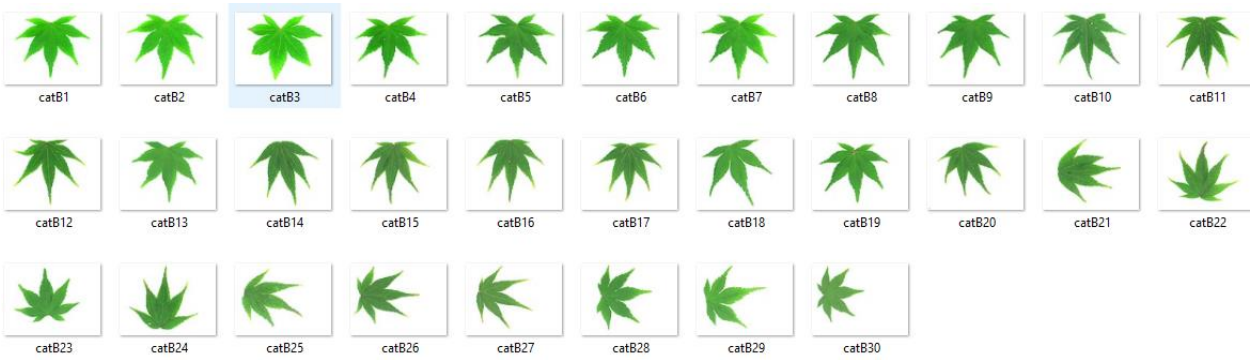

Figure 21: Training set for the second category of images from Flavia dataset

Below is the summary of the model that was used. It had four (4) convolutional layers, responsible for learning features from images samples, and a corresponding four (4) pooling layers. After that, since the fully connected layers accept one-dimensional units, the result from these layers will need to be flattened. The model has two fully connected layers, responsible for putting up the features back into full images for final classification. The number of parameters were above three million as shown below, which means it would take

some time to train on the laptop. Figure 23 below shows the shape of the convolutional neural network model that was trained from scratch. It has four (4) convolutional layers with a corresponding pooling layer after each convolutional layer. It has a flattening layer and then two dense layers:

```
In [11]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Figure 22: Shape of the CNN model for training from the scratch

Training the network took just short of eight (8) hours as shown below. Fifteen epochs (the number of times the algorithm is run over the data) were chosen.

```
Epoch 1/15
100/100 [==============================] - 1895s 19s/step - loss: 0.0514 - acc: 0.9860 - val_loss: 0.0087 - val_acc: 1.0000
Epoch 2/15
100/100 [==============================] - 1870s 19s/step - loss: 8.8378e-06 - acc: 1.0000 - val_loss: 0.0065 - val_acc: 1.0000
Epoch 3/15
100/100 [==============================] - 1876s 19s/step - loss: 1.4448e-07 - acc: 1.0000 - val_loss: 0.0031 - val_acc: 1.0000
Epoch 4/15
100/100 [==============================] - 1884s 19s/step - loss: 1.0969e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 5/15
100/100 [==============================] - 1904s 19s/step - loss: 1.0962e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 6/15
100/100 [==============================] - 1889s 19s/step - loss: 1.0960e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 7/15
100/100 [==============================] - 1883s 19s/step - loss: 1.0961e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 8/15
100/100 [==============================] - 1888s 19s/step - loss: 1.0960e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 9/15
100/100 [==============================] - 1903s 19s/step - loss: 1.0960e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 10/15
100/100 [==============================] - 1883s 19s/step - loss: 1.0963e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 11/15
100/100 [==============================] - 1883s 19s/step - loss: 1.0957e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 12/15
100/100 [==============================] - 1888s 19s/step - loss: 1.0961e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 13/15
100/100 [==============================] - 1902s 19s/step - loss: 1.0960e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 14/15
100/100 [==============================] - 1886s 19s/step - loss: 1.0960e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 15/15
100/100 [==============================] - 1881s 19s/step - loss: 1.0960e-07 - acc: 1.0000 - val_loss: 0.0025 - val_acc: 1.0000
Wall time: 7h 51min 56s
```

Figure 23: Training the network

.

Metrics for the model

Since the training dataset was small, the first iteration had managed to learn almost all the required features. The dataset was almost homogeneous as well. Comparing the training and the validation accuracy, we note that there was over-fitting, and the scores were high. The test accuracy would then validate if learning was generalized. Figure 24 below shows the training and validation accuracy. Figure 25 show the training and validation loss.

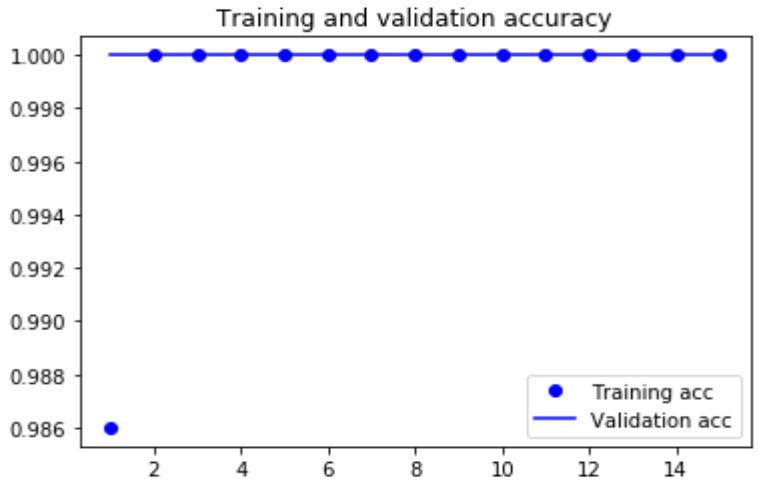

Figure 24: Training and validation accuracy

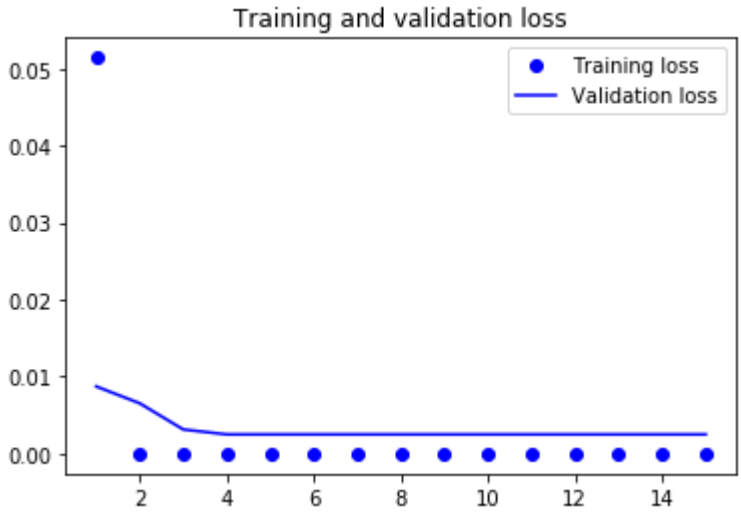

Figure 25: Training and validation loss

Test Accuracy

The most important metric, test accuracy, was 94.99%, which was way below the training and validation accuracy. The score is not bad but shows that our model was not exposed to as many training samples as optimal for its learning requirements. It also shows that although overfitting may seem not to have been detected, it actually might have happened but could not be detected given that a few samples were passed through the model. This gives rise to the

need to have bigger datasets for deep learning purpose. We will see how data augmentation will try to close this gap. Figure 26 below shows the average test score which was obtained:

