UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Classification of leaf diseases in plants applying Deep Learning Techniques

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:

Giovanny Eduardo Caluña Chicaiza

Tutor:

Lorena de los Angeles Guachi, PhD

Urcuquí, marzo de 2020

Urcuquí, 20 de marzo de 2020

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2020-00017-AD

A los 20 días del mes de marzo de 2020, a las 14:00 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Miembro No Tutor	Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.

El(la) señor(ita) estudiante **CALUÑA CHICAIZA, GIOVANNY EDUARDO**, con cédula de identidad No. **1725053910**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **Classification of leaf diseases in plants applying Deep Learning Techniques**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN.**

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Miembro Tribunal De Defensa	Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.	9,5
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.	10,0
Presidente Tribunal De Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.	9,5

Lo que da un promedio de: **9.7 (Nueve punto Siete)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

CALUÑA CHICAIZA, GIOVANNY EDUARDO
Estudiante

Firmado electrónicamente por:
FREDY ENRIQUE
CUENCA LUCERO

Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Presidente Tribunal de Defensa

Firmado Digitalmente por: LORENA DE LOS ANGELES GUACHI GUACHI
Hora oficial Ecuador: 12/06/2020 16:42
Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.
Tutor

Firmado electrónicamente por:
JULIO JOAQUIN
ARMAS
ARCINIEGA

Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.
Miembro No Tutor

TORRES MONTALVÁN, TATIANA BEATRIZ
Secretario Ad-hoc

Autoría

Yo, **Giovanny Eduardo Caluña Chicaiza**, con cédula de identidad 1725053910, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor(a) del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, marzo 2020.

giovanny

Giovanny Eduardo Caluña Chicaiza
CI: 1725053910

Autorización de publicación

Yo, **Giovanny Eduardo Caluña Chicaiza**, con cédula de identidad 1725053910, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, marzo 2020.

[illegible]

Giovanny Eduardo Caluña Chicaiza

CI: 1725053910

Dedication

"For my parents, for the huge effort made to provide me all the necessary resources to complete my career. Also for the love and the unconditional support during every single day of my university stage. For my siblings who were always encouraging me to never give up. Last but not least, for Gabriela who was with me in my best and worst moments."

Giovanny Eduardo Caluña Chicaiza

Acknowledgements

"Foremost, I would like to express my sincere gratitude to my advisor Prof. Lorena de los Angeles Guachi for the continuous support during the elaboration of my thesis, for her patience, dedication, advice, suggestions, motivation, and immense knowledge. Her guidance helped me in all the time during the development of this thesis. Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Fredy Cuenca and Prof. Julio Armas, for their valuable comments and suggestions. My sincere thanks also go to Eng. Fabian Jimenez, for accepting me as an intern and transmitting me all the knowledge about super computing, which was of great importance during the elaboration of this thesis."

Giovanny Eduardo Caluña Chicaiza

Resumen

Una de las principales preocupaciones de los agricultores es la detección temprana de enfermedades en sus cultivos. Sin embargo, supervisar los cultivos extendidos manualmente por los agricultores es una tarea extensa y tediosa. En relación con este objetivo y para proporcionar una solución automática y eficiente, este trabajo se centra en el análisis y la búsqueda de un método óptimo para la creación de un sistema automático informático que sea capaz de detectar plagas y enfermedades en las hojas de las plantas mediante la clasificación automática de imágenes. En este trabajo, se realizó una revisión del estado del arte de las técnicas para la clasificación automática de enfermedades e imágenes de plantas en general. Después de la revisión, se eligió el método más óptimo, el Aprendizaje Profundo (AP) y específicamente las Redes Neuronales Convolucionales (RNC). Actualmente hay muchas arquitecturas de RNC y sus variaciones disponibles, por lo que las más relevantes se seleccionaron después de un estudio de comparación realizado entre los modelos de RNC más destacados en la literatura. Por lo tanto, se obtuvieron cinco redes que destacan por su precisión, costo computacional, número de capas y parámetros. Son Inception 3, GoogLeNet, ZFNet, ResNet 50 y ResNet 101. Las arquitecturas seleccionadas fueron entrenadas y probadas con un conjunto de datos de 13k imágenes de hojas sanas y no saludables obtenidas de Plant Village [1]. Para probar el comportamiento y el rendimiento de los modelos en un entorno realista y, además, para ayudar al entrenamiento y la precisión, se realizaron diferentes experimentos en el conjunto de datos, así como en los hiperparámetros de los modelos. Una vez que se llevaron a cabo los experimentos y se analizaron y compararon los resultados, se obtuvo la RNC ZFnet como la opción más apropiada debido a su eficiencia en términos de costo computacional y nivel de precisión. ZFnet alcanzó una precisión del 91 % con los datos sin procesar y del 93 % después del preprocesamiento de datos y el ajuste. En el tiempo de entrenamiento y despliegue, ZFnet pudo realizar 10 iteraciones de entrenamiento en 2 segundos y desplegar 10 imágenes al mismo tiempo muy por debajo de los otros modelos evaluados en este trabajo.

Palabras clave: Aprendizaje profundo (AP), redes neuronales convolucionales (CNN), clasificación de imágenes, implementación, precisión, precisión, hiperparámetros, datos en bruto, arquitectura.

Abstract

One of the main concerns for farmers is the early detection of diseases in their crops. However, supervising extend crops manually by the farmers is an tedious and time-consuming task. In connection with this aim and to provide an automatic an efficient solution, this work focuses on the analysis and search for a most appropriated method for the creation of an autonomous system that is capable of detecting pests and diseases in the leaves of plants through automatic image classification. In this work, a review of the state-of-the-art of techniques for the automatic classification of plant diseases and images in general was carried out. After reviewing the state-of-the-art, five outstanding Deep Learning and specifically the Convolutional Neural Networks (CNNs), was chosen. They stand out for their accuracy, precision, computational cost, number of layers and parameters. They are Inception 3, GoogLeNet, ZFNet, ResNet 50 & and ResNet 101. The selected architectures were trained and tested with a data set of 13k images of healthy and unhealthy leaves obtained from Plant Village [1]. To test the behaviour and the performance of the models against a realistic environment and, in addition, to help the training and accuracy, different experiments were performed on the data set as well as on the hyper-parameters of the models. Once the experiments were carried out and the results were analyzed and compared, the CNN ZFnet was obtained as the most appropriate option due to its efficiency in terms as computational cost and level of accuracy. ZFnet reached an accuracy and precision of 91% with the raw data and 93% after the data pre-procesing and the fine tune. In the training and deploying time, ZFnet was able to perform 10 train iterations in 2 seconds and deploy 10 images in the same time well below the other evaluated models on this work.

Keywords: Deep Learning (DL), Convolutionary Neural Networks (CNN), image classification, deploying, accuracy, precision, hyper-parameters, raw data, architecture.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the major issues that affects farmers around the world are diseases in plants produced by different pests, viruses, and insects, thereby has generated huge losses of money [13]. The best way to relieve and combat this is timely detection. In almost all real solutions, early detection is determined by human, an expensive and inefficient solution. In this sense, detected or classified unhealthy leaves at an early stage can help to mitigate or even avoid the effects produced by those pests.

On the other hand, in computer science, image classification is a computational task to label pixels of a digital image into one of several classes. Although the image classification has been developed since many years ago, there still some challenges because of the large variety of computational techniques developed to classify images, the quantity and the quality of the images used for train and deploy them and the performance reached under different conditions as position, shadows, noise etc.

In the last years, image classification has had a significant boom because the great advance in computer technologies and the large quantity of images storage on the internet [14]. For this reason, image classification has been applied in different fields as in health care, agriculture, sports, security, fire control etc. Specifically in agriculture, image classification has been used to solve different problems such as: to optimize production on plantations by creating models to determine the right treatment plants for different crop types and regions [15], identify missing vegetation from a crop field [16], detect locations of weeds in images from cereal fields[17], detection of unhealthy regions on leaves plants and their classification [18]. The main techniques used to perform these classification task are Machine Learning and most recently Deep Learning [19].

The present work aims at tackling the unhealthy leaves from an automatic approach. For this purpose, an state-of-the-art review from image classification based on deep learning techniques, is performed to obtain the most outstanding methods. The main measures to be taken into account are accuracy, precision, training and deploying time. Then, with the appropriate hardware and software tools, the selected five deep learning models are implemented. To measure and compare the performance, the models are trained under the same conditions divided in three different experiments called: raw data, pre-procesing data, and the fine tune. Definitely, as final result, it obtains a single model with the best performance and therefore the optimal method to create an automatic leaf classifier computational system.

1.1 Problem Statement

Pests, plant diseases and herbs can be a serious threat to crops. According to [20] to avoid these kinds of problems a regular inspection should be performed. However, controlling all the plants day by day its an expensive, time-comsuming and tedious task. On another hand, because of the high performance and great advantage presented by an automatic image classification as is showed in [21], [22], [23], [24], [25], [26], [27], different algorithms have been developed applying well know techniques such as Neural Networks, Support Vector Machine (SVM), and Convolutional Neural Networks (CNNs) in Artificial Neural Networks (ANN), Machine Learning (ML) and Deep Learning (DL), respectively.

The main problem of existing techniques, and particularly DL ones, is that they have been developed according to general purposes as in the ILSVRC challenge [28], which is aimed at classifying a large number of images in one thousand of classes. In this sense, the models must be tested and fine-tuned with the available data and computational resources in order to implement such techniques in a specific task as unhealthy leaves classification.

In DL domain, Convolutional Neural Networks (CNNs) have in fact demonstrated great performance in the image classification field over well-known algorithms such as SVM and ANN. Nevertheless, currently there exist a lot of CNNs models available. Therefore, this work aims to explore the most outstanding CNN models by applying a specific data set of healthy and unhealthy leaves available in Plant Village [1]. To increase the performance achieved by each model, some variations and pre-processing operations focused on rotations, addition of random classes, and contrast improvement are applied to the data.

1.2 Scope of the Project

Automatic systems provide faster solutions than human inspection to identify unhealthy plant leaves. However, their execution in real scenarios depends on the percentage of reached accuracy, computational requirements, and ease for implementation. This project establishes characteristic metrics to select and compare outstanding DL methods applied to image classification. Moreover, it concerns of selecting the most appropriate method to classify leave images as "healthy" or "unhealthy".

In addition, the developed methodology allows to know the open research challenges on the pre-processing data and classification applied to automatic leave diseases classification. All the selected methods are implemented and trained following different conditions and varying their hyper-parameters. Besides, this work also implements an interactive user interface using the trained models for leaf classification. It shows the prediction performed by each model and the probabilities for each class (healthy and unhealthy). Finally, obtained results provides a point-of-view to choose the method suitable to the implementation expectations of accuracy, time and computational requirements.

1.3 Thesis overview

The rest of this work is organized as following: Chapter 3 has some important concepts and theory related with image classification. It also has the image classification definition its current applications and challenges, and a review from DL techniques applied to image classification. For DL techniques, particularly CNNs, there is a detailed description of its architecture, followed by a description of their different layers and phases. This chapter ends with a description and comparison of the main frameworks to implement and train CNNs. Chapter 4 contains a detailed explanation of the methodology followed in this work, which includes: the criteria to select the most relevant CNN architectures, a brief description of the selected architectures, the software and hardware resources employed to perform this work. Afterwards, this chapter shows and describes the steps and scripts used to prepare the data set. Finally, Chapter 4 has the a brief description of the files needed to implement and deploy a CNN architecture as well as the hyper-parameters to fine tune the model. Chapter 5 starts with the metrics employed to measure and compare the models. It also has a broad description of all experiments and sub experiments to perform this work. The Chapter 6 contends the detailed results from all the experiments described in the Chapter 5 as well as a comprehensive analysis and a discussion of the obtained results. The results are showed in tables and figures to a better understanding. This works ends with Chapter 7, it contains the main points and issues face during the elaboration of this work, conclusions and some recommendations.

Chapter 2

Objectives

2.1 General Objective

- To explore Deep Learning-based image classification methods for selecting the appropriate method to automatic detection of healthy and unhealthy leaves.

2.2 Specific Objectives

- To select the most outstanding Deep Learning techniques for image classification based on metrics as: accuracy, computational complexity, computational cost and training time.
- To select the appropriate software and hardware for training and validation purposes of the evaluated techniques.
- To implement, test and analyze the selected techniques in order to determine their ability to correctly identify a leaf as unhealthy or healthy, considering metrics such as: accuracy, recall, precision, F1 score, training (foreward-backward) and execution time.

Chapter 3

Theoretical Framework of Deep Learning for Image Classification

This chapter gives an overview theoretical foundations about image classification. In the context of computational science, the following questions are answered:

- What is image classification?
- What are the fields application of image classification?
- What are the challenges presented on image classification?
- What are the developed techniques to image classification?
- What is the software available to implement CNNs?

To answer the mentioned questions, in this chapter the general image classification approach is presented, starting with the definition of several important concepts, applications and challenges. Then, a brief state of the art of different image classification techniques are presented. Next, the theoretical foundation of a CNN is stated describing the layers and stages. Finally, the available software tools for CNN training purposes are introduced.

3.1 Concepts

- **Deep Learning (DL)**

  It relates to the field of ML-based algorithms, focusing on how neurons in the brain work and applying a similar analogy to help machines learn by the use of Artificial Neural Networks [29].

- **Convolutional Neural Network (CNN)**

  In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery [30].

- **Max pooling**

  It is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned [30].

- **Feature Map**

  A feature map is a matrix that contains the information of the image. If the image has three channels (like a RGB image) then there is a feature map for each of the three channels [31].

- **Stride**

  It is a parameter in a CNN which controls how the filter moves around the input map [31].

- **Overfitting**

  Overfitting occurs when a network does a pretty good job learning data on the training dataset but has a bad performance when using a different dataset.

3.2 Image classification

In computer science, the image classification process is a computational task to label all pixels of a digital image into one of several classes as in Fig. 3.1 where the input are two images from the class cat and dog respectively. They pass through a classifier and they are labeled with their respective class. There are two types of image classification tasks which are differentiated depending on the knowledge about the class of the dataset as follows:

1. **Unsupervised classification:** It is performed when the outcome is based just in the computational analysis of an input image without any provided information about the classes. The user just can determine which technique will use and the desired number of output classes.

2. **Supervised classification:** It is performed when a computational model is trained using knowledge about class provided by the user. It is done usually labelling the data (images) with the class to train the model. Also, the user has to give the number of classes that an input image is classified into.

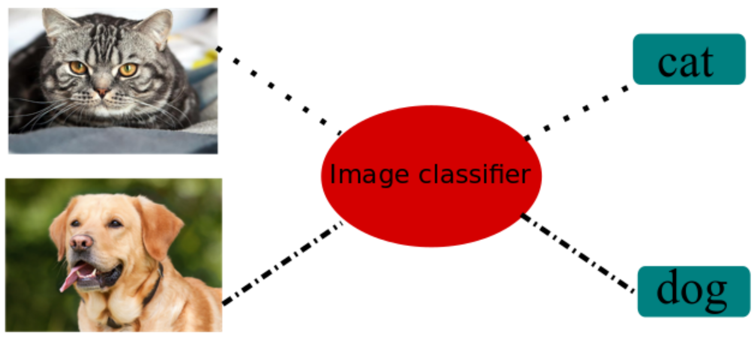


Figure 3.1: Classification of images

The image classification can be performed applying statistical decision rules in the multispectral domain or logical decision rules in the spatial domain [32]. Multispectral domain considers just the spectral information provided by the channels or image bands. On the another hand, spatial domains uses their features as size, geometric shape, texture, patterns of pixels or objects derived from analysis of close objects. The most common techniques used to classify images such as ANN, ML classifiers and CNN use the spatial domain to classify the images. In the following sections these techniques are explained as well as their advantages and disadvantages.

3.2.1 Applications

Currently the image classification often also called image identification has already been embedded in many fields as:

1. **Health care:** In health field, image classification is well suited to analyze x-ray or magnetic resonance imaging results to identify diseases and assist to human clinicians in the medical diagnosis such as melanoma [33] and breast cancer [34] identification.

2. **Agriculture:** In agriculture field, image classification has been used to solve different problems such as: to optimize production on plantations by creating models to determine the right treatment plans for different crop types and regions [15], identify missing vegetation from a crop field [16], detect locations of weeds in images from cereal fields[17], classify health and unhealthy leafs plants [18] etc.

3. **Sports:** In the sports field, image classification has been used to: effectively handle the huge sports video repositories in tasks as video summarizing, key-events selection, and to suppress the misclassification rates [35], predict goal-scoring opportunities in soccer finding goal-scoring opportunities and ball possession [36], study the effectiveness of a team's set analyzed three seasons of English Premier League data pieces[37], among others.

4. **Security:** In the field of private security, image classification helps to identify security breaches of private locations extracting complex features from the images

as faces or bodies in order to increase the security and avoid fake alarms [38]. Also, systems have been created to detect thefts in live video and alert to the owner [39].

5. **Documents Processing:** Image classification allows to analyze documents performing a character and word recognition [40], [41] and constructs the representation of a text [42] etc.

6. **Face Recognition:** Image classification is applied to identify and recognize faces [43]. The faces can be classified by their expression happy, sad, angry etc. Also, they can be classified analyzing their physical features as: gender, age [44] etc. In social networks face recognition is used for automated image organization of large data bases and visual websites [45].

7. **Fire Control:** Image classification allows to minimize the consequences caused by fire. There are systems which differentiate scenes with fire and normal scenes from images taken by cameras for instance: a security camera, camera on a drone etc. It can be used to detect fire in an early stage to help the control entities as firefighters [46].

There are many future applications to image classification as creating city guides, powering self-driving cars, boosting augmented reality applications and gaming, organizing one's visual memory, teaching machines to see, empowering educators and students, improving iris recognition and much more [45].

3.2.2 Challenges

Image classification should handle the following challenges under real-life environments.

1. **Digital image noise:** In digital images, noise refers to the visual distortion or undesirable information added to images due to bad conditions during: acquisition, digitization or transferring of the image. The noise can be seen as random speckles on a smooth surface. Noise plays an important role on image classification because an image can be misapprehend and then miss-classified due to its presence. For

example in [47], the authors said that the noise and shadows are the main factors classification errors.

2. **Rotations:** An image classifier can be tricked performed rotations on the input images. If the image classifier is not trained with enough different rotations of an image it might classify images that are entirely equal as different classes.

3. **Relative location:** Usually, an image classifier only take care for elements in the image and does not for their location [2]. For example in 3.2, an image classifier can classify the image in the same class because both have the same elements as eyes, mouth and nose.

4. **Dataset size:** Image classifiers need a lot of data to train and perform a right image classification [48]. So get the enough amount of data is a important issue in the image classification.

5. **Computational resources:** It is necessary a strong computational resource to compute the huge among of data, currently the best option are GPUs [14].

6. **Shadows:** They are a variation of the light intensity on an scene produced by an object which is interposed between the scene and the light source [49].

7. **Illumination Changes:** Commonly, sudden changes such as light on/off happen often in indoor environments, while during the day, outdoor environments often can experience gradual changes.