```
In [18]: # Calculating the test accuracy
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

%time test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)

Found 20 images belonging to 2 classes.
Wall time: 12min 21s
test acc: 0.949999988079
```

Figure 26: Test accuracy using augmentation

Second experiments: Using data augmentation to mitigate overfitting

Category A augmented images – training set (transformed to 634 images): Figure 27 below shows that the training samples have been increased from about 30 images to 634 images. The snapshot shows some the augmented images.

Figure 27: Category A augmented images

Figure 28 below also shows that we now have 633 category B images transformed via augmentation. It shows part of the augmented category B images.

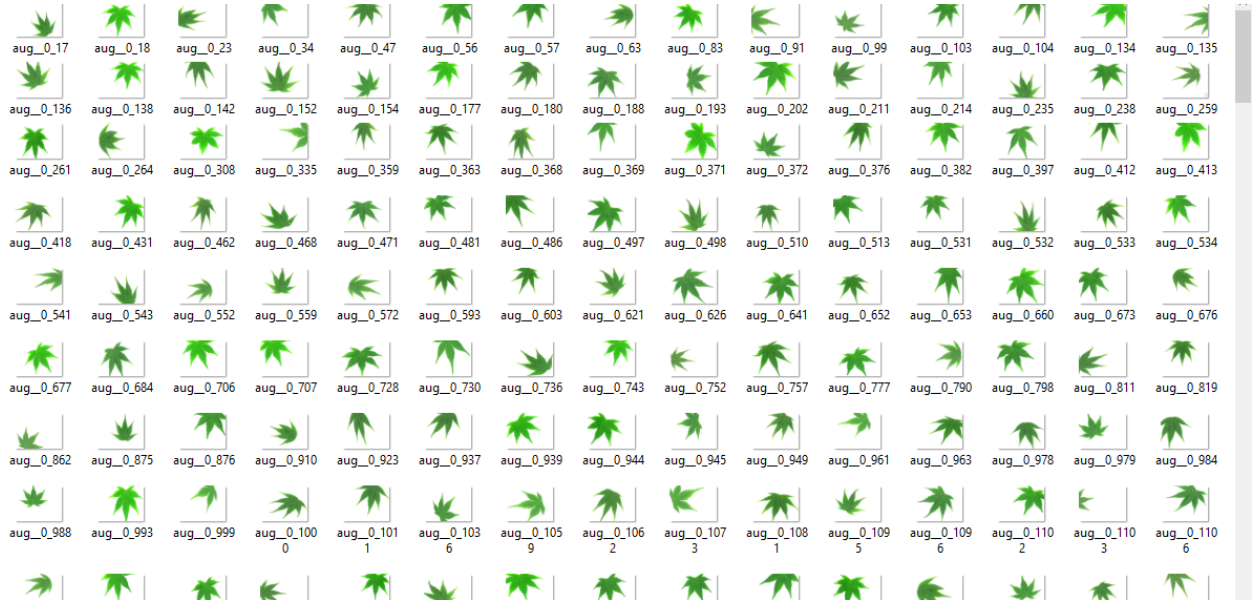

Figure 28: Category B augmented images

To verify the actual number of images, we count them via a Python function as shown in Figure 29 below:

```
In [30]: # print number of images in each folder after augmentation
         print('total training category A images:', len(os.listdir(train_catA_dir)))
         print('total training category B images:', len(os.listdir(train_catB_dir)))
         print('total validation category A images:', len(os.listdir(validation_catA_dir)))
         print('total validation category B images:', len(os.listdir(validation_catB_dir)))
         print('total test category A images:', len(os.listdir(test_catA_dir)))
         print('total test category B images:', len(os.listdir(test_catB_dir)))

         total training category A images: 634
         total training category B images: 633
         total validation category A images: 15
         total validation category B images: 15
         total test category A images: 10
         total test category B images: 10
```

Figure 29: Count of images after data augmentation

Training using the augmented images took nearly five (5) hours as shown below in Figure 30:

```
Epoch 7/15
100/100 [==============================] - 1112s 11s/step - loss: 0.0011 - acc: 1.0000 - val_loss: 3.2123e-05 - val_acc: 1.0000
Epoch 8/15
100/100 [==============================] - 1114s 11s/step - loss: 8.7898e-04 - acc: 0.9995 - val_loss: 2.3532e-05 - val_acc: 1.
0000
Epoch 9/15
100/100 [==============================] - 1119s 11s/step - loss: 8.6086e-05 - acc: 1.0000 - val_loss: 4.3318e-05 - val_acc: 1.
0000
Epoch 10/15
100/100 [==============================] - 1109s 11s/step - loss: 0.0023 - acc: 0.9995 - val_loss: 3.5886e-05 - val_acc: 1.0000
Epoch 11/15
100/100 [==============================] - 1114s 11s/step - loss: 9.1939e-04 - acc: 0.9995 - val_loss: 3.1274e-05 - val_acc: 1.
0000
Epoch 12/15
100/100 [==============================] - 1117s 11s/step - loss: 4.1317e-05 - acc: 1.0000 - val_loss: 1.8430e-05 - val_acc: 1.
0000
Epoch 13/15
100/100 [==============================] - 1110s 11s/step - loss: 0.0075 - acc: 0.9990 - val_loss: 2.7509e-05 - val_acc: 1.0000
Epoch 14/15
100/100 [==============================] - 1127s 11s/step - loss: 9.3378e-06 - acc: 1.0000 - val_loss: 1.6408e-06 - val_acc: 1.
0000
Epoch 15/15
100/100 [==============================] - 1123s 11s/step - loss: 7.2282e-05 - acc: 1.0000 - val_loss: 1.3847e-05 - val_acc: 1.
0000
Wall time: 4h 39min 1s
```

Figure 30: Training time for augmented images

Since we have more training samples, we see the shape of the training accuracy curving as expected, because learning usually is not as sharp as reflected in the previous experiments. Such is also the shape of the training loss curve. It now took the model upto five(5) epochs to learn the important features for the dataset, unlike in the previous experiments where it would take about one epoch. Figure 31 below shows this:

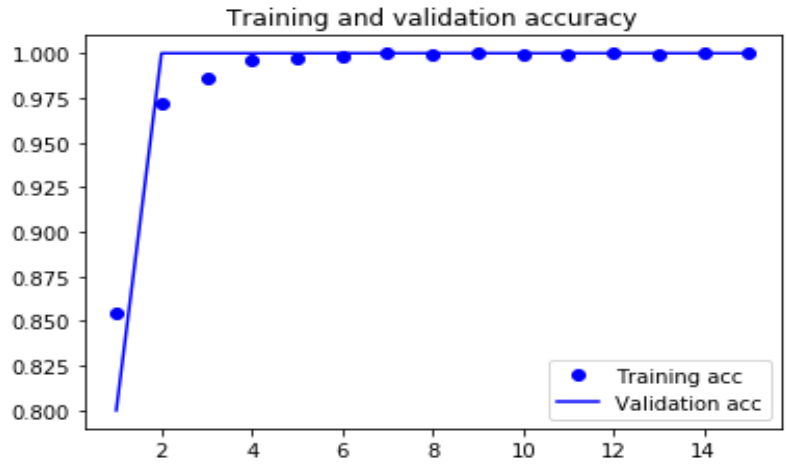

Figure 31: Training and validation accuracy for augmented images

As expected, the training and validation losses also reveal a similar gentle curving, reflecting that the model was exposed to a larger number of images, as shown in Figure 32 below:

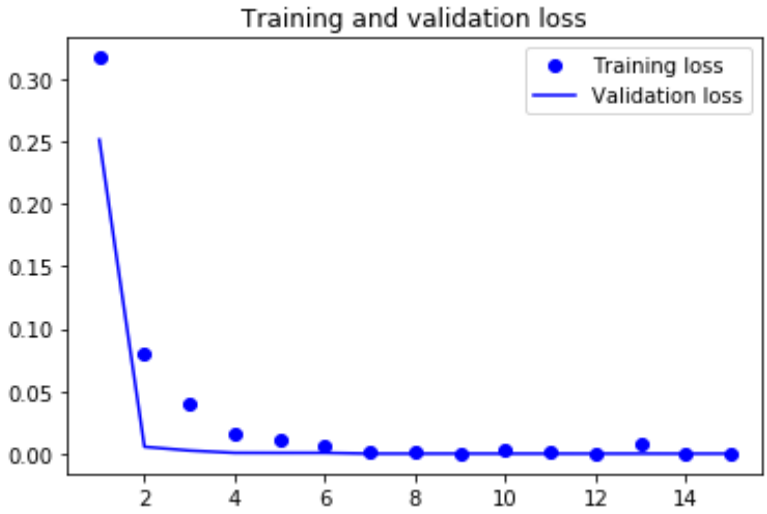

Figure 32: Training and validation losses

From Figure 33 below, it is shown that the test accuracy goes up to 100% because the model has learned enough as to generalize if given a new dataset.

```
In [37]: # Calculating the test accuracy
         test_generator = test_datagen.flow_from_directory(
                 test_dir,
                 target_size=(150, 150),
                 batch_size=20,
                 class_mode='binary')

         %time test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
         print('test acc:', test_acc)

         Found 20 images belonging to 2 classes.
         Wall time: 4min 45s
         test acc: 1.0
```

Figure 33: Test accuracy using augmentation

Test accuracy hits 100% because all the features have been learnt. However, the testing set has fewer samples, which is, of course not much of a worry, but a very big sample would probably make the score to just less than 100%. The next experiment is going to show that using transfer learning with data augmentation will yield better and more stable results, and that the feature extraction option will also take less computational power and therefore will

have less training time. Coupling this with the precast- augmentation methodology will make sure that the score is perfectly 100%!

Third experiments: Using a pre-trained model for feature extraction, with precast-augmentation

Here we use the VGG16 pre-trained from Imagenet. The first part of the model is called the convolutional base of the model. In the case of convolutional neural networks, feature extraction consists of taking the convolutional base of a previously trained network, running the new data through it, and training a new classifier on top of the output.

Figure 34 is an example of Keras implementation of the convolutional base:

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

Figure 34: Showing selection of the base of the CNN feature extraction

The reason we select the lower layers is because the representations learned by the convolutional base are likely to be more generic and therefore more reusable: the feature maps of a convolutional neural network are presence maps of generic concepts over a picture, which is likely to be useful regardless of the computer-vision problem at hand. Since our data is novel from the one used for Imagenet, it is only good for us to use the lower levels of the VGG16.

Figure 35 below is a summary of the convolutional base used for feature extraction with a whooping fourteen million plus parameters!

```
conv_base.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160


_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Figure 35: Shape of the model

Using transfer learning, it only took less than five (minutes) to use the base to extract features. This is reflected in Figure 36 below. Data preparation, of course, took some bit of time.

```
Epoch 14/15
1267/1267 [==============================] - 25s 20ms/step - loss: 4.2423e-07 - acc: 1.0000 - val_loss: 1.0960e-07 - val_acc:
1.0000
Epoch 15/15
1267/1267 [==============================] - 26s 20ms/step - loss: 1.0993e-06 - acc: 1.0000 - val_loss: 1.0960e-07 - val_acc:
1.0000
Wall time: 4min 33s
```

Figure 36: Training only took less than 5 minutes using extracted features.

Transfer learning performance metrics:

The metrics hit 100% as a pre-trained model was used. As can be seen below in Figure 39, it shows that transfer learning with data augmentation can be the way to go for various circumstances, when datasets are small. Figure 37 and 38 show the other metrics for the experiments that, training and validation loss and test accuracy and test loss.