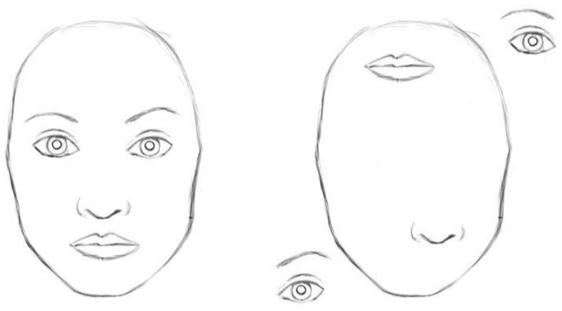


Figure 3.2: Representation of a relative location challenge [2]

3.3 Deep Learning-based Techniques

DL is a sub field of ML and can be described as a large neural network [50]. DL theoretically appeared in 1980s, but they just recently has become popular. The main reasons were the poor labeled dataset and the computing power for the time [51]. DL has been applied in many fields such as automated driving, aerospace and defense, medical research, industrial automation, electronics and more [51]. Those techniques are called 'Deep' because they are composed of many layers or also called hidden layers instead 2 or 3 as in conventional ANN, as is shown in Fig. 3.3. A particular and very effective technique of DL to classify images are the Convolutional Neural Networks (CNNs). They are deep learning models introduced to persistently break down information with a homogeneous structure (filters) [29]. CNNs are a set of multiple layers with different convolutional filters of one or more dimensions to identify an input image based on its features. This is accomplished for the building of a complete feature extraction model which is capable of handling difficulties of conventional methods (ML and ANN) such as: improve the accuracy, work with very large amount of data and leave the dependence of domain expertise [52]. Therefore, this work is based on CNNs architectures, so they are explained in detail in the next section. DL models has many benefits, but they also has some limitations as follows.

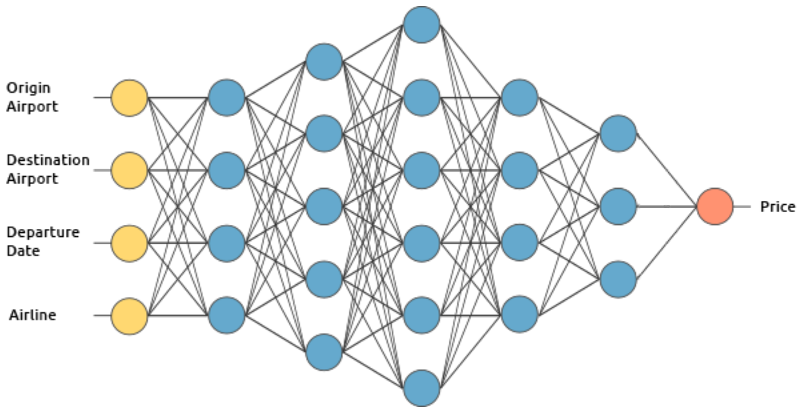


Figure 3.3: A DL model example [3]. Depicted model shows a DL model to predict ticket prices where the inputs are: the origin airport, destination airport, date an airline followed by 5 hidden layers and one output which represents the price predicted by the model.

- **Advantages:**
  - **Automatic Feature extraction:** In the case of CNN models, they are able to extract all the features from a dataset from images and train the model by themselves. The model mainly needs to know what class does each image belongs from the train dataset. This ability is the best advantage against ML models [29].
  - **Adaptivity:** DL models have several architectures easily adaptable to new problems [2].
  - **High Accuracy:** A CNN with a huge train dataset can reach the highest level of accuracy (with respect to traditional AI techniques) which even overcomes the human capabilities.
- **Disadvantages:**
  - **Require high processing power:** All DL models are composed by a lot of layers where the data will be processed and to train the models usually it is necessary to use robust machines with powerfull Graphical Processing Units (GPUs) [53].
  - **Huge size of the dataset:** As in machine learning to train a DL model with a high level of accuracy, the model has to be trained with a large dataset [54].

In the field of image classification, particularly, CNN has become the most popular and used technique because their high accuracy and the availability of different architectures. CNNs have been applied in different fields such as health, to classify anomalies as breast cancer in women [34], melanomas on the skin [55], [56], interstitial lung diseases [54] performing the task with high accuracy even than a clinician. CNNs also have been applied in the agriculture field to classify, specially, leaf diseases in different kind plants [26], [57],[58]. CNNs have been implemented to classify hyper spectral images [59], [60]. In all cases, CNNs classifiers reached accuracies over 95% and have demonstrated better performance than ML techniques.

3.4 Convolutional Neural Networks (CNN): Theoretical Foundation

In the last years DL techniques have had a great reception as is clearly showed in the Fig.3.4. One of the most relevant is the CNN.

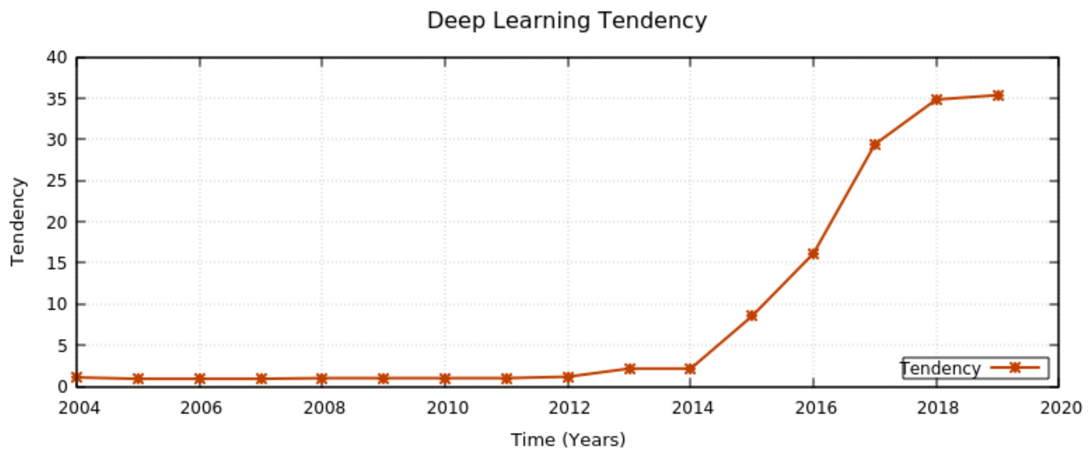


Figure 3.4: Tendency in the use of Deep Learning Techniques, data from Google analytic

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other through the convolution operations with filters. CNNs are widely applied in the field of computer vision because of the great advantages against traditional methods such as ANN and ML. The main advantages against traditional methods are: fewer parameters, distortion tolerance, automatic feature extraction, work with large amount of data and high accuracy. By the advantages mentioned before, they are particularity appropriate to applications in which the data can be represented as a matrix. For instance, an ANNs, working with a color image of 480x480 pixels, will generate 691 200 (=480x480x3) entry weights in the first layer, which will be very complex to manage and compute. A CNN can decrease notably that number because it works with set of pixels instead of individual ones. Their most common applications are: decoding facial recognition, analyzing documents, understanding climate, advertising, among others [61].

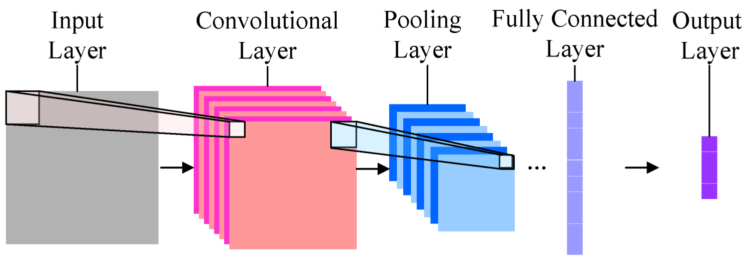


Figure 3.5: CNN Basic Architecture. Obtained from [4].

A basic CNN has three types of layers: convolutional layer, pooling layer and a Fully Connected (FC) layer. The Fig. 3.5 shows a basic CNN architecture adapted to image classification. The CNN input is an image of MxMxR, where M is the height and width of the image and R is the number of channels [31]. The input often is followed by a repeated sequence of convolutional, pooling and fully connected layers. Fig. 3.5 shows the ouput layer, which is modified depending on the application. For example: the outputs on image classifications mean the probability to belong to some class and on object detection they represent the position, dimensions and the class of a particular object.

3.4.1 CNN Layers

As it was mentioned already, a CNNs are a special kind of multi-layer neural networks, specially designed to recognize features directly from images. The architecture of a CNN defines how the different layers will be set up to an specific model. Also, through the time scientists have developed new kind layers which have been added in their own models. However, the most common architectures are formed by:

- **Convolutional layer:** This layer is the most important in a CNN, in fact, their name comes from it. It consists of a set of filters to extract different features from the input image. In mathematical terms convolution is defined by:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m,j-n) \tag{3.1}$$

  where I represents the image and K the filter also often called kernel as is shown in Fig. 3.6. The subscripts m and n represent the dimensions of the filter typically of 3x3, 5x5, 7x7 or 11x11. As it was mentioned the input (image) is composed by ixjxr where r is the number of channels of the image, for instance in a color image (RGB) the r will be 3. The filters are moved from the left up corner to the right down corner applying the convolutional operation to create a featured map.

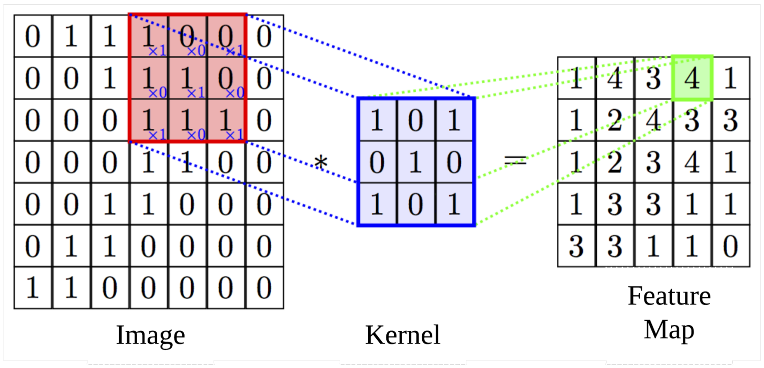


Figure 3.6: Convolution representation

The filter is moved depending on the assigned stride. The stride is the number of pixels that the filter will move (one pixel in Fig. 3.6) to apply the next convolution. The dimensions of the obtained feature map from a convolution can be calculated with:

$$O = \frac{(W - K + 2P)}{S} + 1 = \frac{(7 - 3 + 2.0)}{1} + 1 = 5 \tag{3.2}$$

Where O is the dimensions (weight & length), W is the dimension of the input (image), K is the filter size, S is the stride and P is the padding. The pad is the number of columns and rows that are added to the input image in order to apply the filter in the entire image and do not lose information because the size of the filter.

Each convolutional layer uses a set of filters (not just one). The main purpose of each filter is detect simple structures as borders, lines, squares, etc. and grouping them to construct more complex shapes.

- **Activation functions:** In the convolution process many negative values are generated. These values are unuseful for the next layers and produce more computational load. Therefore, after a convolutional layer, an activation function is applied. The activation function sets to zero the negative values which helps to get a faster and effective training. There are many functions used in DL, but the most common are:

1. **Sigmoid** [62]**:** This kind of functions are commonly used in ML for the logistic regression.

$$f(x) = sigmoid(x) = \frac{1}{1+e^{-x}} \tag{3.3}$$

2. **ReLU** [62]**:** A Rectified Linear Unit set the value to 0 if the input is less than 0, and raw otherwise. This function is the most used in the CNN and ANN because its behaviour is closer to the biological neurons.

$$f(x) = max(x, 0) \tag{3.4}$$

3. **Softmax** [62]**:** It is an activation function based on probability distribution. This function set the values from 0 to 1 and the total sum always gives 1.

$$\sigma(Z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} for j = 1, ..., k \tag{3.5}$$

   where z is the input vector, j indexes the outputs

- **Polling layers:** It aims to reduce progressively the parameters sub sampling the spatial size of its input in order to reduce the computational load. The polling operation is a sliding operation similar to convolutional one. It has a kernel and a stride as in Fig 3.7, but in this case the main features are selected and preserved according the kind of polling applied. There are three different types of sub sampling where the operation to select the pixel is the only difference between them. They are the following:

1. **Average-polling:** It selects the average value of all the pixels from the batch.
2. **Minimum-polling:** It selects the minimum pixel value from the batch.

3. **Maximum-polling:** It is the most used in CNN [63]. The maximum pixel value of the batch is selected as is shown in Fig.3.7.

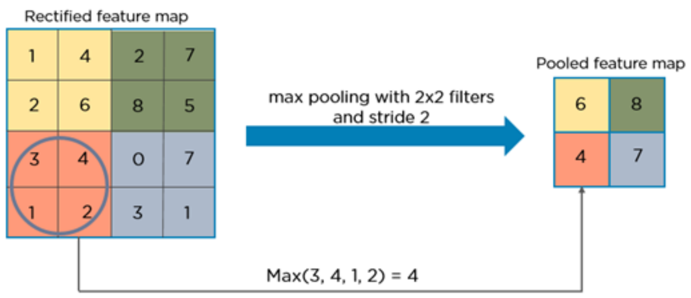


Figure 3.7: Max Pooling Representation [5]. The polling is performed in a kernel of 2x2 and it is moving following a stride of 2

- **Fully Connected layer (FC):** It aims to take the values from the previous layer and order them in a single vector of probabilities to dropout the class which the input belongs to. For instance, in Fig. 3.8, the input is a bird, so after the convolutions, some of the values in the input of the fully connected layer should have high probabilities for the class bird and low for the rest [64].

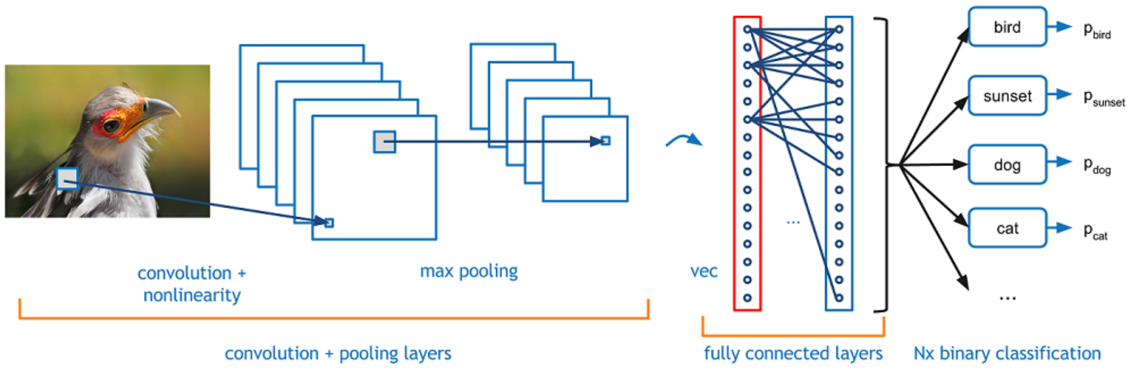


Figure 3.8: Traditional CNN architecture [6].

- **Inception Module:** Through the deployment of CCNs, researchers realized that by increasing the convolutional layers, accuracy of the CNN increases too. However, a conventional convolutional layer already needed high computational resources, so deeper models brought a computational cost problem. To mitigate that problem, in [7] authors present the module Inception. It is a novelty method to apply the

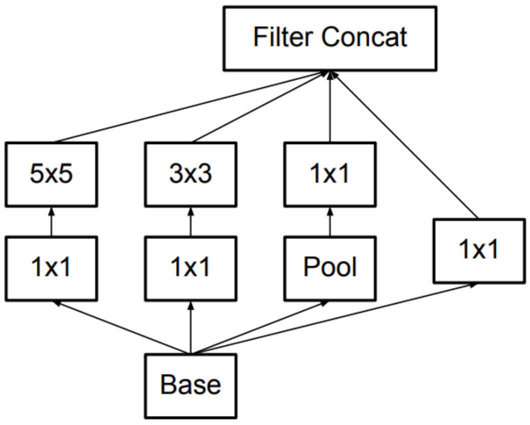


Figure 3.9: A Inception module as is described in [7].

convolution operation using different size filters (see Fig. 3.9) in order to extract the most different and relevant parameters to the training. The different map features obtained are concatenated before the next layer. A significant change over this kind of modules was proposed by [8]. They applied a factorization operation over the inception modules by using large filters and dividing into two or more convolutions with smaller filters (see Fig. 3.10). This structure aims to reduce the number of parameters and keep the main features. Factorized inception module produced x12 lesser parameters than AlexNet [65] (one of the most common implemented architecture). For instance: a convolutional filter of 5x5 is replaced by two convolutional filters of 3x3 (see 3.9 & 3.10) to produce a computational gain of 28%.

- **Residual Module:** Another important problem present to training a CNN, for shallow and deep networks, is the vanish gradient [66]. By this reason, in [9], the authors proposed reformulate the layers as learning residual functions (see Fig. 3.11):

$$H(X) = F(X) - X \tag{3.6}$$

This reformulation was done based on that a shallow network has less training error than the deeper counterpart. They proposed to construct the added layers as

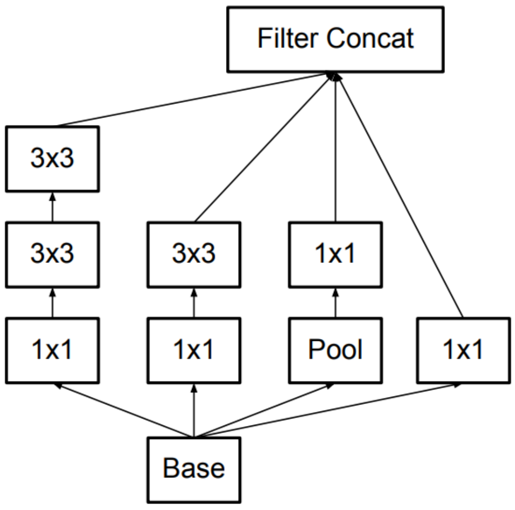


Figure 3.10: A factorized Inception module as is described in [8].

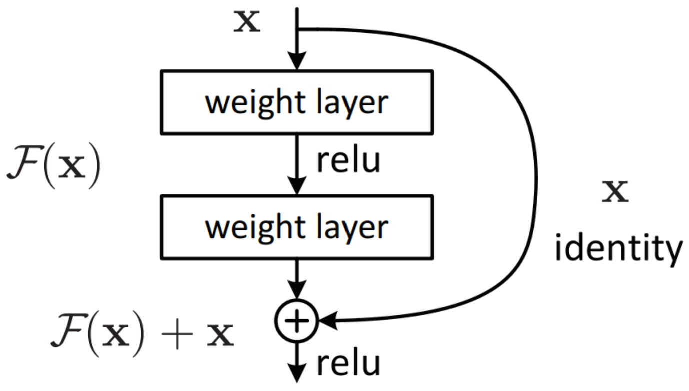


Figure 3.11: Residual module [9].

identity maps (X). So, the residual modules helped to the deeper networks have an error no greater than the shallow ones.

3.4.2 CNN Common Problems with CNNs

During the development of the CNNs there have been many problems. The most common problems are: Over fitting, computational expense and the vanish gradient problem.

- **Computational Expensive:** Through the CNNs evolution, they became deeper models thus they generate significantly more parameters and operations. Therefore, to train and use deeper models usually is necessary high computational resources.

- **Over fitting:** This is the most common problem on ML models. It is produced when models have problems to generalize patterns and instead the model recognizes specific inputs.

- **Vanish Gradient Problem:**This is a particular problem presented on CNNs where they applies gradient based methods (e.g. Back Propagation) to train the network and the gradient of the loss function tends to zero making hard to train the network.

3.4.3 CNN Phases

The CNN implementation in real scenarios often has training and deployment phases. Each one with particular purposes and tasks as follows:

1. **Training stage:** The CNN training usually performs four tasks: normalization, feature extraction, classification, and validation, as is illustrated in Fig. 3.12.