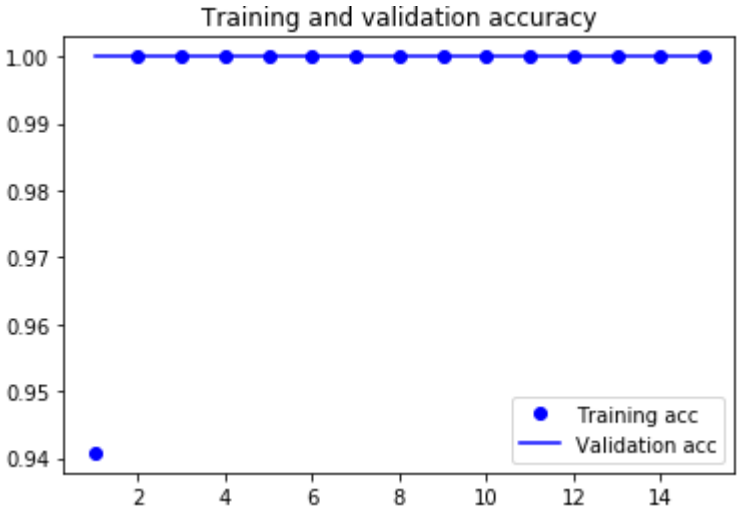

Figure 37: Training and validation accuracy with transfer learning

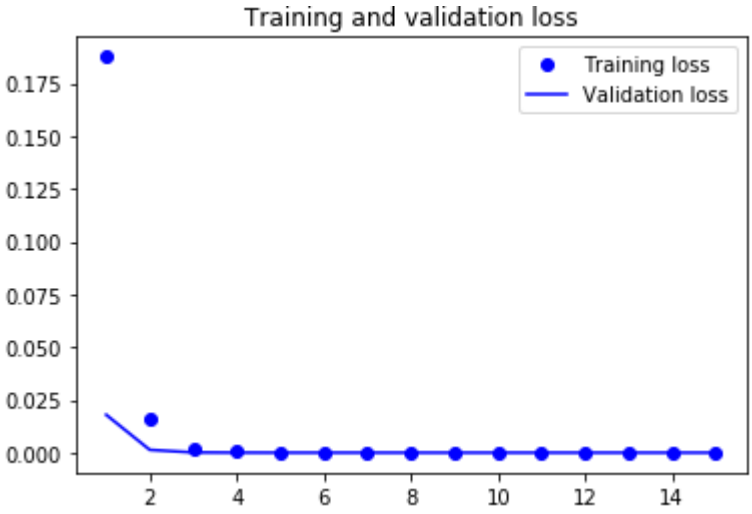

Figure 38: Training and validation loss with transfer learning

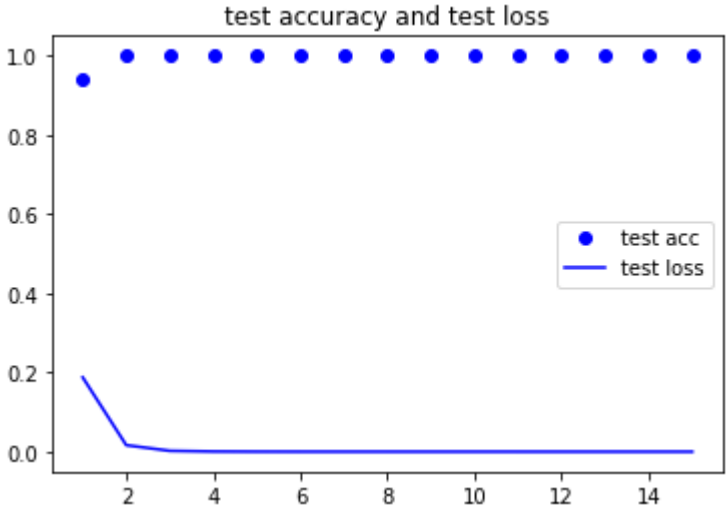

Figure 39: Test accuracy with transfer learning

It is interesting to note that using a pre-trained model managed to learn all important features for recognizing leaves in just less two (2) epochs. This is because the model has been trained with other datasets, and that knowledge has been transferred to the new dataset. Although the new dataset is novel from the one it was trained on, it has an upper hand as it has learnt the abstract concepts on feature extraction which can be applied to diverse datasets.

5.4.1 Summary of results

Table 3 and Figure 40 below shows the average percentages that were obtained from the experiments:

Table 3: Summary of experimental results

CNN Model	Pre-cast Augmentation	Training Score (%)	Validation Score (%)	Test Score (%)	Overfitting
Training from scratch	No	100	99.98	94.99	High
Training from scratch	Yes	100	100	99.99	Medium
Using Transfer Learning	Yes	100	100	100	Low

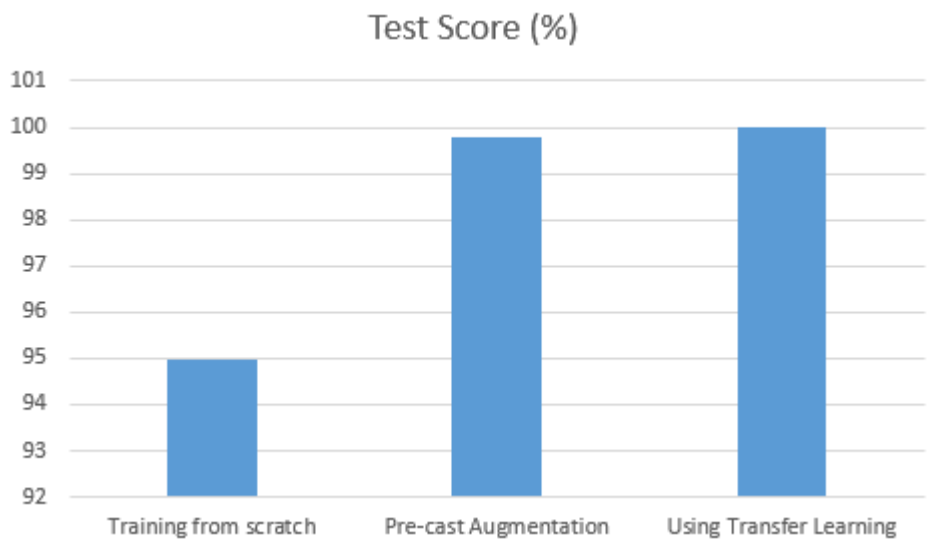

Figure 40: Summary of results

Comparison with similar past work

Table 5 below compares the results obtained in this research with previous results obtained by state-of-art methods of plant identification using machine learning and deep learning. A total of fourteen (14) researches have been examined. Although the circumstances may differ, the results from deep learning scored higher accuracy rates. On average, use of manual feature

extraction scored 89.86%. Using deep learning scored 97.93% on average, including those that were obtained in this research. It is to be noted that in some of these researches, the unavailability of large datasets would limit the accuracy rate for deep learning.

Table 4: Comparison of results with similar researches

Researcher	Use/Determination of leaf features	Category	Accuracy
(Chaki, et al., 2015a)	a) Using individual moment invariant(MI) on leaf features	Machine learning	88.90%
	b) Using joint 2-dimensional moment invariant(MI) on leaf features	Machine learning	95.50%
	c) Using joint 3-dimensional moment invariant(MI) on leaf features	Machine learning	93.30%
(Ehsanirad, 2010)	a) Using Grey Level Co-occurrence Matrix on leaf texture features	Machine learning	78.46%
	b) Using Principal Component Analysis (PCA) on texture features	Machine learning	98.46%
(Ehsanirad, 2010)	a) Using Geometric and morphological features: Probabilistic Neural Network with PCA	Machine learning	91.00%
	b) Using Geometric and morphological features: Support Vector Machine with Binary Decision Trees	Machine learning	96.00%
	c) Using Geometric and morphological features: Support Vector Machine with Fourier moments	Machine learning	62.00%
(Kulkarni, et al., 2013)	Using shape, vein, colour, texture, these combined with Zernike Moments	Machine learning	93.82%
(Kadir, et al., 2013)	Using shape, venation, colour, texture features with Probabilistic Neural Network	Machine learning	93.75%
(GWO 2 & WEI, 2013)	Using leaf contours	Machine learning	97.30%
(Lee & Chen, 2006)	Convolutional Neural Network with different classifiers	Deep Learning	99.60%
(Sladojevic, et al., 2016)	Deep Convolutional Neural Network for plant disease recognition	Deep Learning	96.30%
(Jeon & Rhee, 2017)	Convolutional Neural Network	Deep Learning	96.70%
This research	a) Convolutional Neural Network trained from scratch on a small dataset	Deep Learning	94.99%
	b) Convolutional Neural Network trained from scratch with augmentation applied to the small dataset	Deep Learning	99.99%
	c) Convolutional Neural Network with transfer learning and augmentation on the small dataset	Deep Learning	100.00%

Figure 41 below shows that results from deep learning are high and stable. It shows how precise the Convolutional Neural Network models are to determine the most important features to use for a particular scenario. The last three points denote the averages that were obtained in this research. The model reached a stable accuracy rate of 100% after tweaking the model and applying data augmentation.

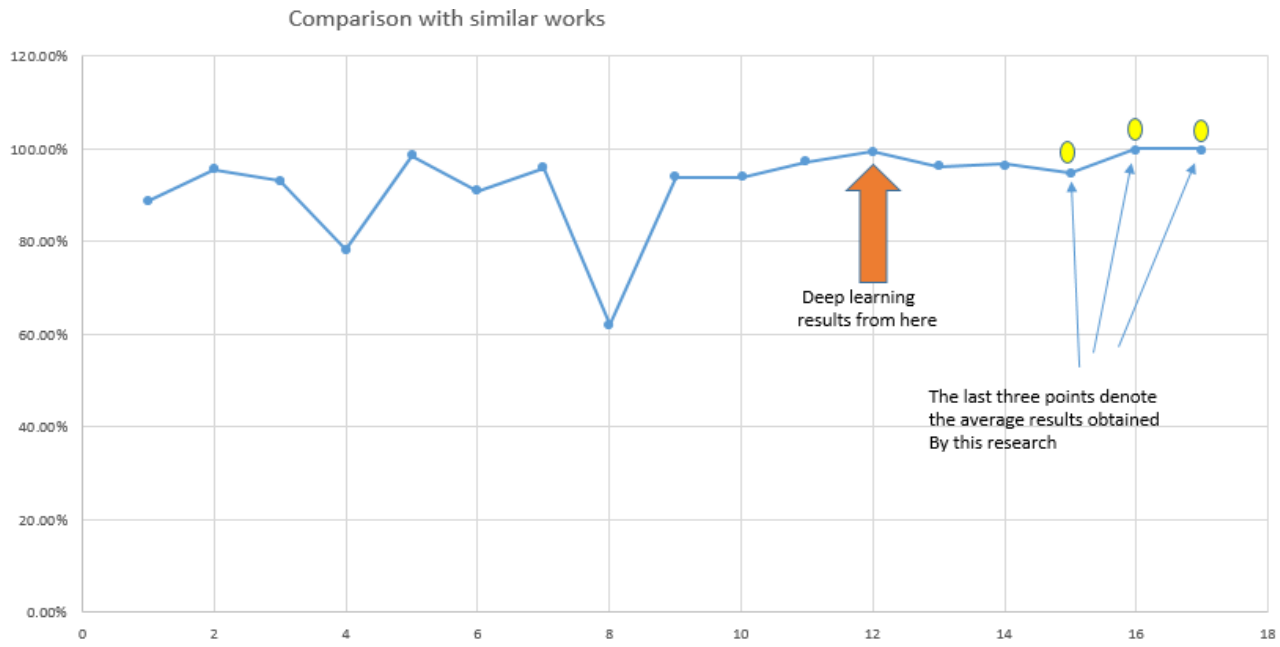

Figure 41: Comparison of results with other researches

5.6 Conclusion

The results of this research showed a performance improvement as augmentation and transfer learning were brought into perspective. Training the Convolutional Neural Network from scratch yielded just below 95% accuracy level. This shot up to 100% accuracy after employing pre-cast augmentation on the dataset. Results stabilized even more after employing transfer learning using the VGG16 pre-trained network, which was trained on Imagenet. The next chapter is going to draw on conclusion and future works.

CHAPTER 6: CONCLUSION AND FUTURE WORKS

This chapter provides a summary of the chapters in this dissertation, outlines briefly the contributions made by this research with reflections from the researcher and then proposes future work for researchers on the subject matter.

6.1 Summary of chapters

Chapter 1 introduces the dissertation, details the motivation behind the study, articulates the problem statement and the objectives of the study and finally outlines how the rest of the dissertation flows. Chapter 2 first briefly discusses leaf anatomy and leaf taxonomy, which knowledge is crucial for understanding classification by leaf identification. It then reviews the literature on the state-of-the-art techniques being used for plant classification, showing that the contemporary methods have a limitation, hence the need for deep learning. Chapter 3 outlines the background and methodology, discussing deep learning and its application in the research. Chapter 4 is dedicated to experiments with data augmentation and transfer learning showing that feature extraction is automated by using deep learning. Chapter 5 discusses the results obtained. Finally Chapter 6 outlines conclusion and future works.