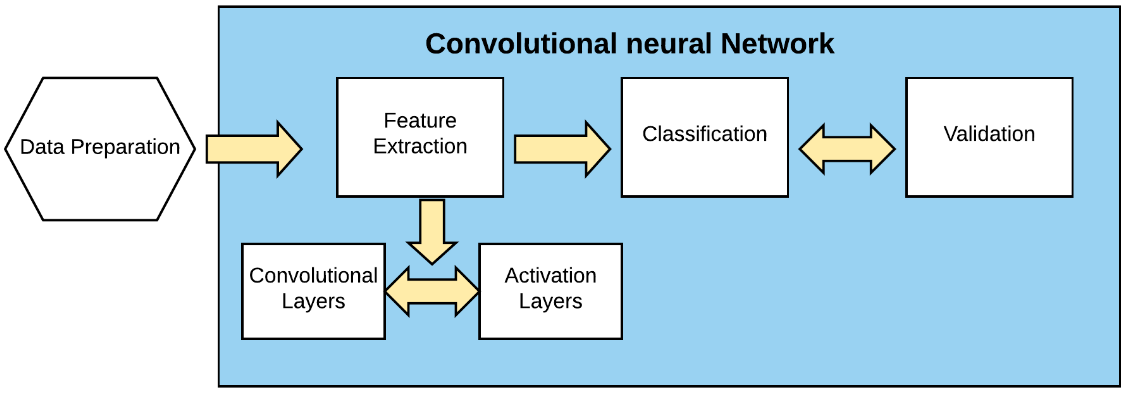


Figure 3.12: CNN process stages

(a) **Data Preparation:** It is the process to normalize the input of the CNN. In other words, the images are modified in order to attempt to the CNN architecture. For example, in the AlexNet architecture [67], the input must have a dimensions of 227x227x3 where, as it was mentioned before, are: height, width and the number of channels of the input images respectively. Even in some architectures, It is necessary or at least recommended perform a pre-processing of the images [58], in order to obtain a better feature extraction and accuracy.

(b) **Feature Extraction:** It is the process performed by the convolutional layers and the activation functions were the most relevant features are extracted from the input data.

(c) **Classification:** It is the process to alter the weights from the fully connected layers applying the data obtained from the convolutional layers.

(d) **Validation:** Once some train iterations are performed, a validation is performed. This process uses the validation set to check the accuracy reached by the model and applies the back propagation.

2. **Deploy:** Once the model is trained, the predictions can be obtained providing an input image that will pass through the stages depicted in Fig. 3.13.

   - **Data preparation:** Usually it is necessary perform a resize in the input image in order to attempt to the CNN architecture.

- **Feature Extraction:** It applies the convolution operations and the activation functions to extract the features from the input.
- **Classification:** In deployment phase, the network used the learned weights and compare them with the features extracted from the input. Finally the model sets the obtained probabilities for all the classes.

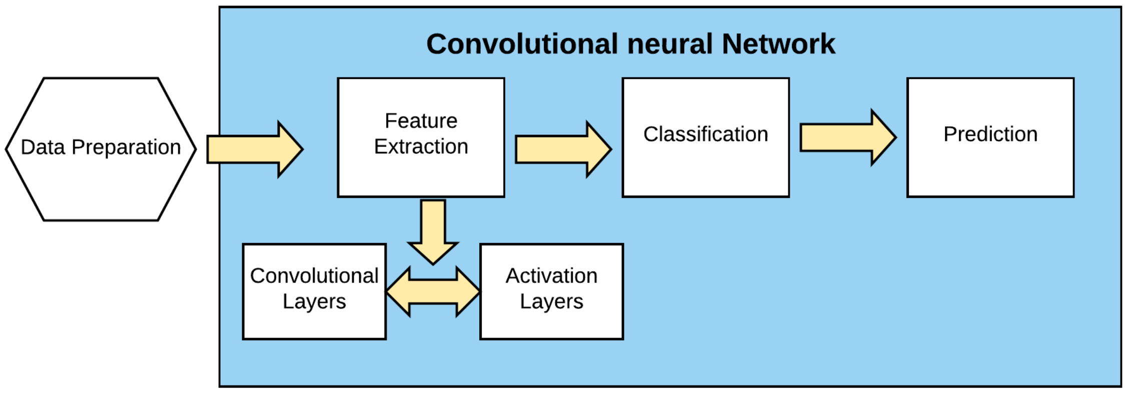


Figure 3.13: CNN Deploy stages

3.5 CNN Learning Frameworks

Some frameworks have been developed to facilitate the training and implementation of different DL algorithms and specially the CNNS. The most popular are: Caffe, Neon, TensorFlow, Theano, and Torch [12]. All of the mentioned frameworks are Open source. Authors in [12] presented a comparative study of them mainly based on the popularity in the community, their properties and the implementation features applied to a particular architecture to measure some parameters, such as gradient and forward time. The most relevant information provided on that work is summarized in Table 3.2 & 3.1.

	Caffe	**Neon**	**TensorFlow**	**Theano**	**Torch**
Core	C++	Python	C++	Python	Lua
Multi-threaded CPU	Blas	Only data loader	XEigen	XBlas conv2D limited OpenMP	XWidely used
GPU	✓	Nvidia backend	✓	✓	✓
Multi-GPU	only data parallel	✓	Most Flexible	Experimental version	✓
Nvidia cuDNN	✓		✓	✓	✓
Quick deploy	Easiest	Supports Op-Tree	✓	flexible	✓

Table 3.1: Properties of the most popular deep learning frameworks [12].

Caffe is one of the first frameworks to implement DL models and it is specialized in tasks for artificial vision as image classification. Therefore, most models have already been implemented in this framework. As showed in Table 3.2, Caffe counts as the large support community in forums and webs. In Table 3.1 are all the main characteristics of Caffe and popular frameworks. Training a DL model could take several hours, days or even months. For this reason, the DL models usually are trained applying parallel computing with GPUs. Caffe is written in C++ and therefore compatible with CUDA, an important parallel computing platform developed by NVIDIA to take advantage of their GPUs, which can be exploited to a faster training. Caffe also provides Python and Matlab interfaces which makes an easier implementation, training and deployment the models. Another fact is that Caffe is already installed in the Supercomputer 'Quinde I'.

Measures	**Caffe**	**Neon**	**TensorFlow**	**Theano**	**Torch**
Number of members in Google groups	4220	73	661	2827	1874
Number of contributors in Github	172	31	81	207	77

Table 3.2: Community involvements for some of the deep learning frameworks as of 02/08/2016 [12].

Chapter 4

Methodology

In this chapter the taken methodology to develop the comparison study among different CNN architectures is described. The following questions are answered over the course of this chapter:

- What were the parameters which were taken into account to select the models?
- What are the main features from the selected models?
- What data set was used for the comparison?
- What are the metrics used to measure the performance of the models?
- Which are the resources (hardware and software) used to perform the study?

In order to perform the comparison study, the process depicted in Fig.4.1 was performed. The first stage consisted on establishing the criteria to select the most outstanding CNN architectures. After, the selection and analysis of CNN models were performed (models selection). Then, the necessary tools of hardware and software are selected for the study (HW and SW Selection). Next, the data set was prepared (data preparation). Then, the models were trained, tuning their hyper-parameters to reach higher accuracy (Training and Tuning). Finally, the models were implemented and tested to obtain their performance with different metrics (deployment).

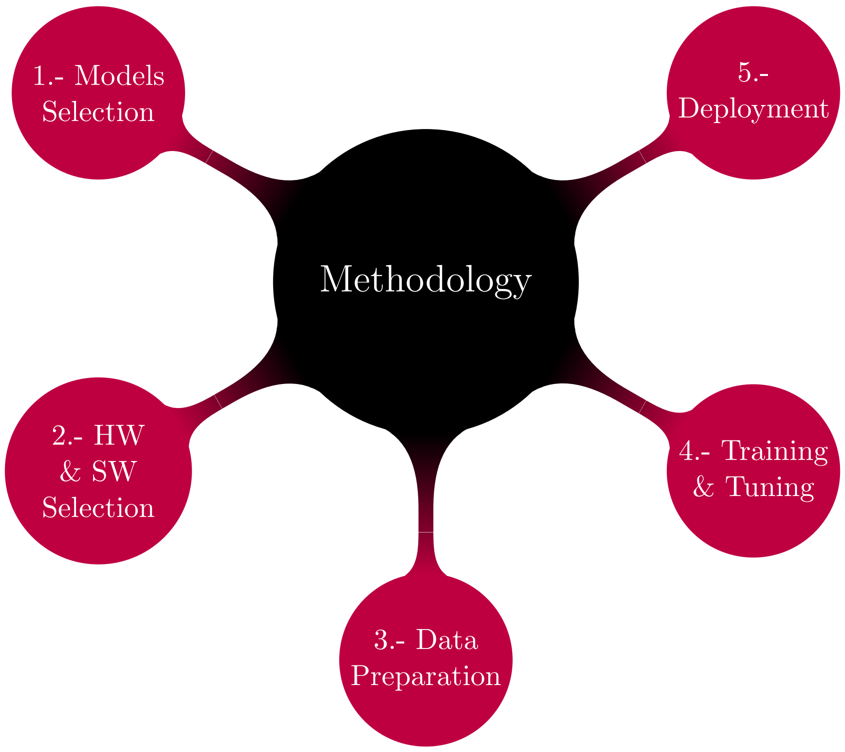


Figure 4.1: Methodology

4.1 Models Selection

For leaf diseases classification purposes, this work aims to identify models with the following criteria: a) high accuracy, b) low computational cost, c)low number of parameters and d) capability to be implemented in equipment with moderate computational resources. The data used to identify and select the models was extracted from comparative studies, such as [10] and results published in the most famous image classification challenge [28].

Quantitative results in terms of accuracy level and the number of operations performed in a forward step by the CNN are depicted in Fig.4.2. Also, the number of parameters were included with the blob size of each model. It can be seen how a model with high accuracy, low computational cost and low number of parameters remains closer to the upper left corner and a small blob such as ENet [68], GoogLeNet [7] , ResNet 34 [9], ResNet 18 [9], ResNet 50 [9], Inception V3 [8] & ResNet 101 [9], meanwhile a model with low accuracy and number of operations as AlexNet remains in the bottom left corner and

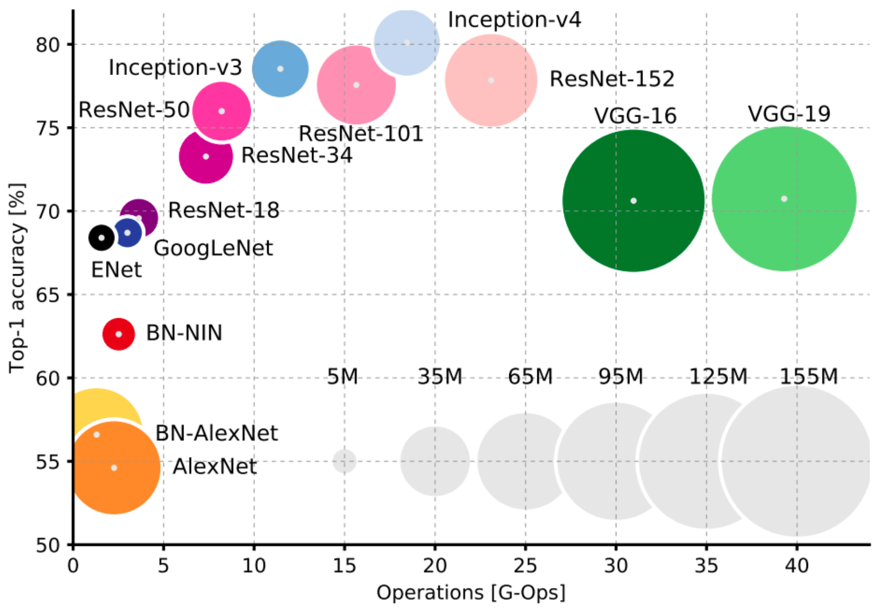


Figure 4.2: Top1 vs. operations, size α parameters. Obtained from [10].

models with high accuracy and computational cost as: VGG 19 [65] and ResNet 152 [9] remain in the right upper corner. AlexNet [69] has a high rate of falls because its low accuracy and models as VGG 19 [65] requires high computational resources to be applied in an effective way. From illustrated results, the selected models are: GoogLenet [7], InceptionV3 [8], ResNet 50 [9] and ResNet 101 [9].

On the other hand, regarding to the simplicity and low computational cost, ZFNet [31] was included in this work. It was characterized in image classification challenge [28] for obtaining lower rates of error as is described in Table 4.1.

	Number of Layers	**Techniques**	**Top 1 Error %**	**Top 5 Error %**
ZFNet [31]	8	Normal Convolution	38.4	11.2
GoogleLenet [7]	22	Auxiliary Classifiers	-	7.89
ResNet 50 [9]	50	Residual Mapping	20.74	5.25
ResNet 101 [9]	101	Residual Mapping	19.87	4.60
Inception V3 [8]	159	Factorizing Convolutions Auxiliary Classifiers	17.3	3.5

Table 4.1: Models Comparison Table

The selected models are individually described in the following.

Layer	**Details**
Input	256x256
Layer 1	224 Conv, 7x7 Max polling
Layer 2	110 Conv, 5x5 Max polling
Layer 3	13 Conv, 3x3 13 Conv, 3x3
Layer 4	13 Conv, 3x3 Max polling
Layer 5	FC 4096 units FC 4096 units

Table 4.2: ZFNet Architecture Details

4.1.1 ZFnet Achitecture

ZFnet [31] is a simple model composed by just 8 layers which are grouped and represented by five 'layers' in Table 4.2. This architecture is characterized by having a max pooling layer after each convolutional one. This architecture has decreasing filters starting with a filter size of 7x7 as is showed in Table 4.2 in 'Layer 1'. Also, ZFNet is known for being an improved version of Alex Net [69], one of the most popular model.The main difference falls in the dimensions of the convolution filters which were changed from 11x11 in Alex Net [69] to 7x7 in ZFnet [31]. The change was performed based on the hypothesis that, bigger filters loss more pixel information which can be conserved with the smaller ones. This model uses the ReLu activation function described in Section 3.4.1 and the iterative optimization of batch stochastic gradient descent.

4.1.2 GoogLeNet - Inception V1

GoogLenet [7] or Inception V1 marked a new state of the art for image classification winning the ILSVRC 2014 [28]. It started to save computational resources with the use of Inception Modules (described in Section 3.4.1)improving its use through the network.

Layer	Details
Input	224x224
Layer 1	112 Conv 7x7 Max Pool 56 Conv 3x3 Max Pool
Layer 2	Inception module } x2 Max pool
Layer 3	Inception module } x5 Max pool
Layer 4	Inception module } x2 Avg pool
Layer 5	FC

Table 4.3: InceptionV1 Architecture Details

Their design was based on the Hebbian principle, a theoretical type of cell activation widely applied in computer science. It has 22 layers of deep, but it was summarized in 5 main layers. It starting with a size filter of 7x7 which are decreasing through the network. They are followed by 9 inception modules as is described in the Table 4.3. GoogLenet [7] take advantage of average pooling before the FC layer to save a lot of parameters and improve the accuracy.

4.1.3 GoogleNet-Inception V3

GoogleNet-Inception V3 [8] is a improved version of previous version presented. By rethinking the inception architecture, computational efficiency and fewer parameters are realized. With 42 layers deep, the computation cost is only about 2.5 higher than that achieved by GoogLeNet[7], and much more efficient than that of VGGNet [65]. It became the 1st Runner Up for image classification in ILSVRC 2015 [28].The most relevant features of Inception V3 are the factorization technique applied trough the network and the inceptions modules described in Section 3.4. The factorization is applied in order to reduce the number of parameters keeping the network efficiency. Inception V3 starts with small filters of 3x3 and keep them untill reach the inception modules as is described in Table 4.4.

Layer	Details
Input	299x299
Layer 1	229 Conv 3x3 149 Conv 3x3 147 Conv 3x3 Max Pool
Layer 2	73 Conv 3x3 71 Conv 3x3 35 Conv 3x3
Layer 3	see Fig. 3.9
Layer 4	see Fig. 3.10
Layer 5	see Fig. 3.9
Layer 6	Max Pool FC Softmax

Table 4.4: InceptionV3 Architecture Details

4.1.4 ResNet 50 & ResNet 101

The Residual Networks ResNet can have a very deep network of up to 152 layers by learning the residual representation functions instead of learning the signal representation directly. ResNet 101 and ResNet 50 [9] introduced skip connection (or shortcut connection) to fit two or more stacked layers in a desired residual mapping instead of hoping that they directly fit. The shortcuts are a solution to the degradation problem (accuracy saturation) generated in the convergence of deeper networks as ResNet. The shortcuts show their effectiveness, improving the learning process in deeper networks as was mentioned in 3.4. ResNet becomes the Winner of ILSVRC 2015 [28] in image classification, detection, and localization. The difference among these two networks is the number of convolutional layers, ResNet 101 has twice layers than ResNet 50 as is described in Table 4.5.

Layer	**Details ResNet 50**	**Details ResNet 101**
Input	224x224	224x224
Layer 1	64 Conv 7x7, Max Pool	64 Conv 7x7, Max Pool
Layer 2	$\left.\begin{array}{l}\text{64 Conv 1x1}\\ \text{64 Conv 3x3}\\ \text{256 Conv 1x1}\end{array}\right\} x3$	$\left.\begin{array}{l}\text{64 Conv 1x1}\\ \text{64 Conv 3x3}\\ \text{256 Conv 1x1}\end{array}\right\} x3$
Layer 3	$\left.\begin{array}{l}\text{128 Conv 1x1}\\ \text{128 Conv 3x3}\\ \text{512 Conv 1x1}\end{array}\right\} x4$	$\left.\begin{array}{l}\text{128 Conv 1x1}\\ \text{128 Conv 3x3}\\ \text{512 Conv 1x1}\end{array}\right\} x4$
Layer 4	$\left.\begin{array}{l}\text{256 Conv 1x1}\\ \text{256 Conv 3x3}\\ \text{1024 Conv 1x1}\end{array}\right\} \mathbf{x6}$	$\left.\begin{array}{l}\text{256 Conv 1x1}\\ \text{256 Conv 3x3}\\ \text{1024 Conv 1x1}\end{array}\right\} \mathbf{x23}$
Layer 5	$\left.\begin{array}{l}\text{512 Conv 1x1}\\ \text{512 Conv 3x3}\\ \text{2048 Conv 1x1}\end{array}\right\} x3$	$\left.\begin{array}{l}\text{512 Conv 1x1}\\ \text{512 Conv 3x3}\\ \text{2048 Conv 1x1}\end{array}\right\} x3$
Layer 6	Avg Pool, 1000 FC, Softmax	Avg Pool, 1000 FC, Softmax

Table 4.5: ResNet 50 and ResNet 101 Architecture Details

4.2 Hardware & Software Selection

In this section the hardware and software tools required to perform this work are described. It includes the computational resources, main libraries, and framework used to: prepare the data, train the models and perform the experiments described in Chapter 5.

4.2.1 Hardware tools

- **High-performance computing (HPC) supercomputer 'Quinde I':** DL training is a challenging task due to the massive used data. For handling the exponentially growing data demand for DL, high-performance computing (HPC) supercomputers are increasingly being used [70]. 'Quinde I' is a HPC supercomputer located in Urcuquí-Ecuador. 168 GPUs and high-speed interconnect network (infiniband) are provided for high-performance data load and processing. It has 84 nodes, each one with processor Power8 and two graphics cards Nvidia K80. The supercomputer was used to train all of the models.

- **Personal Computer:** Trained models deployment was performed on personal computer. It is a HP computer with Intel Core I7 and 16 GB RAM.

4.2.2 Software tools

- **Caffe Framework**: Caffe is a well-known and widely used DL framework for research purposes that provides C, Matlab's and python interfaces for implementation of CNNs. Caffe is mainly intended for image deep-learning applications as was mentioned in Section 3.5. This framework was used to set and train the models regarding the following relevant pros and cons.
  - Pros:
    * Good for image processing and feedforward networks.
    * Good for finetuning existing networks.
    * Train models without writing any code by using available interfaces.
    * Python and matlab interfaces are pretty useful.
  - Cons:
    * Need to write C++ / CUDA for new GPU layers.
    * Not extensible.
    * No commercial support
    * Slow deployment.
- **Python 3.6.9**: It is a preferred programming language widely used for teaching and learning Machine learning. It was selected regarding the following reasons.
  - Very useful and quick for obtaining experimental results and prototyping

  - Easy to learn and read
  - Great data handling capacity
  - Useful and easy to couple with other languages and platforms
  - Great availability of libraries for machine learning, neural networks, and exploratory data analysis

  Python was used to create different scripts which were used to normalize and label the data and run the trained models.

- **Open CV**: It is a computer vision library compatible with Python. It allows to perform the brightness intensity in order to improve the quality of the data set , thus increasing the accuracy.

- **Augmentor:** It is another useful python library commonly used in ML and DL. It performs different augmentation techniques as rotations, zooms, distortions, mirror, crops and others over the data set in order to create a better real-world data set and improve the accuracy in the models.

- **GNU plot:** It is an open source program to generate plots of different dimensions. It was used to plot the results obtained from the tests on the models.

4.3 Data Preparation

4.3.1 Original Data Set

The original data set was obtained from Plant Village [1]. The size of the set was of 326 MB with 13,000 images between healthy and unhealthy leafs with a resolution of 256x256. The data set was divided in : training (85%) , validation (10%) and test (5%) from the original data set.

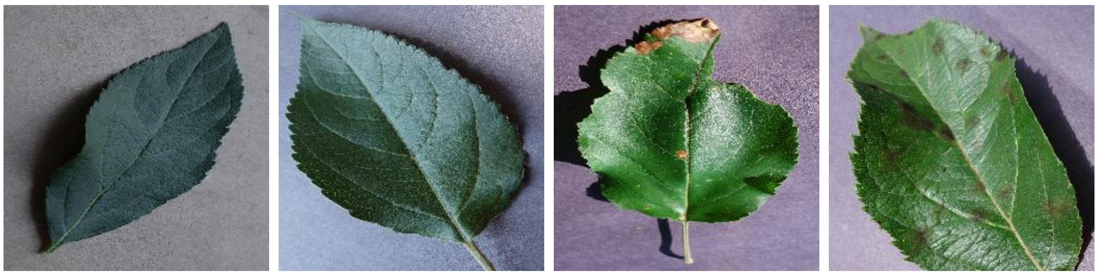

(a) Healthy Leaf (b) Healthy Leaf (c) Unhealthy Leaf (d) Unhealthy Leaf

Figure 4.3: Samples from Plant Village [1]

In order to prepare and improve the quality and quantity of the data set and avoid problems, such as over fitting, data size ,shadows and illumination challenges, as described in Section 3.2.2, the following operations were performed:

1. **Pre-Processing (Brightness and Contrast Variation):** This task is focused on image enhancement using operations for Brightness and Contrast Variation. In Digital image processing, brightness of an image is defined as the amount of energy output by a source of light and the contrast can be explained as the difference between maximum and minimum pixel intensity in the image. Brightness and contrast variations can enhance an image and help to extract some useful information from images. In order to increase the data set, the original data was augmented applying a variation on the bright and contrast of each image.

```
import cv2
    image = cv2.imread(i)
    alpha = 1.5 # Contrast control (1.0-3.0)
    beta = 0 # Brightness control (0-100)
    adjusted = cv2.convertScaleAbs(image,alpha,beta)
f.close()
```

   To perform the operation were varied the alpha and beta values which control the contrast and the brightness to create a new output image. The range of those values are [0.0-100.0] and [1.0-3.0], respectively. For this work, each image from the original data set was adjusted applying two different changes: a) increase contrast an brightness and b)decrease contrast and brightness.

2. **Data Augmentation:** The data set with the brightness and contrast augmentation can be still poor to train a CNN on a real world application and to avoid problems, such as over fitting and other problems, such as rotations, relative location, data set size etc. (some of them were mentioned in Section 3.4), the data set was increased applying the Augmentor library which perform different operations over the image as was described in Section 4.2.2. Particularly, this work applied left and right rotations, distortions,random zooms and flips. To perform it, the following function was implemented:

```

def generateDataAugmented(pathSrc):
    p=Augmentor.Pipeline(pathSrc)
    p.rotate(probability=1,
    max_left_rotation=5,
    max_right_rotation=5)
    p.random_distortion(probability=1,
    grid_width=4, grid_height=4, magnitude=8)
    p.flip_left_right(probability=0.5)
    p.flip_top_bottom(probability=0.5)
    p.zoom_random(probability=0.5, percentage_area=0.8)
    p.flip_top_bottom(probability=0.5)
    p.sample(35000)
```

The most relevant parameter in the function is the sample number (p.sample). It is the final number of images after the data augmentation with originals and the augmented images. For this work we set: 35,000 for training and 10,000 for validation. **The test set was not altered.**

3. **Data Labeling:** To create an LMDB (an image format) all the images must be labeled. We used two labels: healthy and unhealthy, which were encoded in the names of the images after applying the following code:

```bash
#!/bin/bash
cont=0
healthy='unhealthy_'
name='_0'
 for picture in ls *.JPG
 do
 newname=$healthy$cont$nombre
 echo "renaming... $picture"
 echo "a $newname.jpg"
 mv $picture $newname.JPG
 ((cont=$cont+1))
 done
```

4. **LMDB creation:** Caffe framework uses the LMDB format (an image format) to take the images for training and validating the models. To create the LMDBs the script described in A.2.3 was used. The script applies a Caffe function to create the LMDBs. This script also allows resizing the images in order to fit the data to the dimensions for each model.

4.4 Training & Tuning

Before predicting an image class, firstly, all CNN models need to be trained or a pre-trained model needs to be loaded (Transfer learning). To not depend on pre-trained models and validate the pre-processing contribution, this work train all models from empty ones. To train a CNN model, it is necessary to perform the following two steps.

4.4.1 Definition of the CNN Model Structure

The architectures of each model must be described on a prototxt file (configuration file to define the network for caffe framework). The head of file is dived in two phases: train and test. The main parameter in these phases is the batch size (set of images to take simultaneously in one iteration) which be different depending on the CNN architecture and the features of hardware used to train them. After the phases, the model contain the description for all layers of the architecture an example (one single layer) is described in the following:

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
```

Hyper parameter	**Value**
test iter	$\boxed{5}$
test_interval	$\boxed{50}$
display	100
average loss	10
base lr	$\boxed{0.001}$
lr policy	‘step’
type	‘SDG’
stepsize	$\boxed{8000}$
gamma	$\boxed{0.96}$
max iter	$\boxed{10000}$
power	1.0
momentum	0.9
weight decay	0.0002
snapshot	2000
snapshot prefix	‘/snapshot’
solver mode	GPU

Table 4.6: Hyperparameters CNNs. The values in boxes were changed for each model and test performed meanwhile the rest values were conserved.

```
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

The complete architecture description of all evaluated CNNs are presented in Section A.1.

4.4.2 Tuning of the Training hyper-parameters

Once the model is set in a prototxt file, it is need to tell Caffe how the net will be trained by using the configuration file often called solver.prototxt. The hyper-parameters are defined in the solver.prototxt as in Table 4.4.2. All these parameters must be set up based on each application and model for improving accuracy and training time. In the following the parameters will be described:

- **base lr:** This parameter set up the start learning rate in the network. It is has a big influence because each time that the weights of the model are updated the learning rate control the change according the estimated error [71]. The Fig.4.4 shows the influence of the learning rate on the loss through the training. It is defined as:

$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1), \tag{4.1}$$

  Where θ_1 is the new weight, α is the learning rate and J(θ_1) is the gradient.

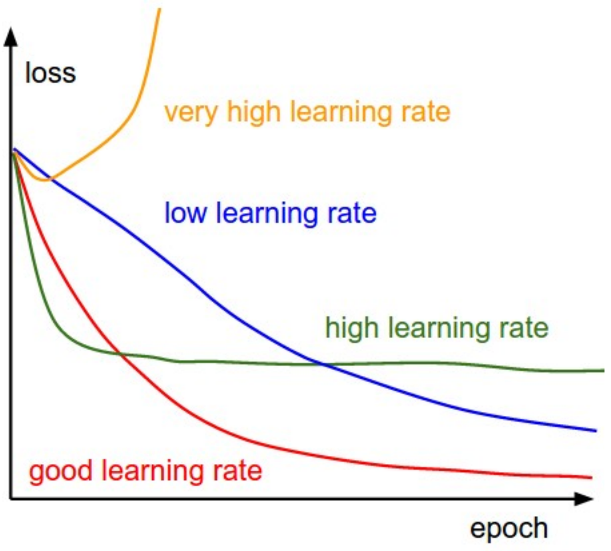


Figure 4.4: Learning Rate Effect. [11]

  For this work, three different values will be tested on each model. These values are highly recommended by [72] because they produce optimal results on different models.

- **lr policy:** It indicates how the learning rate should change over the time. In caffe the following options are available:
  - "step" : It updates the learning rate in the steps defined by the **gamma** parameter.
  - "multistep": It updates the learning rate at each specified **stepvalue**.
  - "fixed": It keeps the learning rate all the time.

  - "exp": The change in the learning rate is defined by: $base_l r * gamma^{iter}$
  - "poly": It follows a polynomial decay of the learning rate.

  For this work, the lr policy was set to "step".

- **gamma:** This parameter set how much the learning rate should change every time we reach the next "step". The gamma parameter in this work was set to 0.1.
- **test iter:** This parameter indicates how many test iterations should occur per test interval. This value is a positive integer an can be approximated by:

$$\text{test iter} = \frac{\text{Number of Validation Images}}{\text{Testing Batch Size}} \tag{4.2}$$

- **stepvalue:** This parameter indicates one of potentially many iteration counts that we should move onto the next "step" of training or refresh the lr. This value is a positive integer. According to [73], the best values correspond to around 80% from the maximum number of iterations.
- **momentum:** This parameter indicates how much of the previous weight will be retained in the new calculation. This value is a real fraction.
- **weight decay:** This parameter indicates the factor of (regularization) penalization of large weights. This value is a often a real fraction.
- **solver mode:** This parameter indicates which mode will be used in solving the network.Options include:CPU,GPU
- **max iter:** This parameter indicates when the network should stop training. The value is an integer indicate which iteration should be the last. An approximated number can be calculated by:

$$\text{max iter} = \frac{\text{Number of Training Images}}{\text{Training Batch Size}} \tag{4.3}$$

- **type:** This parameter indicates the back propagation algorithm used to train the network. This value is a quoted string. Options include:
  - Stochastic Gradient Descent "SGD"
  - AdaDelta "AdaDelta"
  - Adaptive Gradient "AdaGrad"
  - Adam "Adam"
  - Nesterov's Accelerated Gradient "Nesterov"
  - RMSprop "RMSProp"

  For this work, Stochastic Gradient Descent "SGD" was selected.
- **snapshot:** This parameter indicates how often caffe should output a model and solverstate. This value is a positive integer.
- **snapshot prefix:** This parameter indicates how a snapshot output's model and solverstate's name should be prefixed. This value is a double quoted string.
- **net:** This parameter indicates the location of the network to be trained (path to prototxt). This value is a double quoted string.
- **display:** This parameter indicates how often caffe should output results to the screen. This value is a positive integer and specifies an iteration count.

4.4.3 Training of the Model on the Supercomputer 'Quinde I'

After defining the model and the hyper-parameters, each model is trained by sending a 'job' to the LSF (Load Sharing Facility) service of Supercomputer 'Quinde I'. It indicates the model to be trained and the direction to all the required data. To send the job over 4 cores and 1 GPU the script described in A.2.4 was used. In the script is established all the libraries used by Caffe. Last, the scripts executes the Caffe-train and set the solver.prototxt file for each model.

4.5 Deployment

Once the model is trained, a caffemodel file is created. It contains the weights learned during the training. To use the caffemodel obtained it is necessary a prototxt file whit the net definiton (it is similar to net definition used to training) removing the phases of training and test, and the last FC layers because FC ones are provided by the caffemodel file. To be able to use the caffemodel and implement the CNN model to make prediction on new unseen data, the python script described in A.2.5 was created. The python script calls the **mean image**, **the deploy model** and the **caffemodel** to build a classifier. Finally the script is run over the test set and a CSV file is created with the prediction performed by the classifier.

Chapter 5

Experimental Setup

Python software routines have been on purpose implemented to train and evaluate selected CNN models. Training script have been executed on supercomputer ‘Quinde I’, meanwhile validations on new unseen data were performed on personal computer (see hardware tools in 4.2.1). This chapter describes the metrics used to measure and compare the models. Also, it contains description of the different tests performed over each model in order to determine the most appropriate model for leaves image classification The following questions will be answered in detail, with Section 5.1 focusing on the metrics used to measure the performance of each model, and Section 5.2 the experiments performed in this work.

- Which metrics were used to evaluate the models performance?
- Which hyper-parameters are the most influential?
- What experiments and sub-experiments were performed?
- What changes were performed over the raw data set?

5.1 Metrics

The classification performed by the trained models, was measured following parameters, such as : Precision, F1 score, recall, training and deployment time. To describe some of the metrics first it is necessary establish and describe a confusion matrix. In ML and AI a confusion matrix (see Table 5.1) is a tool to see the performance of algorithms employed on supervised learning. In the confusion matrix each column represents the number of predictions of each class and the rows represents the instance in the real class. Over the confusion matrix, the following metrics are measured:

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Table 5.1: Confusion Matrix

where TP corresponds to the healthy leafs correctly classified as healthy ones, TN corresponds to unhealthy leafs correctly classified as unhealthy ones, FP corresponds to classify healthy leaves as unhealthy ones and finally, one of the most relevant for this work, FN which corresponds to unhealthy leaves classified as healthy ones.

- **Accuracy:** It is the most common metric to measure the performance of a model. It is simple a ratio of all correctly predict instances to the total. It is natural to think that a high accuracy means that the model is the best, but it is not entirely true. In fact it is only true if the test set is symmetric and the FP and FN are similar.
$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

- **Precision:** It is the ratio of correctly predicted positive instances to the total predicted positive instances. The question that this metric answer is of leaves that labeled as healthy, how many actually are healthy? High precision relates to the low false positive rate. Presicion is defined as:
$$precision = \frac{TP}{TP + FP} \tag{5.2}$$

- **Recall (Sensitivity):** It is the ratio of correctly predicted positive instances to the all instances in actual class. The question recall answers is: Of all the leafs that truly are healthy, how many did the model labeled as healthy?

$$recall = \frac{TP}{TP + FN} \tag{5.3}$$

- **F1 Score:** It is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. This metric shows how precise a classifier is, as well as how robust. Usually F1 score is more useful than accuracy. For instance, high precision, but lower recall gives an extremely accuracy, but it then misses a large number of instances that are difficult to classify. Mathematically, it can be expressed as:

$$F1 = 2.\frac{recall.presicion}{recall + presicion} \tag{5.4}$$

- **Training time:** This metric measure the compute time during the training phase. This metric was measured by the LSF service on the Supercomputer 'Quinde I' .

- **Execution time:** This metric states the compute time to set a prediction from one to a N number of instances. It was measured with the clock function from the operative system Ubuntu 18.04.

5.2 Experiments

In order to measure the performance of the models under different conditions, three different experiments over all the models were performed. In the first one, the models were tested without any augmentation over the original data. The second test was performed implementing the data augmentation over the data set. Finally, a test affining the hyper-parameters of each model were performed.

5.2.1 Experiment 1: Training with Raw Data

In this experiment the models were trained and deploying applying the original data obtained from Plant Village [1] (13,000 RGB images of 256 x 256, among healthy and unhealthy leaves). As was described in Section 4.1, all the models have different size as input and for that reason a resizing has to be performed on the data set before training phase. The resize was performed during the LMDB creation. In this experiment, the hyper-parameters of the models described in Section 4.4.2 uses default values because this parameters will fine tune in experiment 3.

5.2.2 Experiment 2: Training with Pre-processing

In order to avoid the over fitting because of the poor variation between the classes (health and unhealthy leaves) and obtain a better real world data set, the following pre-processing operations were done over the original data set and tested on the model ResNet 101. ResNet 101 was chosen by the following reasons: a) it is one of the most deeper networks in this work, and b) it is more prone to suffer over fitting with poor data variation as is the case with this work.

- **Brightness and Contrast Augmentation:** In this sub experiment, the contrast and brightness of the all images from the data set were varied creating new images. The purpose is to determine the influence of different variations in the contrast and brightness values shown in Table 5.2. For instance in the lit change, alter the brightness to 80 and contrast to 1.5 generates a clearer picture, highlighting some details. In the dark case, the assigned values create the effect of a shadow over the image and hide some details that may are not relevant for the training.

Change	**Brightness**	**Contrast**
Lit	80	1.5
Dark	20	0.5

Table 5.2: Brightness & Contrast values to perform the experiment.

- **Data augmentation:** It is based on operations as rotations, zooms, distortions, mirror, crops etc. (described in Section 4.3.1). They must be performed over the original data set and tested on the model.

- **Random classes:** In this experiment, the data set was altered adding 6 random classes as is described in Table 5.3 in order to add variation in the data set and to disturb the weights from the FC layers during the training phase trying to avoid an over fitting.

Classes	**Number of Images**
Airplane	700
Brain	73
Chandelier	90
Turtle	80
Jaguar	170
Dog	125

Table 5.3: Classes added to perform the experiment.

- **All changes:** After measuring the effect and relevance produced by the previous changes, a last experiment will be performed. It consisted in generate a bigger data set applying all the pre-processing techniques mentioned before. We started from the original data set applying a brightness and contrast variation. After that, the random classes described in Table 5.3 were added. Finally, a data augmentation over that data set was performed.

5.2.3 Experiment 3: Fine Tune Training

In this experiment, the hyper-parameters, such as base lr, step, test iter and max iteration described in Section 4.4.2 are modified in order to find the optimal values for training each model. The tested values are described in Tables 5.4, 5.5 and 5.6. In order to perform this experiment, the data set with the high results obtained in experiment 2 was used in order to obtain the best performance.

Test #	**lr**	**Test iteration**	**Max iteration**	**Step**
1	0.1	500	3,500	2,800
2	0.1	700	50,000	40,000
3	0.01	500	3,500	2,800
4	0.01	700	50,000	40,000
5	0.001	500	3,500	2,800
6	0.001	700	50,000	40,000

Table 5.4: Hyper-parameter values for tuning process over ResNet 101.

Test #	**lr**	**Test iteration**	**Max iteration**	**Step**
1	0.1	313	2188	1750
2	0.1	500	50,000	40,000
3	0.01	313	2188	1750
4	0.01	500	50,000	40,000
5	0.001	313	2188	1750
6	0.001	500	50,000	40,000

Table 5.5: Hyper-parameter values for tuning process over Inception V3 & ResNet 50.

Test #	**lr**	**Test iteration**	**Max iteration**	**Step**
1	0.1	78	547	438
2	0.1	500	50,000	40,000
3	0.01	78	547	438
4	0.01	500	50,000	40,000
5	0.001	78	547	438
6	0.001	500	50,000	40,000

Table 5.6: Hyper-parameter values for tuning process over ZFNet & GoogLeNEt

Chapter 6

Results

This chapter describes the obtained results of each one of the experiments mentioned in Section 5.2 through figures and tables to have a better visual appreciation. In this regard, the figures have statistical measures based on the confusion matrix 5.1 obtained from the deployment of the trained models.

6.1 Experiments

There are a total of 3 experiments in which their results are shown in the following subsections.

6.1.1 Experiment 1: Training with raw data

The results obtained during the performance of this experiment are divided in 2 sections: Training and Deployment. In the training section we show the results obtained during the training time. Training loss, the error on the training set of data, and test accuracy, the accuracy reached by the model at some iterations. In the deployment section, we show the following metrics: accuracy, precision, recall, F1, training & deployment time (see metrics definition in 5.1). These metrics are obtained after deploying the models over the test set.