6.2 Achievement of the research objectives

The research objectives were:

- To explore state-of-the-art methods for classification of plants based on leaf recognition.
- To improve the classification of plants via identification of leaves using CNNs.
- To model a framework for accurate classification of plants based on leaf recognition

State-of-the-art methods for automating plant classification via leaf identification were explored. Deep learning through convolutional neural networks was then inquired for possible higher performance, with data augmentation and transfer learning being systematically applied to the research, leading to improvements in accuracy scores. A step by step framework was then outlined for the model. The objectives of the research were met.

6.3 Reflections

Deep learning with convolutional neural network has the capacity to continue excelling in its application to computer vision problems. It will continue to surpass other contemporary methods in use for same tasks, especially because it automates the image feature extraction step for classification.

6.4 Contributions

Modern techniques aimed at the classification of plants through leaf recognition by means of machine learning have been explored in this research. The conclusion drawn is that deep learning produces higher and more accurate results as compared to these contemporary methods.

Deep learning was applied on the leaf image dataset, achieving high results. This has been achieved by fine tuning and varying the methodology of using data augmentation, applying it in a pre-cast form, differing with the many cases where it has been applied on-the-fly.

Data augmentation has also been used with transfer learning on the VGG16 pre-trained model for feature extraction, which resulted in higher and more stable classification accuracies. Normally, using this pre-trained model would require high computational power, but the methodology used allows its effective use with lower computing resources.

Finally, an outline of a framework for accurate plant classification using leaf recognition is presented in this dissertation. The deep learning framework has been developed with Python, Open CV, and Anaconda with Jupyter Notebook and Keras Deep learning model implementing Tensorflow in the backend.

6.5 Future works

The future works of this dissertation are based on the improvement of the proposed model, by finding other variants to data augmentation, followed by the conception of an ontology for plant classification. This will involve applying the same model to different datasets.

APPENDIX A

Flavia Leaf Dataset

Figure 42: Samples from the Flavia leaf dataset

APPENDIX B

UCI Dataset

Figure 43: UCI leaf dataset samples

APPENDIX C

Leafsnap Database

Figure 44: Samples from the leaf snap dataset

APPENDIX D

Python Codes

1. Used to augment images

```
#The following code was used for augmenting the images
#import the necessary libraries
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
import os
import glob

pwd = os.getcwd() # used to print the current working directory, if necessary

# the following object will transform the images according to the specifications
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

# augmentation is being applied to each image and the augmented images stored in a file
for filename in glob.glob('augment/from/*.jpg'):

    img = load_img(filename)  # this is a PIL image
    x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
    x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 150, 150)

        # the .flow() command below generates batches of randomly transformed images
        # and saves the results to the `preview/` directory
    i = 0
    for batch in datagen.flow(x, batch_size=1,
                              save_to_dir='augment/to', save_prefix='aug_', save_format='jpeg'):
        i += 1

        if i > 5:
            break  # otherwise the generator would loop indefinitely
```

2. Rename image filenames taken from the Flavia Dataset

```
#Collect sample leaves from the Flavia dataset. Select only two categories. Use the below script to rename the files per category
#Category A images and Category B images
import os
os.getcwd()

collection = "C:/Users/Innocent/Final Project/data/train/catA"
print('The pictures are currently located in directory"', collection ,'"')

for i, filename in enumerate(os.listdir(collection)):
    print("Number ", i+1,"filename is ", filename)
    os.rename("" + filename, "C:/Users/Innocent/Final Project/data/train/catA" + str(i) + ".jpg")
```