1. **Training**

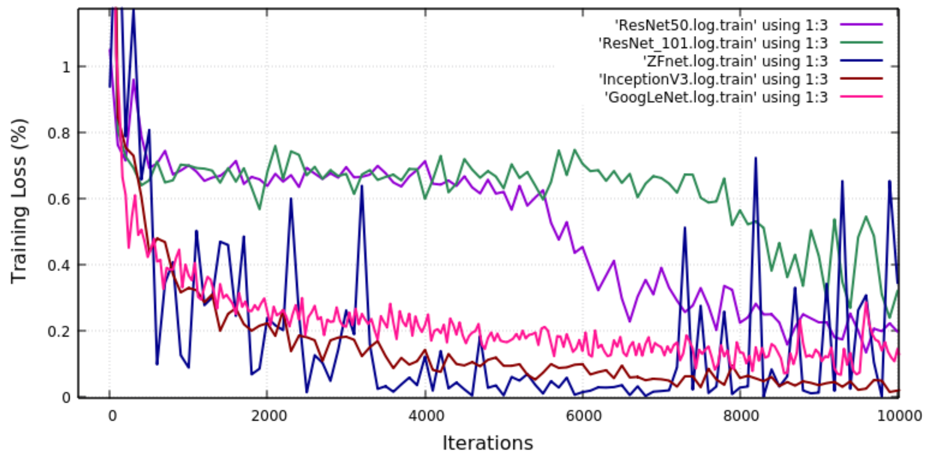


Figure 6.1: Train Loss vs Iterations

Fig.6.1 shows the training loss over the first 10,000 iterations. It can be seen that the models ResNet 50 and ResNet 101 have a similar behaviour because they have similar architectures as is decribed in Section 4.1. A remarkable point is that a small number of layers allows to ResNet 50 converges faster than ResNet 101. In the case of GoogLeNet and Inception 3, which differ in a large number of layers and parameters, have a faster convergence against the ResNet 50 -101. Its important highlight than although Inception 3 has more layers, it converges a little faster than GoogLeNet because the factorization technique applied in the model, which significantly reduces the number of parameters generated in the training. On the other hand, ZFnet shows a particular behaviour, it has a faster but unstable convergence. At 6k iterations converges faster than all the models, but at 8k it seems far away to the convergence. It can be attributed to the poor refining of the model. The model contains just 8 layers, but a lot of parameters which clearly affects the training.

Fig. 6.2 depicts the accuracy obtained during the tests performed in the training time. Similar to results illustrated in Fig. 6.1, it is observed three different patterns. a) ZFNet and Inception 3 obtains a high accuracy in the first 2k iterations and keep it during all the training, it is attributed to the small number of layers which conform ZFnet and the significant reduction of parameters of Inception V3, performed by the factorization technique and the inception modules. b) ResNet 50

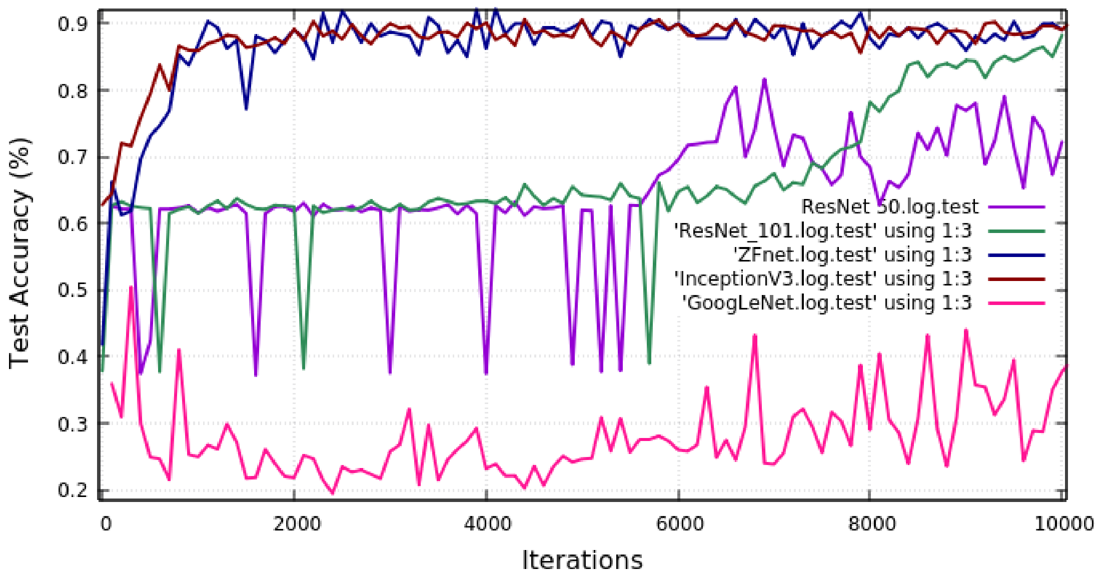


Figure 6.2: Test Accuracy vs Iterations

and ResNet 101 shows a similar and particular behaviour again. The models presented big changes in their accuracy in the first 6.5K iterations, generated while the network gets an established training. After that, they started to establish a growth pattern. That behaviour is caused by the large number of parameters produced by the the layers through the network.

Another important fact is, although the convergence of ResNet 101 is slower than ResNet 50, it reaches a 20% more of accuracy in the same number of iterations. It is an effect produced by the number of parameters managed for a deeper network which retards the convergence in the training, but acquires more specificity and therefore a high accuracy. Finally, GoogLeNet although that the training converges fastest its accuracy was the worst in this experiment. It is attributed to the low number of parameters generated by the model which improves the convergence, but does not assure a reliable accuracy.

2. **Deployment**

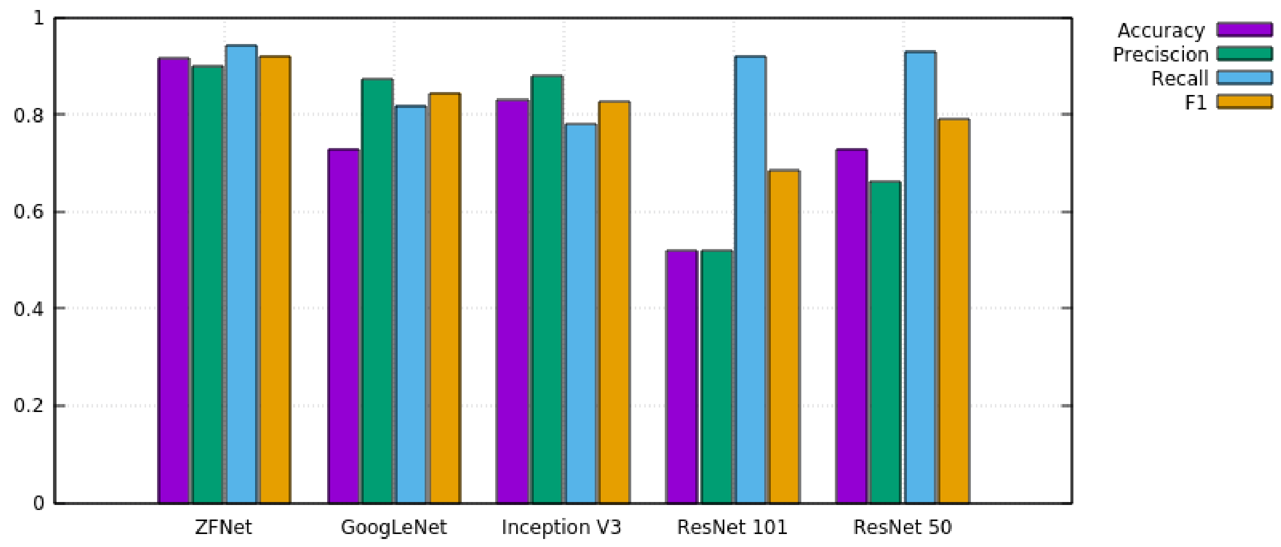


Figure 6.3: Accuracy, Precision, Recall and F1 measures reached by the models.

Fig.6.3 presents the obtained results of Accuracy, Precision, Recall and F1 measured in the deployment phase over each model. It is necessary to point out again that the measures obtained came from the original data set, formed by just two classes which are slightly differentiated. In this experiment, ZFnet, the most shallows model, reached the highest level of accuracy which is balanced with precision, Recall and F1 measures. It is followed by Inception 3 which achieved great results, but the lowest level in Recall showing that a deep model with few parameters fails more labeling healthy leaves as unhealthy. Finally, ResNet 50 and ResNet 101 ,despite its similarity , got completely different measures unlike the measures obtained in the training time. ResNet 101 has the worst accuracy, precision and F1 measures. The poor level of accuracy shows an important problem presented in deep models trained with a unsuitable data set, over fitting. For instance, ResNet 101 always said that the leaves were healthy. This is assured by ResNet 50, which presented an improvement in all the measures, it is attributed to the less layers and therefore less parameters managed by the model.

For timing evaluation, the time per iteration in training phase, and the time per image prediction in deployment phase are illustrated in Fig.6.4. It is necessary stand

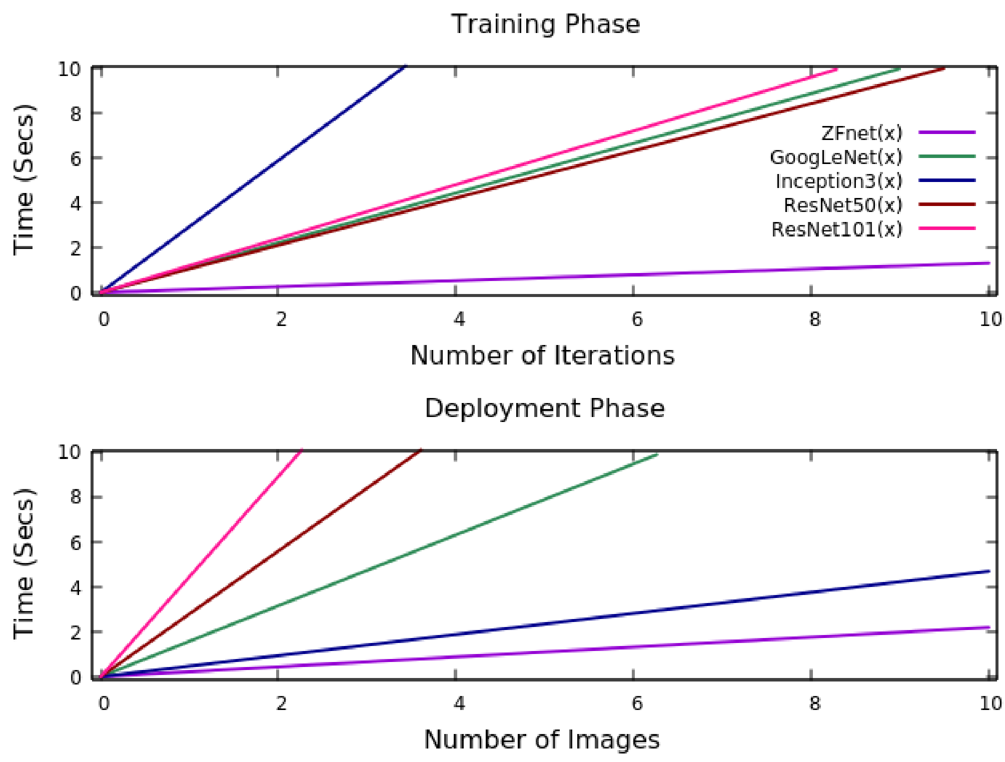


Figure 6.4: Time during the training and deployment phases.

out again the fact that the training was performed over the Supercomputer 'Quinde I' and the deployment over a personal computer I7. However, Fig.6.4 clearly shows a clear difference among the models, in both cases. For instance, Inception 3 spends more time training the model, but it has a very short deployment time which is a great advantage for the purpose of this work.

ResNet 50 and ResNet 101 spend less time training. However, the deployment time is extremely large. ResNEt 101 takes four seconds to classify one single image which is totally opposite to what this work is looking for. ResNet 50 decreases this time according to the number of layers, half, however, it is still too long. GoogLeNet presented a better performance, in both cases is lower than ResNets, but higher than Inception V3 and ZFNet. ZFNet has the best time in the training and deployment phase. It just take 2 seconds in 10 iterations which is extremely fast in the same way ZFnet can classify 10 images in just 2 seconds an incredible advantage for this work.

6.1.2 Experiment 2: Training with pre-processing

In the same way as in the experiment 1, the results are divided in two phases: Training and Deployment. As was detailed in the Section 5.2, the experiment 2 has sub-experiments which were consolidated in the following Figs. 6.6, 6.5 and 6.7 to a better interpretation. Also, all the changes performed over the data set were tested over ResNet 101, the model on which the raw data from experiment 1 mostly affects its performance.

1. **Training:**
   Fig. 6.5 displays the influence of different pre-processing operations over the data set described in Section 5.2, during the training time. It can be seen that adding 'Lit' or 'Lit & Dark' illumination changes over the data set does not alter the training because they presented the same behaviour as was illustrated in Fig. 6.1. However, the convergence in both cases presented abrupt changes over the 3k and 6k iterations. After that, they seem to converge constantly. On another hand, adding random classes and apply the data augmentation seems to influence an almost equal way to training. They present the same behaviour over all the iterations. It gives an idea that data augmentation provides much more variation than an illumination change. Finally, applying all the changes mentioned before, in one data set, affects considerably the training convergence. Because the big variation produced by combine all the changes, the model takes more time to converge also its behaviour looks more unstable over the iterations. Fig. 6.6 shows the accuracy over the iterations of the model with the changes performed over the data. In this case, a clearly effect is presented among 'lit 1' and 'Lit- Dark' illumination change. Although they seem converge to the same accuracy at 10,000, adding 2 illumination changes gives a high accuracy in less iterations, which at first instance could be attributed an over fitting due to the repetition of the images. A similar effect is produced by data augmentation and adding random classes. Adding the random classes gives a higher accuracy in the first 5k iterations, but after it, they present a similar accuracy over

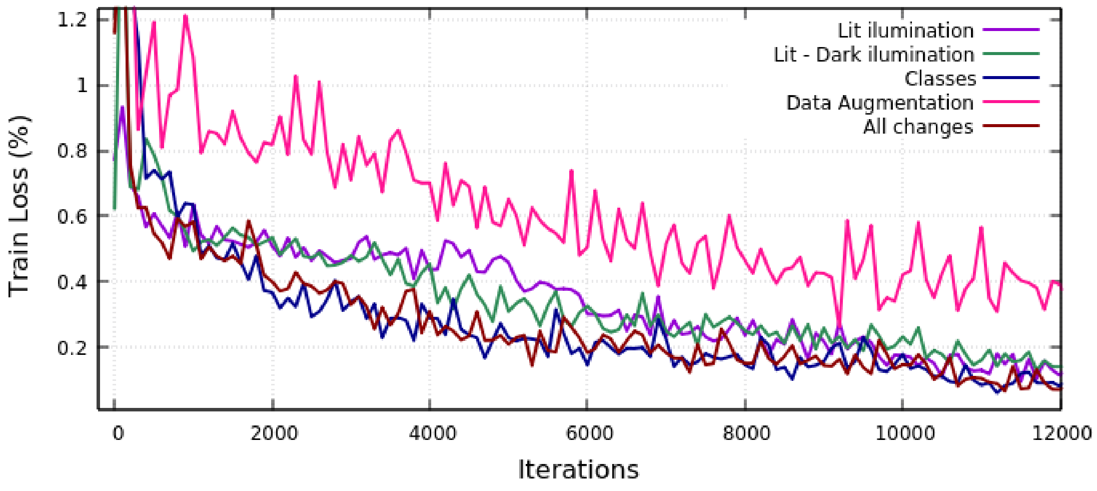


Figure 6.5: Training Loss vs Iterations

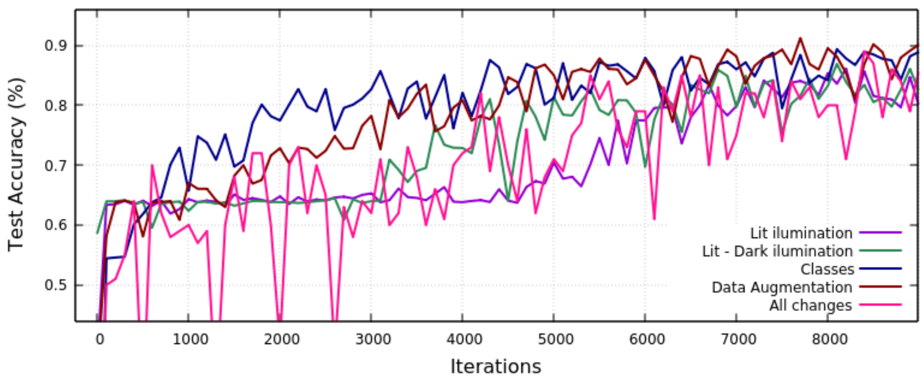


Figure 6.6: Test Accuracy vs Iterations

the iterations. It is due to the greater variation in the data set which causes greater alteration in the weights. However, after certain iterations, the model converges to the same point since the original data set continues to have greater influence during training. Finally, adding all changes, presented an abnormal behaviour in the accuracy. Although the accuracy improves over the iterations, it presented big changes in short intervals. In some points it overcame the accuracy of the models and others presented the worst performance. This is produced by a bigger and varied data set which generate big changes on each test.

2. **Deployment:**

Deploying the trained models with the different variations and measuring their performance, Fig 6.7 was obtained. It also contains, to see the effect of each variation, the results obtained from the raw included at position 6. As is showed in Fig. 6.7, the accuracy of the model decreases when the data was augmented with illumination changes. It can attributed to an over fitting due to the repetition of the same image. It can be supported because the accuracy decreases more when two illumination changes are performed. On another hand, with the random classes added, the accuracy performance diminish a 0.5 %. It was produced because although the variation in the data set was increased, the difference among the two main classes (healthy and unhealthy leaves) was still the same. In the case of data augmentation, a notable increasing of the accuracy is performed because in this case new information, relevant for the the main classes, was added avoiding the over fitting produced in the previous cases. It can be affirmed because, in the last experiment when more variation was added the accuracy of the model increased in a great percentage of almost 20%.

In the precision measures happened something similar, the precision decreases with illumination changes, increases a little with data augmentation and the added classes, meanwhile the precision with all the changes got the best performance.

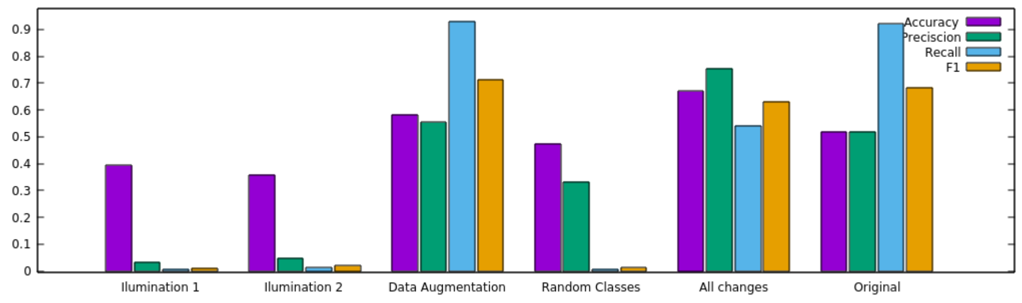


Figure 6.7: Accuracy, Precision, Recall and F1 measures reached by the models.

It is important to highlight that the precision is the most relevant measure in this work because it establish the rate of false negative values. On another hand, with measures as F1 and Recall the performance decreases in all the cases. In other words, applying the variations hinders the model when it must identify healthy leaves, however it improves when it have to identify diseased leaves.

6.1.3 Experiment 3: Fine Tune Hyper-parameters

As was demonstrated the positive impact of adding variations to the data set, the fine tune experiment was developed using that altered data set, called all changes in the previous experiment. To a better visualization and interpretation of the obtained results, they are sub plotted in two sets: the first contains the tests 1,3,5 and the second the 2,4 and 6. Also, as in the previous experiments, the results are divided in: Training and Deployment phase detailed in following.

1. **Training:**

   In Fig. 6.8 are showed the training loss from all tests, performed on Inception 3, separated in two graphics. In both cases, the training loss followed two patterns, one slower and two faster. A high base lr of 0.1 slows down the training from the beginning and produces a bigger training loss even in a large number of iterations, so that choose a right base lr is a key point in the training phase. On another hand, when the learning rate decreases in the next orders of magnitude (0.01, 0.001) the

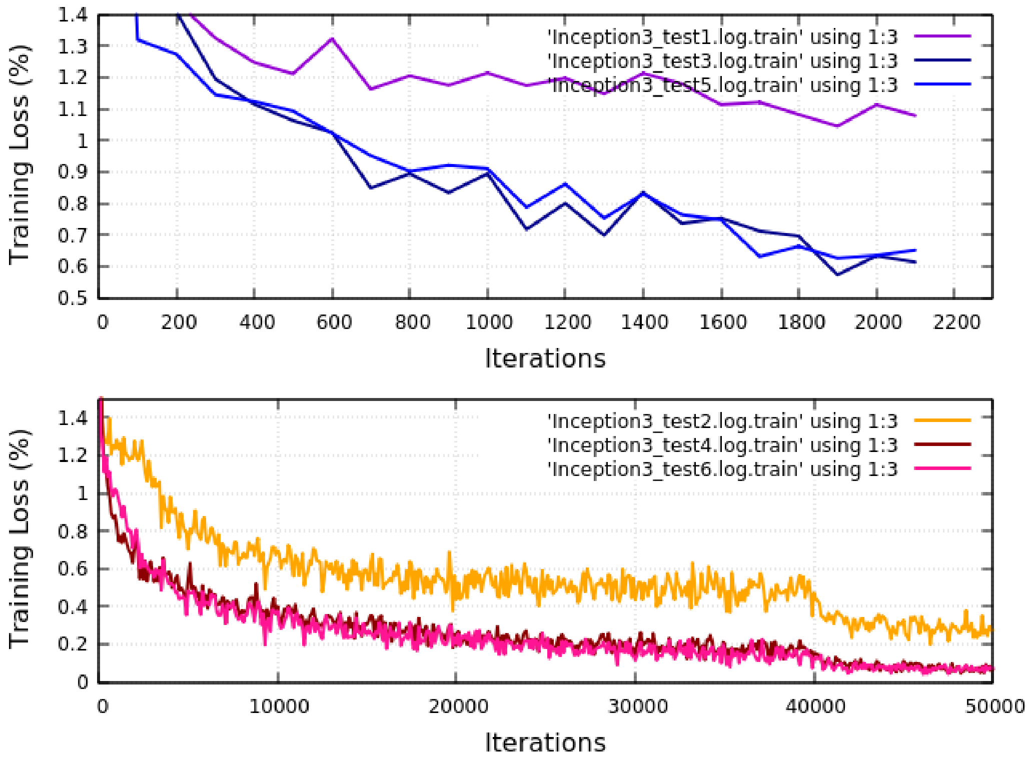


Figure 6.8: Inception V3: Training loss vs Iterations

training loss is almost the same, faster and constant. Inception 3 showed be very susceptible to high learning rates or, in other words, fast changes and stable with low base lr values. It may attributed to depth of the model and the poor parameters generated because the factorization technique and the Inception modules. Another interesting fact showed is the maximum number of iterations, clearly, the model needs more iterations than the described in Eq. 4.3.

In the Fig. 6.9 is showed the accuracy of Inception 3 reached on training from all tests. The reached accuracy is too high, as can be seen in the tests 1,3,5, related with the number of iterations. In this case as before, the performance of Inception 3 with a high base lr is less even in a large number of iterations. Fig. 6.9 showed a zoom close to the 40,000 iterations and there a constant accuracy can be appreciated. Although the high number of iterations the accuracy reached with a high base lr is lesser than the obtained on test 4 and 6. It ensures the impact of training an Inception 3 model with a wrong base lr.

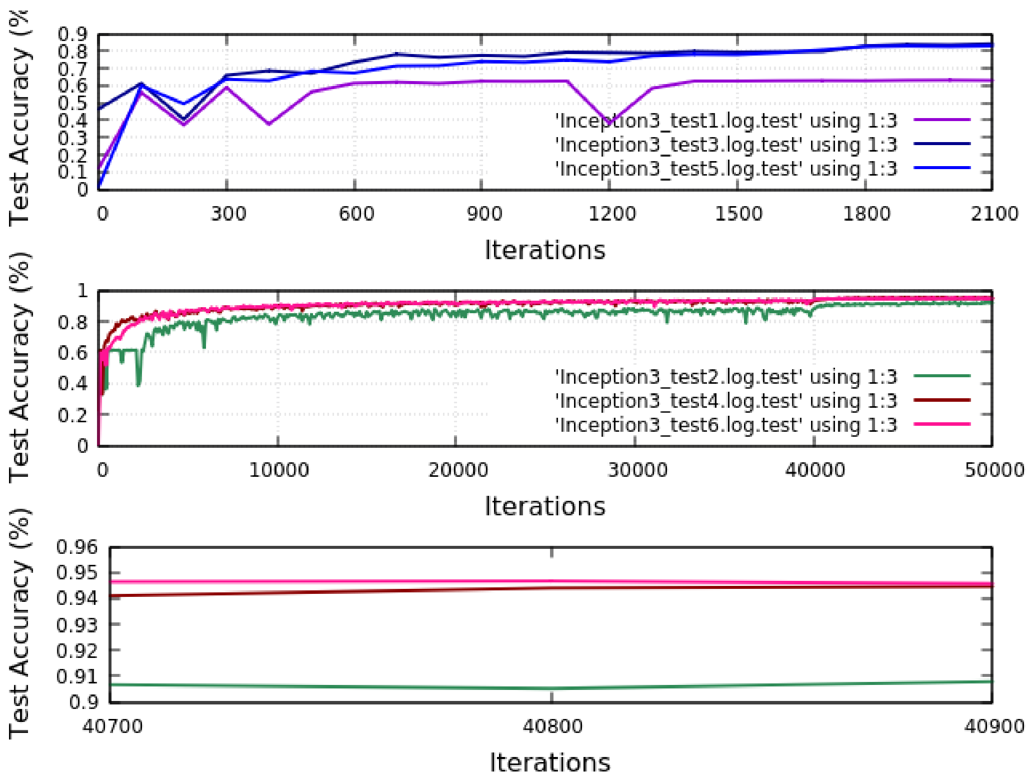


Figure 6.9: Inception V3: Accuracy vs Iterations

In Fig. 6.10 is showed the training loss from all the test performed on ResNet 101. The Fig. 6.10 showed the particular behaviour of ResNet 101, because of a high base lr slows down the training and a very small lr slows down the training too, but in less quantity. ResNet 101, a deep model, showed its susceptibility to the base lr in the first iterations as well as in a large range of them. Also, although a small base lr from test 6 showed a high training loss, it converges to the same point of test 4 (the best for this model), but in more iterations which is also translated in spending more time in training phase. On another hand, a high base lr from test 2 is much slower and converges in a upper point. These behaviour is attributed to fast changes in a deep model with large number of parameters differentiated from Inception 3.

In Fig. 6.11 are showed the accuracy reached by all the tests performed on the models. The patter produced by the base lr showed in the training loss, is also maintained in the accuracy. The accuracy of ResNet decreases with a high base lr

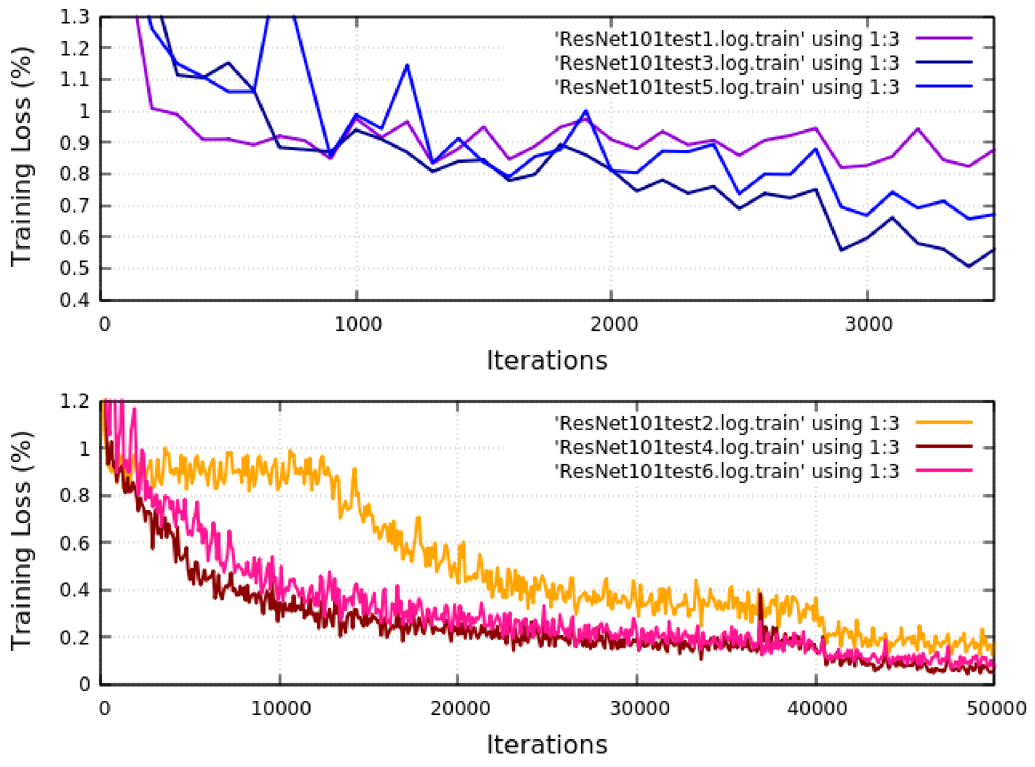


Figure 6.10: ResNet101: Training loss vs Iterations

and also decreases with a very high base lr. In short range, the accuracy presented abrupt changes in the first 2.7k iterations, but after that they look constant. They differ over 10%, a very significant difference. In a wide range, It is possible to appreciate, in a better way, the influence of the base lr. For instance, among 5k and 15k iterations a big is observed among the tests, where the accuracy of a ResNet 101 with a high base lr keeps constant and low and the others increase the accuracy according the iterations. It showed the sensibility of the model with high base lr. However, this difference decreases over 1.5%, in a large range of iterations (around 40,000) as can be appreciated in the zoom.

The training loss performed by ResNet 50 over all test is summarized in Fig. 6.11. In this case, the ResNet 50 showed a behaviour similar to ResNet 101 and Inception 3. The training loss increases (gets worse) with a high base lr and also increase with a very small one as ResNet 101, but to a lesser extent, resembling the behavior of Inception 3. It is produced because ResNet 50 is a deep model, but less than

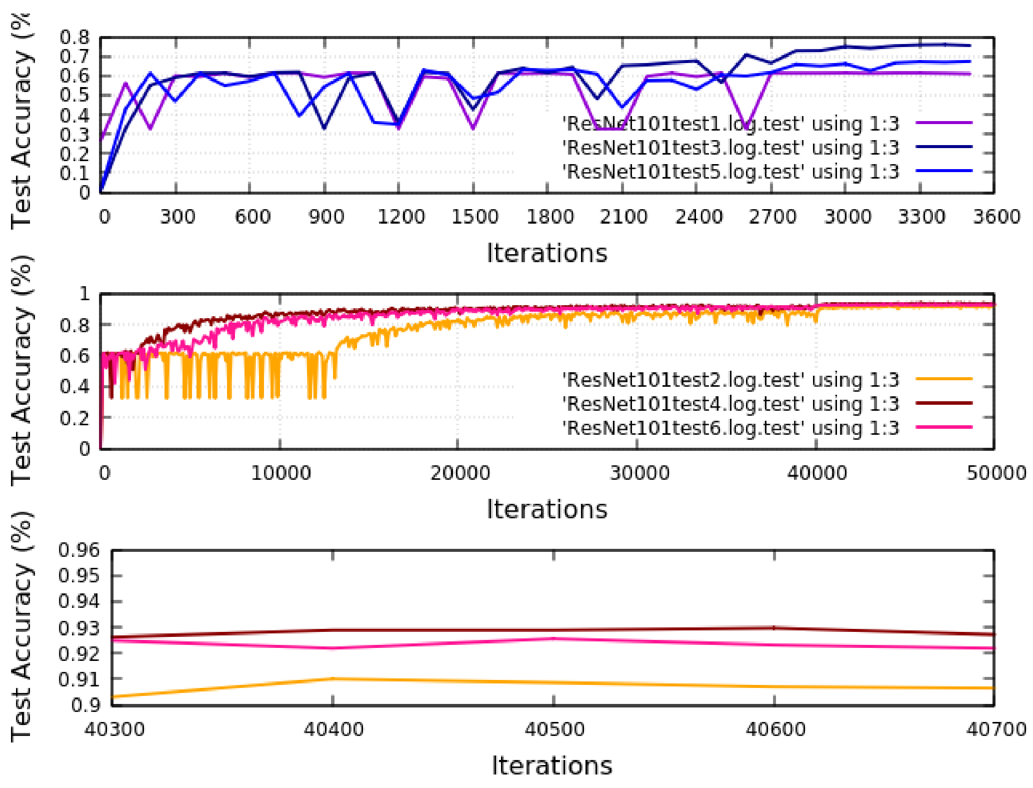


Figure 6.11: ResNet101: Accuracy vs Iterations

ResNet 101 and also manage more parameters than Inception 3 because it does not use techniques to reduce the number of parameters. This effect can be balanced in a wide range of iterations as can be observed in Fig. Fig. 6.11 over the 40,000 iterations, but again spending more time in the training phase. The accuracy of ResNet 50 over all test is described in Fig. 6.13. The accuracy reflected in a better way the influence of the base lr than the training loss. The Fig 6.13 showed a considerable difference amoung the accuracy reached by the test with base lr of 0.1 and the test tests with smaller values. In a short range, the accuracy of the test 1 is lower and does change nothing. In an interval among 4k and 15k, the difference is even more notable. However, after 20k iterations all the tests reached a close accuracy 94%. It can be appreciated in the zoom over the 40,000 iterations, even in some points the test 2 (with a high lr) reached a higher accuracy, which would not necessarily indicate that it is better. ResNet 50 stated a behaviour similar, but no equal to ResNet 101.

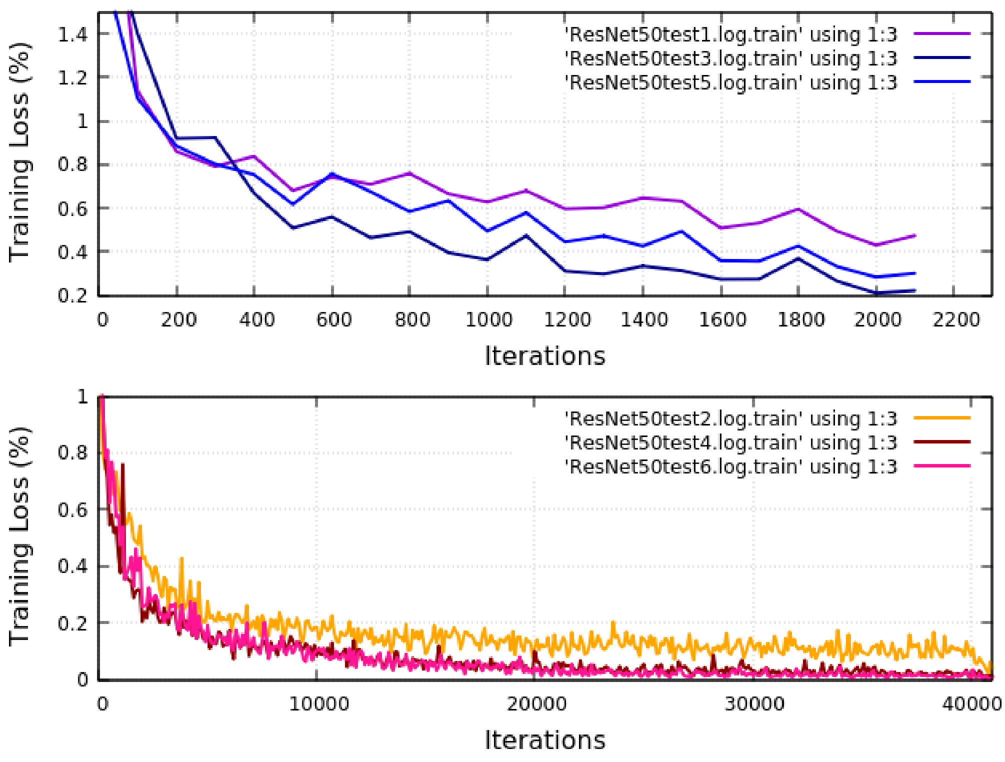


Figure 6.12: ResNet50: Training loss vs Iterations

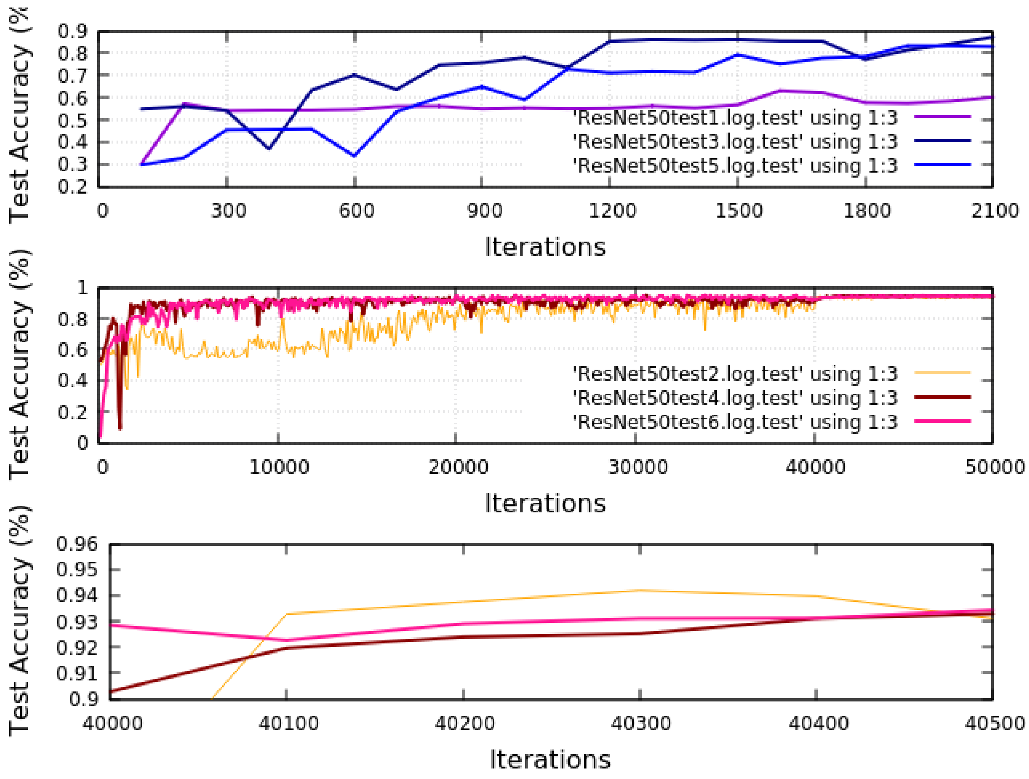
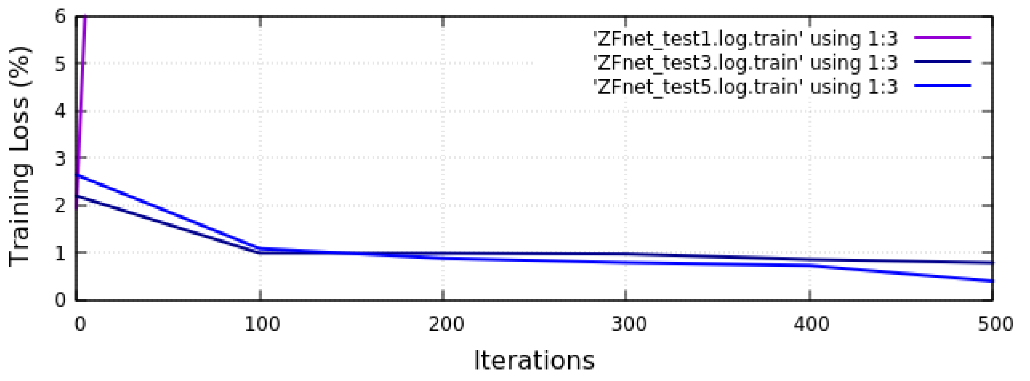


Figure 6.13: ResNet50: Accuracy vs Iterations

In Fig. 6.14 is described the training loss behaviour of ZFNet over the different tests. During the performance of experiment 3, ZFNet showed its high dependency on the base lr. The training loss of ZFNet was infinite, when the learning rate is high as in test 1 and 2. For this reason they can not be observed in the Fig. 6.14. This can be produced because the model manage a large train and test batch size making impossible correctly adjust the weights of the model in a faster way. On another hand, when the learning rate decreases, but is not the adequate, the training loss decreases and converges, but very slow. But, when the base lr is lower and adequate for the model, the training loss decreases at an impressive rate. The sensibility the ZFNet to the base lr can also be attributed to the simplicity of the model. ZFNet is formed by few layers what make it unstable to large and fast changes.