3. The model

```
# The model

from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()

#importing the optimiser
from keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])

from keras.preprocessing.image import ImageDataGenerator
```

```
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

#Model training
%time history = model.fit_generator(train_generator, steps_per_epoch=100, e
      validation_steps=50)

#Saving the model
model.save('two_categories_aug.h5')
```

```
#Plotting the graphs for results

import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

References

Rumelhart, D. E., Geoffrey, H. E. & R, J., 1986. Learning Internal Representations by Error Propagation. *Parallel distributed processing: Explorations in the microstructure of cognition,* Volume 1.

Aakif , A. & Khan , M., 2015. Automatic classification of plants based on their leaves. *Biosyst Eng,* Volume 139, p. 66–75.

Adetiba, E. & Olugbara, O., 2015. 65, 72. Improved classification of lung cancer using radial basis function neural network with afine transforms of voss representation. *PloS one,* 10(12), pp. 65, 72.

Ahmed, N., Ghani Khan, U. & Asif.Shahzad, 2016. An automatic leaf based plant identification. *Science International,* 28(1), p. 10.

Aimen, A. & Faisal, K. M., 2015. Automatic classification of plants based on leaves. *Biosystems Engineering,* Volume 139, pp. 66-77.

Al-Hiary, H. et al., 2011. Fast and accurate detection and classification of plant diseases. *Machine learning,* Volume 14, p. 5.

Anon., 2018. *Investopedia.* [Online]
Available at: https://www.investopedia.com/terms/o/overfitting.asp
[Accessed 30 05 2018].

Azencott, R., Wang, J. & Younes, J., 1997. Texture classification using windowed Fourier filters. *IEEE Trans Pattern Anal Mach Intell,* Volume 19, pp. 148-153.

Backes, A. & Bruno, Q., 2009. Plant leaf identification using multiscale fractal dimension.. *Image analysis and processing ICIAP,* Volume 5716, p. 143–150.

Beghin, T., Cope, J. S., Remagnino, P. & Barman, S., 2010. Shape and texture based plant leaf classification. In: *Advanced Concepts for Intelligent Vision Systems.* s.l.:s.n., p. 345–353.

Bengio, Y., Hinton, G., LeCun, Y. & Ng, A., 2010. *Workshop on Deep Learning and Unsupervised Feature Learning,* University of Michigan: NIPS .

Blum, H., 1967. *A transformation for extracting descriptors of shape..* s.l.:s.n.

Botanical-Online, 2018. *Botanical Online.* [Online]
Available at: https://www.botanical-online.com/hojastiposangles.htm
[Accessed 17 01 2018].

Bruno , O., de Oliveira , P. R., Falvo , M. & de Castro, 2008. Fractal dimension applied to plant identification. *Information Sciences,* 178(12), p. 2722–2733..

Burges, C. J., 1998. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery,* 2(2), pp. 121-167.

Caballero , C. & Aranda, M., 2010. Plant species identification using leaf image retrieval.. *Proceedings of the ACM international conference on image and video retrieval,* p. 327–334.

Cartwright, H., 2015. *Artificial Neural Networks.* s.l.:Humana Press.

Casanova, D., de Mesquita, S., Junior, J. & Bruno, O., 2009. Plant leaf identification using gabor wavelets.. *Int J Imaging Syst Technol ,* 19(3), p. 236–243.

Chaki, J. & Parekh, R., 2011. Plant Leaf Recognition using Shape based Features and Neural Network classifiers. *(IJACSA) International Journal of Advanced Computer Science and Applications,* 2(1o).

Chaki, J., Parekh, R. & Bhattacharya, S., 2015a. Plant leaf recognition using texture and shape features with neural classifiers. *Pattern Recognit Lett,* Volume 58, p. 61–68.

Chakraborty, S., Tiedemann, V. A. & Teng, P. S., 2000. Climate change: potential impact on plant diseases. *Environmental Pollution,* 108(3), p. 317–326 .

Champ, J. L. T. S. M. J. A., 2015. A comparative study of classification methods in the context of the lifeclef plant identification. *CLEF,* Volume 1391.

Charters, J. et al., 2014,. Eagle: A novel descriptor for identifying plant species using leaf lamina vascular features. In: *ICME-Workshop.* s.l.:s.n., p. 1–6..

Chollet, F., 2018. *Deep Learning with Python.* 1st ed. Shelter Island: Manning Publications Co.

Ciresan Dan, C., Meier, C. & Gambardella, L. M., 2011. Flexible, high performance convolutional neural networks for image classification. *In Proceedings of the International Joint Conference on Artificial Intelligence,* 22(1), p. 1237–1242.

Clausi, D. A., 2002. An analysis of co-occurrence texture statistics as a function of grey level quantization. *Can. J. Remote Sensing,* 28(1), pp. 45-62 .

Coakley, S. M., H. Scherm & Chakraborty, S., 1999. Climate chnage and plant disease management. *Annual Review of Phytopathology,* 37(1), pp. 399–426, .

Cope, J. et al., 2012. Plant species identification using digital morphometrics: a review. *Expert Syst Appl,* 39(8), p. 7562–7573.

Cope, S. J. et al., 2012. Plant species identification using digital morphoetrics: A review. *Expert Systems with Applications,* 39(8), pp. 7562-7573.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. In: *Mathematics of Control, Signals, and Systems, 2(4),*. s.l.:s.n., p. 303–314.

Daugman, J. & Downing, C., 1995. Gabor wavelets for statistical pattern recognition. *In The handbook of brain theory and neural networks, M.A. Arbib,* Volume MIT Press, Cambridge, MA, 1995, pp, p. 414–419.

Davies, R. E., 2004. *Machine vision: theory, algorithms, practicalities.* s.l.:Elsevier.

Du, J.-X., Wang, X.-F. & Zhang, G.-J., 2007. Leaf shape based plant species recognition. *Applied mathematics and computation,* Issue 4, 5, 13, 21, 47, 67, pp. 883-892.

Ehsanirad, A., 2010. Plant Classification Based on Leaf Recognition. *International Journal of Computer Science and Information Security,,* July.8(4).

Ehsanirad, A., 2010. Plant Classification Based on Leaf Recognition. *(IJCSIS) International Journal of Computer Science and Information Security,,* July.8(Vol. 8,).

Garrett, K. A. et al., 2006. Climate change effects on plant disease: genomes to ecosystems. *Annual Review of Phytopathology,* Volume 44, pp. 489-509.

Ge, Z. M. C. C. P., 2015. Content specific feature learning for fine-grained plant classification. *IWorking notes of CLEF* .

Goodfellow , I., Bengio , Y. & Courvil, A., 2016. *Deep Learning.* s.l.:MIT Press.

Graham, J., 2000. *Application of the Fourier-Mellin transform to translation-, rotation and scale scale invariant plant leaf identification.* Montreal: McGill University.

Graves, A. et al., 2009. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 5(31).

Grosky, W. I. & Mehrotra, R., 1990. Index-based object recognition in pictorial. *Computer Vision, Graphics, and Image Processing,* 52(3), pp. 416-436.

GWO 2 , C.-Y. & WEI, C.-H., 2013. PLANT IDENTIFICATION THROUGH IMAGES: USING FEATURE EXTRACTION OF KEY POINTS ON LEAF CONTOURS. *Applications in Plant Sciences,* 1(11: 1200005), p. 1 (11):.

Haralick, R., 1979. Statistical and structural approaches to texture. *IEEE,* Volume 67, pp. 786-804.

Hastie, T. T. R. F. J., 2009.. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. In: New York: Springer, NY.

Hawkins, D. M., 2004. The problem of overfitting. *Journal of chemical Information and Computer Sciences,* 44(1), pp. 1-12.

Hearn, D. J., 2009. Shape analysis for the automated identification of plants from images of leaves. *Taxon,* 58(3), pp. 934-954.