Fig. 6.15 showed the accuracy reached by ZFNet during the training time. The accuracy results of ZFNet reflects that the model can not be trained at all with a large base lr, as was showed in Fig. 6.14. In Fig. 6.15 is showed that the accuracy collapsed to zero in these cases (test 1 & 2). On another hand, when the lr decreases in one and two orders of magnitude, the accuracy gets highly improved. A great difference among these tests can be seem in the first 10,000 iterations, after that, around the 40,000 iterations they have a small, but relevant difference (over 5%), in this case unlike other models like ResNet 101, ZFNet does not reach the same accuracy even in a large range of iterations. It remarks that ZFNet performance depends on the correct choose of a base lr.

In Fig. 6.16 is showed the training loss and accuracy of googLeNet over the all tests. Analyzing the results of GoogLeNet, the pattern formed by its behaviour shows similar to ZFnet and Inception V3 because a high base lr affects training of the model in a big proportion. In fact, it tends to infinite, as ZFNet, and it can not be seen in the graph. As in ZFNet, this is attributed to the large size of the training and test batch. Although GoogLenet is a more sophisticated model than ZFNet, It also is not able to adjust the weights of the model in a faster way managing big amount of data. On another hand, when the base lr decreases in different orders of



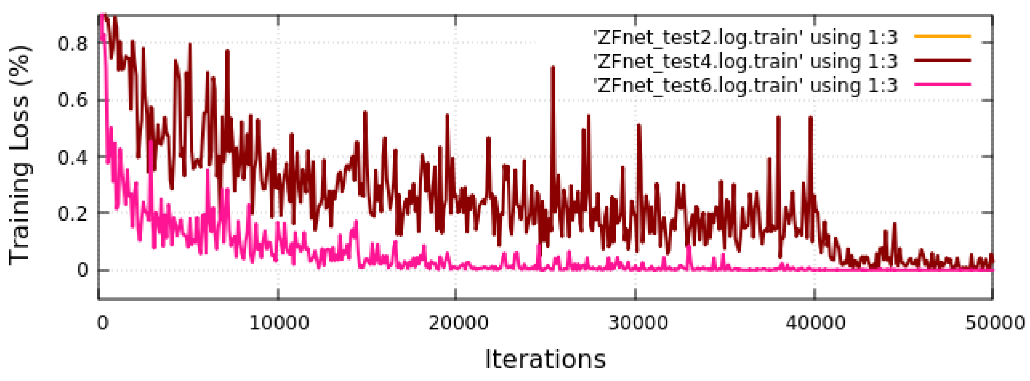


Figure 6.14: ZFNet: Training loss vs Iterations

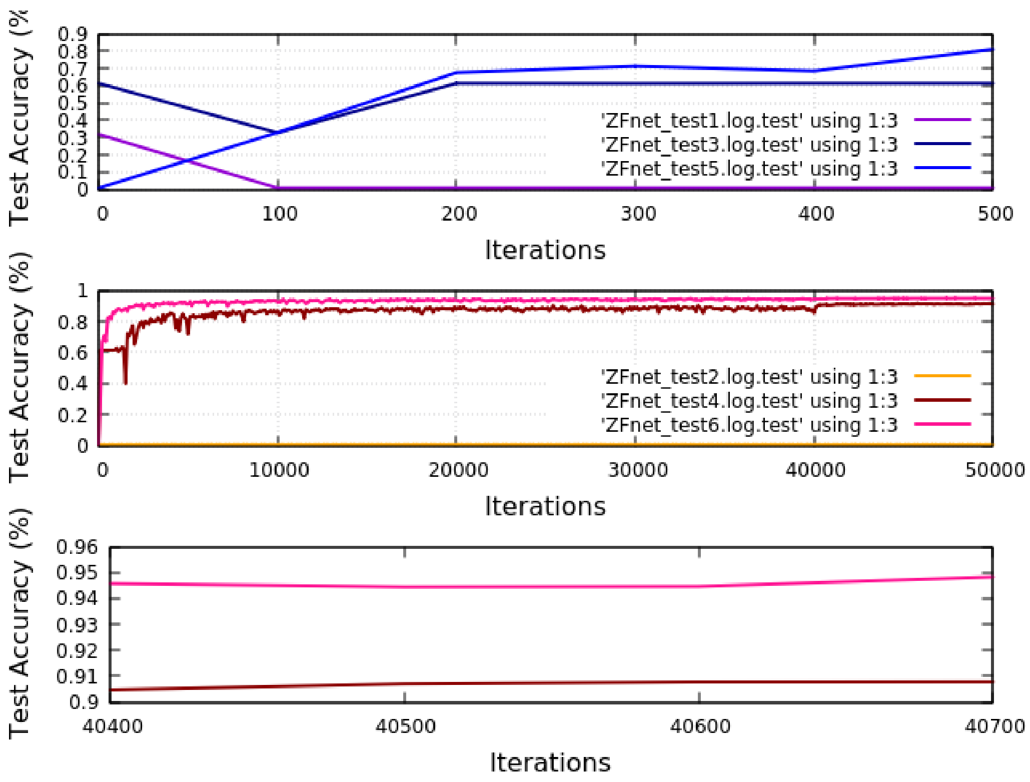


Figure 6.15: ZFNet: Accuracy vs Iterations

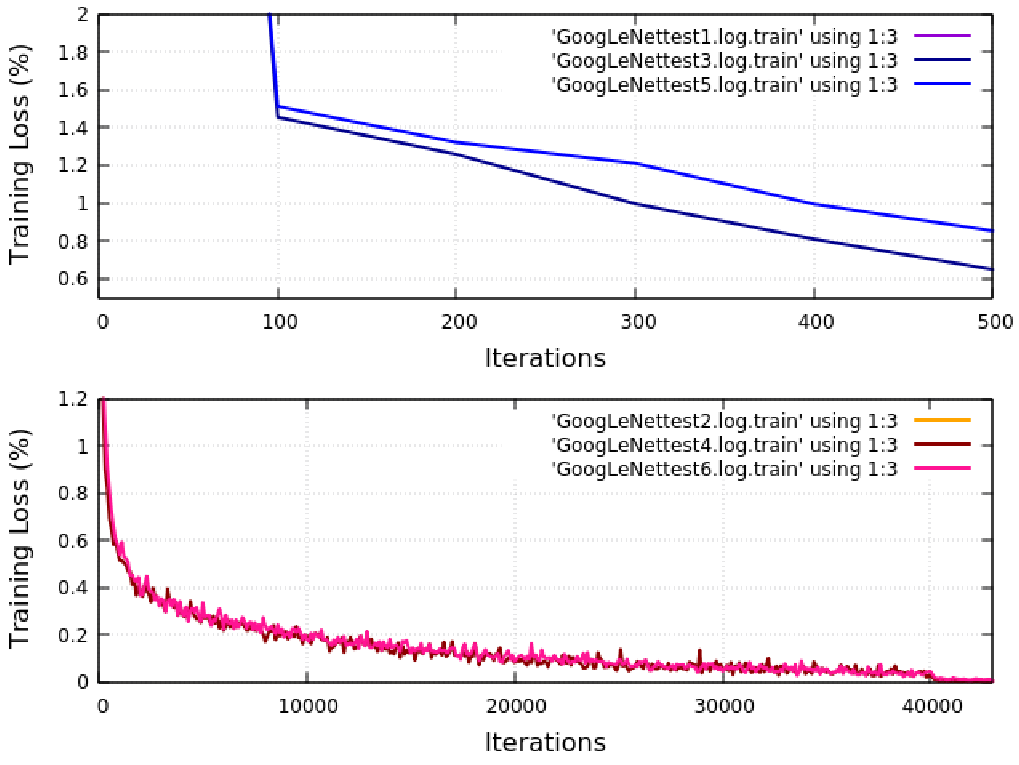


Figure 6.16: GoogLeNet: Training loss vs Iterations

magnitude the difference in the patters of the training loss is very small, similar to Inception V3. This behaviour has much sense because its architecture is similar to Inception v3, but more robust. The training loss of GoogLeNet with an 'optimal' base lr (in this case 0.001) decreases fast almost as ZFnet.

In the same way as in training loss, the accuracy reached by GoogLeNet, in the case of test 1 and 2 with a high base lr, falls down to zero and it can not be appreciated in the graph. On another hand, when the base lr decreases under 0.01 the behaviour is practically the same, even in the zoom over the 40,000 iterations the performance reached by the test are very close as the Inception 3 behaviour explained before.

2. **Deployment:**

In this phase are showed the measures obtained after deploying the models in the different tests, the performance results obtained of these experiments are summarized

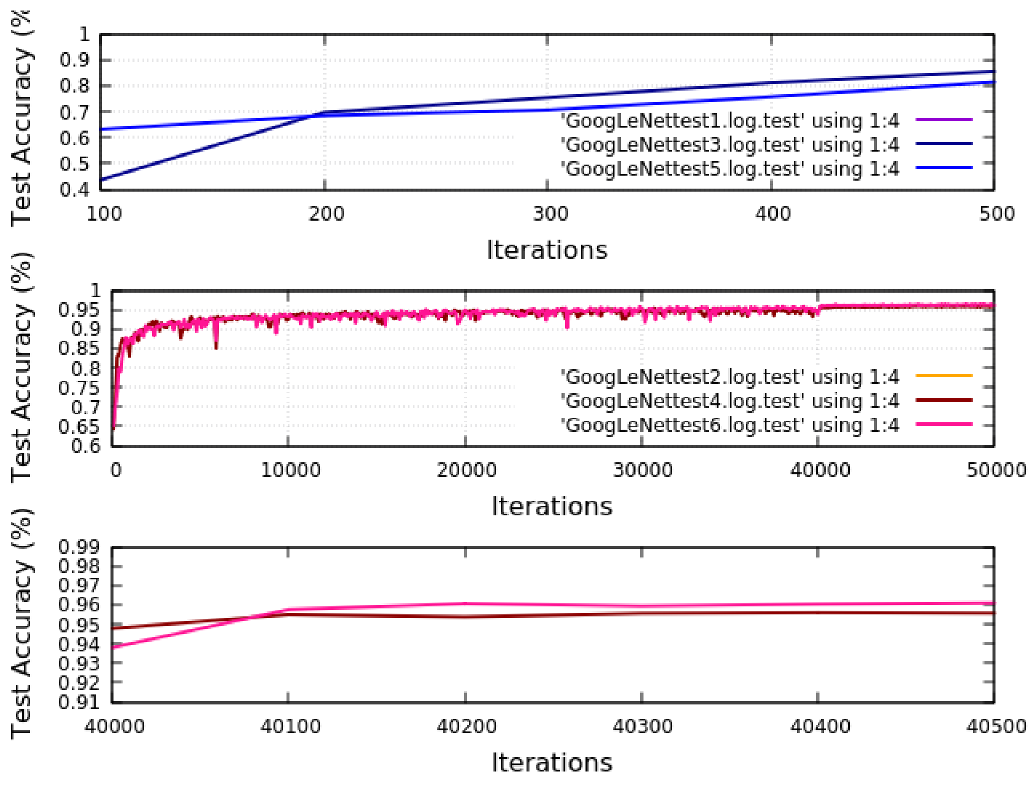


Figure 6.17: GoogLeNet: Accuracy vs Iterations

in Table 6.1.

GoogLeNet, as was described in the training phase, showed poor results with a high base lr. For this reason, the measures obtained from the test 1 and 2 are very close to zero in all the cases. The model showed a faster training because the performance does not change significantly with a large increment in the number of iterations. It is produced because the large train and test batch size of the model, which gives a fast learning with an adequate base lr.

Inception V3 presented similar result in almost all tests (except test 1 & 6). This model got high accuracy with different learning rates and few iterations which were kept over many iterations. The most relevant fact is produced on test 6 (apparently the optimal conditions for Inception 3) where the accuracy falls to its worst point (under 50%). It is presumed to an over fitting because of the number of iterations.

ResNet 50 had an accuracy under 55 % in all the tests, a very low performance against the other models. This fact is mostly attributed to the data set used to

Model	Test #	Accuracy	Precision	Recall	F1
GoogLeNet	1	0.48	0.50	0.006	0.01
	2	0.48	0.50	0.006	0.01
	3	0.80	0.77	0.94	0.84
	4	**0.87**	0.83	**0.95**	**0.89**
	5	0.77	**0.94**	0.63	0.75
	6	0.81	0.76	0.96	0.85
Inception V3	1	0.55	0.54	**0.89**	0.67
	2	**0.77**	0.78	0.79	**0.79**
	3	0.71	0.77	0.64	0.70
	4	0.74	**0.96**	0.51	0.67
	5	0.73	0.82	0.62	0.71
	6	0.48	0.50	0.006	0.01
ResNet 50	1	0.32	0.37	**0.41**	**0.39**
	2	0.09	0.03	0.01	0.02
	3	0.24	**0.53**	0.25	0.34
	4	0.10	0.06	0.04	0.05
	5	0.32	0.30	0.12	0.17
	6	**0.52**	0.03	0.01	0.02
ResNet 101	1	0.44	0.42	0.21	0.28
	2	0.79	**0.94**	0.65	0.77
	3	0.48	0.50	0.006	0.01
	4	0.81	**0.94**	0.68	0.78
	5	0.52	0.52	**1.00**	0.69
	6	**0.86**	0.84	0.91	**0.87**
ZFNet	1	0.48	0.50	0.006	0.01
	2	0.48	0.50	0.006	0.01
	3	0.52	0.52	**0.99**	0.69
	4	0.52	0.52	**0.99**	0.68
	5	0.83	0.83	0.89	0.86
	6	**0.93**	**0.91**	0.97	**0.94**

Table 6.1: Fine tune results

the different tests (called all changes in experiment 2) because in all the variations performed on the model, the accuracy obtained was poor.

ResNet 101 showed that although the the base lr change, the accuracy reached by the model does not change significantly unless the iterations increases. This model showed the best improvement on the performance against raw data experiment.

ZFNet does not work with high lr, in fact the training loss tends to infinite. This is also reflected in the deployment phase, where the 4 first test presented a poor performance in all the measures. However in the last two test, with a smaller base lr, the accuracy and all the measures obtained increase in a significant percentage (around 40%). It important highlight that this improvement happens in just few iterations (test 5) reaching a 80% and increase a 10% after many other iterations (test 6).

6.1.4 Additional Results

ROC Curve

In the Fig. 6.18 is showed the ROC curve (Specificity vs Sensitivity) formed by the performance obtained by each model. For this work, Sensitivity measures the proportion of actual positives that are correctly identified as such healthy leaves that are correctly classified as healthy leaves whereas the specificity measures the proportion of actual negatives that are correctly identified as such unhealthy leaves that are classified as unhealthy leaves. In this sense, in Fig. 6.18 shows the high performance reached by the model ZFNet in both cases. It is followed by ResNet 101, GoogLeNet, Inception 3 and ResNet 50 respectively. The ROC curve showed in Fig. 6.18 reaffirms the worst performance described and explained before of ResNet 50.

A graphic user interface

As additional results, a graphic interface, developed in Python, was created in order to replicate and use the five trained CNNs. The application has the option to load an image

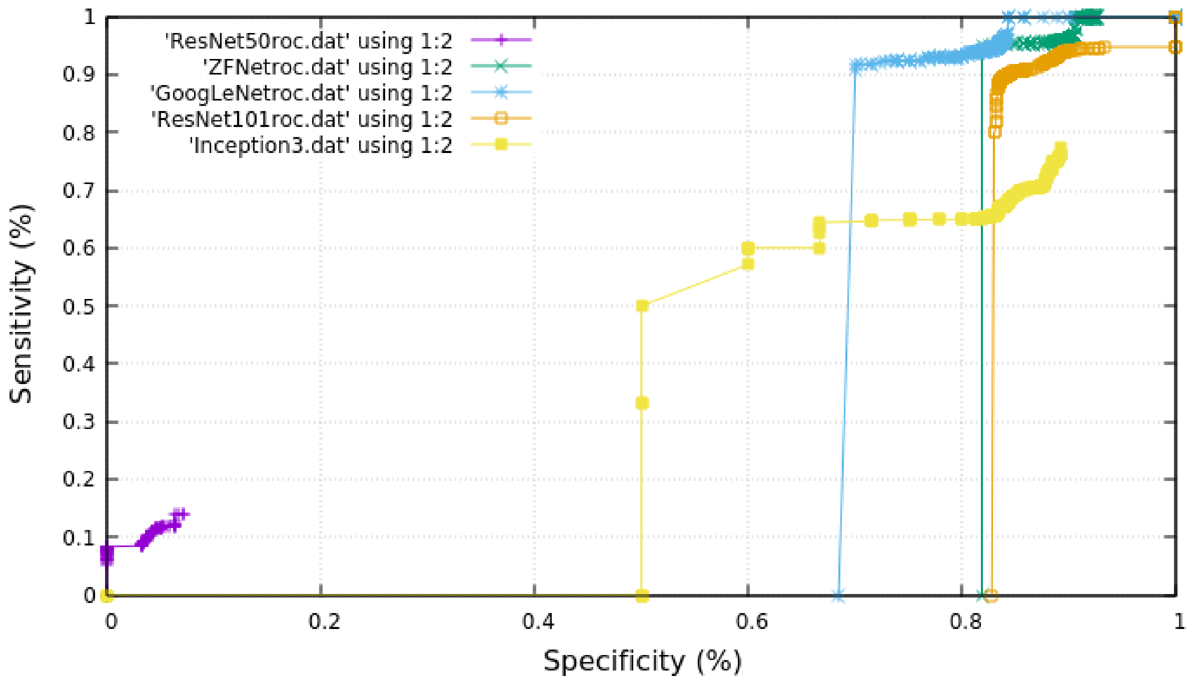
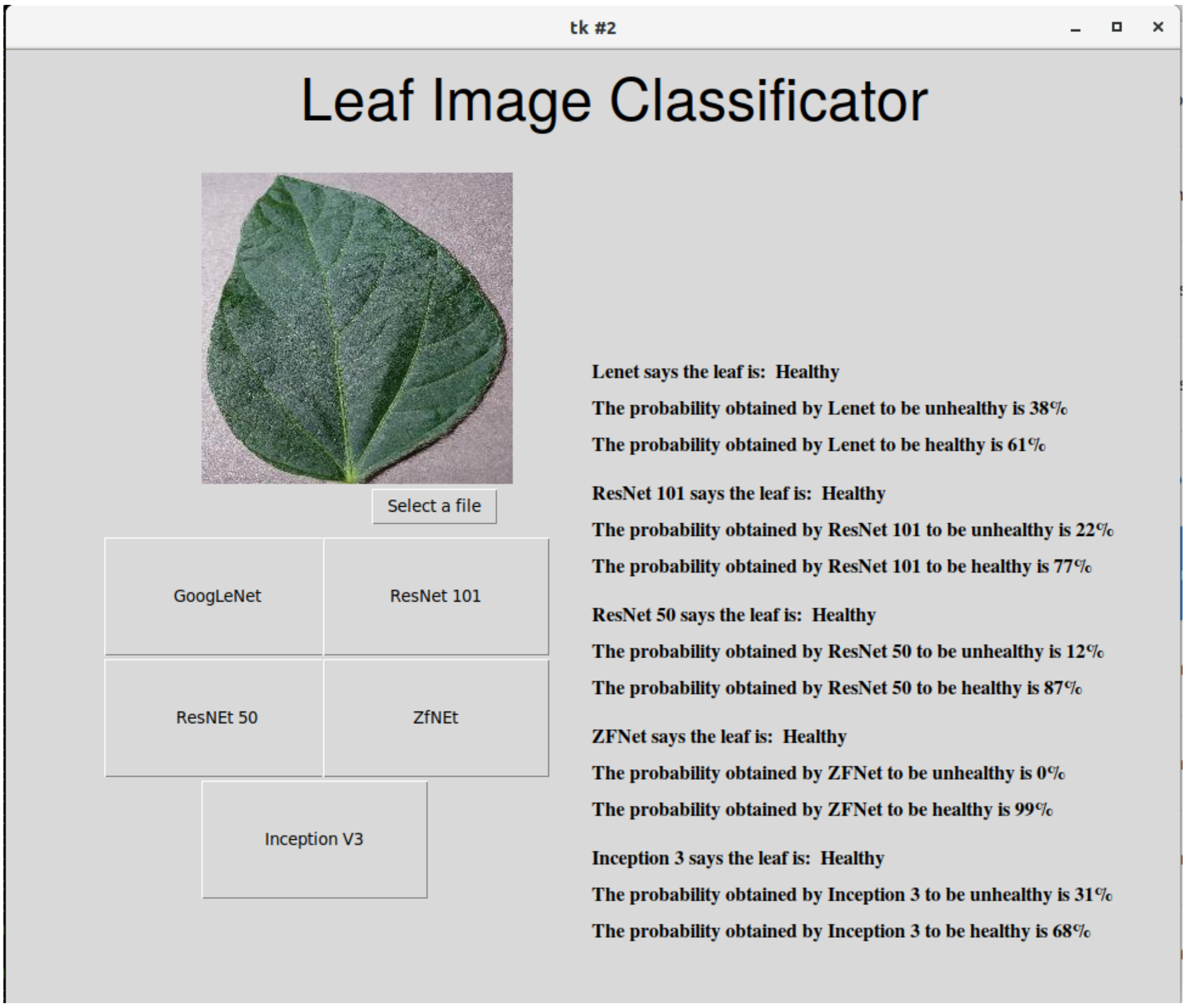