Helmut, A. & Feran., H., 2008. Packing convex polygons into rectangular boxes. *In Japanese Conference on Discrete and Computational Geometry,* Volume 35, pp. 67-80.

Hickey, L. R., 1973. *Classification of architecture of dicotyledonous leaves.* s.l.:AM Jbot.

Hof, Robert D, 2018. *MIT Technology Review*. [Online]
Available at: https://www.technologyreview.com/s/513696/deep-learning/
[Accessed 23 01 2018].

Howse, J., 2013. *OpenCV ComputerVision with Python.* Birmingham: Packt Publishing.

Idrissa, M. & Acheroy, M., 2002. Texture classification using Gabor filters Pattern Recognition. *Pattern Recognit Lett 23,* p. 1095–1102.

Jain, A. K. & Farrokhnia, F., 1991. Unsupervised texture segmentation using Gabor filters. *Pattern Recognit 24* , p. 1167–1186.

Jeon, W.-S. & Rhee, S.-Y., 2017. Plant Leaf Recognition Using a Convolution Neural Network. *International Journal of Fuzzy Logic and Intelligent Systems,* 17(1), pp. 26-34.

Jia, Y. et al., 2016. Multimedia life species identification. *Lifeclef 2016.*

Kadir, A., Nugroho, L. E. & Susanto, A., 2011. Leaf Classification Using Shape, Color, and Texture Features. *International Journal of Computer Trends and Technology,* July to Aug.Issue 2011.

Kadir, A., Nugroho, L. E., Susanto, A. & Santosa, P. I., 2013. *Leaf classification using shape, color, and texture features,* s.l.: arXiv preprint arXiv:1401.4447 .

Kalyoncu , C. & Toygar, Ö., n.d. *Geometric leaf classification.* [Online]
Available at: http://dx.doi.org/10.1016/j.cviu.2014.11.001.

Khalil , M. & Bayoumi, M., n.d. A dyadic wavelet a_ne invariant function for 2D shape recognition. *IEEE Trans.*

Krishna , S., Indra, G. & Sangeeta , G., 2010. SVM-BDT PNN and Fourier Moment Technique for Classification of Leaf Shape. *International Journal of Signal Processing, Image Processing and Pattern Recognition,* December.3(4).

Krizhevsky, A., Sutskever, I. & Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25,,* pp. 1097 - 1105.

Kulkarni, A. H., Rai, H. M., Jahagirdar, K. A. & Upparamani, P. S., 2013. A Leaf Recognition Technique for Plant Classification Using RBPNN and Zernike Moments. *International Journal of Advanced Research in Computer and Communication Engineering,* January.2(1).

Kumar1, N. et al., n.d. Leafsnap: A Computer Vision System for Automatic Plant Species Identification.

Kumar, N. et al., 2012. Leafsnap: A computer vision system for automatic plant species identification. In: *In Computer Vision ECCV.* s.l.:Springer, pp. 502-516.

Larese, R. M. et al., 2014. Multiscale recognition of legume varieties based on leaf images. *Expert Systems with Applications,* 41(10), pp. 4638-4647.

Lee, C.-L. & Chen, S.-Y., 2006. Classification of leaf images. *International Journal of Imaging Systems and Technology,* 16(1), pp. 15-23.

Lee, C. & Wang, S., 2001 . Fingerprint feature reduction by principal Gabor basis function. *Pattern Recognition,* Volume 34, pp. 2245-2248.

Linnaeus, C., 1967. *Mantissa Plantarum. Generum Editionis vi et Specierum Editionis ii.* 2 ed. s.l.:Laurentius Salvius.

Liu, N. & Kan, J.-m., 2016. Improved deep belief networks and multifeature fusion for leaf identification. *Neurocomputing, 2016. 3, 10, 17,* Volume 3, pp. 10, 17.

Li, X. & Wu, X., 2014. Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition. *arXiv,* 15 10.pp. 1410-4281 .

Loffe, S., 2006. Probabilistic linear discriminant analysis. *European Conference on Computer Vision,* Volume 4, pp. 531-542.

MathWorks, n.d. *Introducing Deep Learning With MATLAB.* s.l.:MathWorks.

Mezzo, D. B. R. G. F. C., 2003. Weed leaf recognition in complex natural scenes by model-guided edge pairing. *Precision agriculture,* 5(67), pp. 141-147.

Muskopf, S., 2018. *Bilology Corner.* [Online]
Available at: https://www.biologycorner.com/worksheets/leaf_coloring.html
[Accessed 14 01 2018].

Nilsback , M. & Zisserman, A., 2006. A visual vocabulary for flower classification.. *2006 IEEE computer society conference on computer vision and pattern recognition,,* Volume 2, p. 1447–1454.

Peura, M. & Iivarinen, J., 1997. *Aspects of Visual Form,* Volume 27, pp. 443-452.

Pratt, 2007. Digital image processing. *Wiley-Interscience,* Volume 24, p. 24.

Reed , T. R. & Dubuf, J. M., 1993. A review of recent texture segmentation and feature extraction techniques. *CVGIP: Image Understanding,* 57(3), p. 359–372.

Rosenblatt, F. X., 1961. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Washington DC: Spartan Books.

Rumelhart, D., Hinton, G. & Williams, R., 1986a. *Learning representations by back-propagating errors.* s.l.:Nature.

Sak, H., Senior, A. & Beaufays, F., 2014. "Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling.

Schmidhuber, J., 2015. Deep Learning in neural networks: an overview. *Neural Networks,* Volume 61, p. 85–117.

Seeland, M. et al., 2016. Description of flower colors for image based plant species classification.. *Proceedings of the 22nd German Color Workshop (FWS) Zentrum für Bild- und Signalverarbeitung,* p. 145–154.

Sladojevic, S. et al., 2016. Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification. *Computational Intelligence and Neuroscience* , 2016(3289801), p. 11.

Sonka, M. & Hlavac, V., 1993. Roger boyle `image processing. *Analysis, and Machine Vision',* Issue 28, 31, 35, p. 10.

Sponton, H. & Cardelino, J., 2015. A review of classic edge detectors.. *Image processing online,* 5(25), pp. 90-123.

Stanford, 2017. *CS231n Convolutional Neural Networks for.* [Online]
Available at: http://cs231n.github.io/convolutional-networks/
[Accessed 06 12 2017].

Steinwart , I. & Christmann, A., 2008. *Support Vector Machines.* New York: Springer Science & BusinessMedia, .

Stricker , M. & Orengo, M., 1995. *Similarity of color images.* s.l., s.n., pp. 381-392.

Theano Development Team, 2015. *Deep Learning Tutorial.* Release 0.1 ed. LISA Lab: University of Montreal.

Vogel, N., 2018. *ThoughtCo.* [Online]
Available at: https://www.thoughtco.com/plant-leaves-and-leaf-anatomy-373618
[Accessed 14 01 2018].

Wäldchen, J. & Mäder, P., 2016. Plant Species Identification Using Computer Vision Techniques: A Systematic Literature Review. *Arch Computat Methods Eng,* 11.Issue DOI 10.1007/s11831-016-9206-z.

Wang, Z., Lu, B., Chi , Z. & Feng, D., 2011. Leaf image classification with shape context and sift descriptors. *2011 International conference on digital image computing techniques and applicaTIONS (DICTA),* pp. 650-654.

Wechsler , H., 1980. *Texture analysis: a survey. Signal Process.* 3 ed. s.l.:s.n.

Wilkin, P., Clark, J. Y., Corney, D. & Cope, , J. . S., 2012. Plant species identication using digital morphometrics: A review. *Expert Systems with Applications,* 39(8)(7562-7573).

Xiaoyi Song, X. & Li, Y., 2008. *A Textural Feature Based Image Retrieval Algorithm.* s.l., s.n.

Yanikoglu , B., Aptoula , E. & Tirkaz, C., 2014. Automatic plant identification from photographs. *Mach Vis Appl,* 25(6), p. 1369–1383.

Yuan, T., 2009. Multiple Classifier Combination For Recognition Of Wheat Leaf Diseases. *Intelligent Automation and Soft Computing,* 15(X), pp. 1-10.

Zhang, D., Islam, M. & Lu, G., 2012. A review on automatic image annotation techniques. *Pattern Recognition*, 45(1), p. 346–362.

Zhou , Z. H. & Chen, S. F., 2002. Neural network ensemble. *Chinese Journal of Computers*, 25(1), pp. 1-8.