Figure 6.18: Performance measurement - ROC Curve for all models.

(a leaf image) and predict if the leaf is health or unhelthy. Additionally, the interface shows the probability to belong each class. In Fig.6.19 is a general view of the graphic user interface once the predictions were done. The source code is available in A.3.



Lenet says the leaf is: Healthy

The probability obtained by Lenet to be unhealthy is 38%

The probability obtained by Lenet to be healthy is 61%

ResNet 101 says the leaf is: Healthy

The probability obtained by ResNet 101 to be unhealthy is 22%

The probability obtained by ResNet 101 to be healthy is 77%

ResNet 50 says the leaf is: Healthy

The probability obtained by ResNet 50 to be unhealthy is 12%

The probability obtained by ResNet 50 to be healthy is 87%

ZFNet says the leaf is: Healthy

The probability obtained by ZFNet to be unhealthy is 0%

The probability obtained by ZFNet to be healthy is 99%

Inception 3 says the leaf is: Healthy

The probability obtained by Inception 3 to be unhealthy is 31%

The probability obtained by Inception 3 to be healthy is 68%

Figure 6.19: Screenshot of the graphic interface

Chapter 7

Conclusions and Future work

In general terms, the main goal of this work is evaluate and select the most appropriate technique and model to implement an automatic leaf disease classifier. At a more technical and specific level, the following conclusions were reached:

- Deep learning technique and later the models were selected after a state-of-the-art revision of image classification works. The explored models used for this work are: Inception V3, GoogLeNet, ResNet 101, ResNet 50 and ZFNet. The models were selected under a comparison study of accuracy, forward time and number of parameters and the top 1 and 5 error reached on the ILSVRC challenge [28] at different years. The CNN models explored written in Caffe language were extracted from a repository on GitHub. The scripts used to: prepare the data set, train the models over the Supercomputer 'Quinde I' and deploy the models have been written in Python 3.6.9. All of these models were adapted to be tested with the different data sets used in this work.

- The models were trained and tested under the same computational conditions. To determine the models performance with different data versions, three experiments were done: Training with raw data, training with data pre-processing and training with fine tune. To compare the models metrics as accuracy, precision, Recall, F1 , training and execution time were used. Moreover, it presents figures like box plots where it is analyzed the training loss and accuracy over the training. In all the experiments, the CNN ZFNet proved be the best candidate for a leaf disease classifier because its high accuracy (greater than 90%) under different conditions and its low training and deploying time. However, Inception 3 also presents excellent results in accuracy (around 90% in its best conditions), precision, recall, F1 and deployment time in all the experiments. Its Achilles's heel or main disadvantage is the training time, Inception needs a lot of time to train the model and very susceptible with the data pre-processing. The rest of the models as ResNet 50, ResNet101 and GoogLenet showed a good performance on different metrics in experiment 1, but they also presented shortcomings in the experiment 2 and 3.

- The performance of CNN models improved when applying techniques of pre-processing on the data set as brightness and contrast variations, data augmentation, random classes and all changes. ResNet 101 and GoogLeNet showed an relevant increasing on their performance, meanwhile ResNet 50 and Inception 3 showed a decreasing on the performance. ZFNet kept and increase its performance in less quantity than ResNEet 101 and GoogLeNet.

- Prove the influence and the relevance of fine tune the hyper-parameters on the models in the trainig phase as well as in the performance reached in the deployment phase. In this work, the base lr as well as the max iter showed the big impact over the models. Even in some cases as GoogLeNet, a bad base lr prevents the training and therefore the deploying on the model.

- Finally, this work concludes and establishes ZFNet as an optimal model to develop a Leaf Image classifier. It is based on the obtained values in the metrics as accuracy, precision, training and deploying time from all the tests performed on this work.

As future work, it is proposed to implement, on real scenarios, the selected CNN architecture exploring appropriate hardware and software requirements. Also, new CNN alternatives to analyse input data from devices on real-time are going to be studied. In addition, pre-processing task based on spectral filtering and techniques of image processing will be explored to enhance images and reduce computational complexity, which can be mainly applied to High Dimensional Images such as satellite images. Afterwards, it is proposed compare the efficiency reached applying the transfer learning techniques. Furthermore, leaves disease classification with selected CNN model are to be explored with images extracted from different sources as drones, satellite and recorded videos in order to asses the suitability for real application in Ecuador to identify diseases on leaves from large crops such as bananas and potatoes.

References

[1] T. O. E. (2018) Plantvillage dataset. [Online]. Available: https://www.kaggle.com/emmarex/plantdisease

[2] N. Feller. (2018) What are convolutional neural networks and deep belief network weakness? [Online]. Available: https://www.quora.com/What-are-the-current-problems-with-the-use-of-convolutional-neural-networks-CNN-for-vision.

[3] K. Cook. Know How Deep Learning Works? Here's A Quick Guide For All Engineer. (2019, august 29). [Online]. Available: https://www.houseofbots.com/news-detail/11733-4-know-how-deep-learning-works-heres-a-quick-guide-for-all-engineer

[4] S. Puran. (2020) Will capsule neural network replace traditional neural networks? [Online]. Available: https://www.quora.com/Will-Capsule-Neural-Network-replace-Traditional-Neural-Networks

[5] A. Kesarwani. What is max pooling in convolutional neural networks? (2019, august 29). [Online]. Available: https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks

[6] missinglink. aii. A Beginner's Guide To Understanding Convolutional Neural Network. (2019, September 1st). [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[7] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, pp. 1–9.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567. [Online]. Available: https://arxiv.org/abs/1512.00567

[9] H. Kaiming, Z. Xiangyu, R. Shaoqing, and S. Jian, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[10] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," 05 2016.

[11] Z. H. (2018) Understanding learning rates and how it improves performance in deep learning. [Online]. Available: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

[12] A. Shatnawi, M. Al-Ayyoub, g. albdour, and R. Al-Qurran, "A comparative study of open source deep learning frameworks," 04 2018.

[13] S. Savary, A. Ficke, J.-N. Aubertot, and C. Hollier, "Crop losses due to diseases and their implications for global food production losses and food security," *Food Security*, vol. 4, 12 2012.

[14] S. Lim. (2019) Gpu vs cpu deep learning: Training performance of convolutional networks. [Online]. Available: https://phoenixnap.com/blog/gpu-deep-learning

[15] R. S. H. Sadiyah Abdullahi and F. Mahieddine, "Convolution neural network in precision agriculture for plant image recognition and classification," 08 2017, pp. 1–3.

[16] A. Kamilaris and F. X. Prenafeta-Boldú, "A review of the use of convolutional neural networks in agriculture," *The Journal of Agricultural Science*, vol. 156, no. 3, p. 312–322, 2018.

[17] M. S. L. M. Dyrmann, S. Skovsen and R. N. Jorgensen, "Using a fully convolutional neural network for detecting locations of weeds in images from cereal fields," 04 2018.

[18] S. Arivazhagan, S. Newlin, S. Ananthi, and V. Varthini, "Detection of unhealthy region of plant leaves and classification of plant leaf diseases using texture features." *Agricultural Engineering International: CIGR Journal*, vol. 15, 2013. [Online]. Available: https://www.researchgate.net/publication/287577015_Detection_of_unhealthy_region_of_plant_leaves_and_classification_of_plant_leaf_diseases_using_texture_features

[19] M. Hrabia. (2020) Deep learning vs. machine learning. [Online]. Available: https://towardsdatascience.com/deep-learning-vs-machine-learning-e0a9cb2f288

[20] H. H. Guides. (2019) Control de plagas y enfermedades de las plantas. [Online]. Available: https://es.hesperian.org/hhg/A_Community_Guide_to_Environmental_Health:Control_de_plagas_y_enfermedades_de_las_plantas

[21] H. Bischof, W. Schneider, and A. Pinz, "Multispectral classification of landsat-images using neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 30, pp. 482–490, 1992. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/142926

[22] T. Rumpf, A.-K. Mahlein, U. Steiner, E.-C. Oerke, H.-W. Dehne, and L. Plümer, "Early detection and classification of plant diseases with support vector machines based on hyperspectral reflectance," *Computers and Electronics in Agriculture*, vol. 74, pp. 91–99, 10 2010.

[23] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christe, and S. Mougiakakou, "Lung pattern classification for interstitial lung diseases using a deep convolutional neural network," *IEEE Transactions on Medical Imaging*, vol. 35, pp. 1–1, 02 2016.

[24] D. Bhimrao Kadam, S. Gade, M. Uplane, and R. Prasad, "Neural network based brain tumor detection using mr images," *Int. J. Comp. Sci. Communications*, vol. 2, pp. 325–31, 01 2011.

[25] W. Hu, Y. Huang, L. Wei, F. Zhang, and H. Li, "Deep convolutional neural networks for hyperspectral image classification," *Journal of Sensors*, vol. 2015, pp. 1–12, 07 2015.

[26] M. A. A. Sladojevic, S.and Arsenovic, D. Culibrk, and S. D., "Deep neural networks based recognition of plant diseases by leaf image classification," *Computational Intelligence and Neuroscience*, vol. 2016, p. 11, 2016. [Online]. Available: https://www.hindawi.com/journals/cin/2016/3289801/

[27] Y. Jo, S. Park, J. Jung, J. Yoon, H. Joo, M.-h. Kim, S.-J. Kang, M. Choi, S. Lee, and Y. Park, "Holographic deep learning for rapid optical screening of anthrax spores," *Science Advances*, vol. 3, p. e1700606, 08 2017.

[28] P. U. Stanford Vision Lab, Stanford University. (2016) Imagenet. [Online]. Available: http://image-net.org/

[29] M. Manoj krishna, M. Neelima, H. Mane, and V. Matcha, "Image classification using deep learning," *International Journal of Engineering and Technology*, vol. 7, p. 614, 03 2018.

[30] S. Saha, "A comprehensive guide to convolutional neural networks—the eli5 way," 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[31] M. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[32] T. Kavzoglu, "Object-oriented random forest for high resolution land cover mapping using quickbird-2 imagery," *Handbook of Neural Computation*, 2017.

[33] Thinkstock. (2018) What is deep learning and how will it change healthcare? [Online]. Available: https://healthitanalytics.com/features/what-is-deep-learning-and-how-will-it-change-healthcare

[34] B. Matheus and L. Lattari, "Convolutional neural networks for static and dynamic breast infrared imaging classification." pp. 174–181, 2018. [Online]. Available: https://www.researchgate.net/publication/330470461_Convolutional_Neural_Networks_for_Static_and_Dynamic_Breast_Infrared_Imaging_Classification

[35] A. I. M. T. M. R. Minhas, A. Javed and Y. B. Joo, "Shot classification of field sports videos using alexnet convolutional neural network," *Applied Sciences*, vol. 9, p. 483, 01 2019.

[36] W. F. M. Wagenaar, E. Okafor and M. Wiering, "Using deep convolutional neural networks to predict goal-scoring opportunities in soccer," 02 2017.

[37] V. Holman, "Sports Analytics Models - Convolutional Neural Networks," accessed 2019-07-26. [Online]. Available: https://www.agilesportsanalytics.com/sports-analytics-models-convolutional-neural-networks/

[38] G. R. Kotapalle and S. Kotni, "Security using image processing and deep convolutional neural networks," in *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, May 2018, pp. 1–6.

[39] N. Dahringer, "Electricity theft detection using machine learning," 08 2017.

[40] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, Aug 2003, pp. 958–963.

[41] K. Chellapilla, S. Puri, and P. Y. Simard, "High performance convolutional neural networks for document processing," 2006.

[42] K. L. S. Lai, L. Xu and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, pp. 2267–2273. [Online]. Available: http://dl.acm.org/citation.cfm?id=2886521.2886636

[43] S. Lawrence, C. L. Giles, Ah Chung Tsoi, and A. D. Back, “Face recognition: a convolutional neural-network approach,” *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, Jan 1997.

[44] R. Ranjan, S. Sankaranarayanan, C. D. Castillo, and R. Chellappa, “An all-in-one convolutional neural network for face analysis,” in *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, May 2017, pp. 17–24.

[45] R. Golemanova. (2019) 7 image recognition applications of the future. [Online]. Available: https://imagga.com/blog/7-image-recognition-uses-of-the-future/

[46] J. A. K. Muhammad and S. Baik, “Early fire detection using convolutional neural networks during surveillance for effective disaster management,” *Neurocomputing*, 12 2017.

[47] X. Gigandet, M. B. Cuadra, A. Pointet, L. Cammoun, R. Caloz, and J. . Thiran, “Region-based satellite image classification: method and validation,” in *IEEE International Conference on Image Processing 2005*, vol. 3, Sep. 2005, pp. III–832.

[48] P. Shrivastava. (2019) Challenges in deep learning. [Online]. Available: https://hackernoon.com/challenges-in-deep-learning-57bbf6e73bb

[49] Photokonnexion. Definition: Shadow. (2019, September 19th). [Online]. Available: https://www.photokonnexion.com/definition-shadow/

[50] J. Brownlee. (2016) Deep learning and artificial neural networks. [Online]. Available: https://machinelearningmastery.com/what-is-deep-learning/

[51] MATLAB. What Is Deep Learning?: How It Works, Techniques and Applications. (2019, august 29). [Online]. Available: https://www.mathworks.com/discovery/deep-learning.html

[52] S. Mahapatra. (2019) Why deep learning over traditional machine learning? [Online]. Available: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063

[53] MissingLink.ai. (2019) Neural networks for image recognition: Methods, best practices, applications - missinglink.ai. [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/neural-networks-image-recognition-methods-best-practices-applications/

[54] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. F. Feng, and M. Chen, "Medical image classification with convolutional neural network," *2014 13th International Conference on Control Automation Robotics and Vision, ICARCV 2014*, pp. 844–848, 03 2015.

[55] A. Rezvantalab, H. Safigholi, and S. Karimijeshni, "Dermatologist level dermoscopy skin cancer classification using different deep learning convolutional neural networks algorithms," 10 2018.

[56] H. Haenssle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. Hassen, L. Thomas, A. Enk, and L. Uhlmann, "Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists," *Annals of oncology : official journal of the European Society for Medical Oncology*, vol. 29, 05 2018.

[57] K. Zhang, Q. Wu, A. Liu, and X. Meng, "Can deep learning identify tomato leaf disease?" *Advances in Multimedia*, vol. 2018, pp. 1–10, 09 2018.

[58] J. Amara, B. Bouaziz, and A. Algergawy, "A deep learning-based approach for banana leaf diseases classification," pp. 79–88, 2017. [Online]. Available: http://btw2017.informatik.uni-stuttgart.de/slidesandpapers/E1-10/paper_web.pdf

[59] W. Hu, Y. Huang, L. Wei, F. Zhang, and H. Li, "Deep convolutional neural networks for hyperspectral image classification," *Journal of Sensors*, vol. 2015, pp. 1–12, 07 2015.

[60] H. Lee and H. Kwon, "Going deeper with contextual cnn for hyperspectral image classification," *IEEE Transactions on Image Processing*, vol. 26, pp. 1–1, 07 2017.

[61] F. S. P. Ltd. 7 APPLICATIONS OF CONVOLUTIONAL NEURAL NETWORKS. (2019, august 28). [Online]. Available: https://www.flatworldsolutions.com/data-science/articles/7-applications-of-convolutional-neural-networks.php

[62] H. Sharma. Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning. (2019, august 29). [Online]. Available: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d

[63] J. Bagnato, "¿cómo funcionan las convolutional neural networks? visión por ordenador," 2018. [Online]. Available: http://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

[64] missinglink. aii. Fully Connected Layers in Convolutional Neural Networks: The Complete Guide. (2019, august 29). [Online]. Available: https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/

[65] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[66] F. Huang, J. Ash, J. Langford, and R. Schapire, "Learning deep resnet blocks sequentially using boosting theory," 06 2017.

[67] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[68] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," 06 2016.

[69] A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks.” pp. 1097–1105, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[70] J. Han, L. Xu, M. M. Rafique, A. Butt, and S.-H. Lim, “A quantitative study of deep learning training on heterogeneous supercomputers,” 09 2019, pp. 1–12.

[71] B. J. (2019) How to configure the learning rate when training deep learning neural networks. [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[72] J. Brownlee. (2020) Understand the impact of learning rate on neural network performance. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-network

[73] Silva, “Implementing YOLO Algorithm for Real Time Object Detection on Embedded System,” November 2019.

Appendices

Appendix A

Algorithms Code

In this Section, the corresponding source codes for pre-processing the data set are presented. Also the source code for implement, train and deploy the different models on the framework caffe used in this work.

A.1 Caffe models

The original models train-val.prototxt as well as the solver.prototxt are available in the following repository. The modified models are available here.

A.2 Scripts

A.2.1 Brightness and Contrast Variation Script

```
import cv2

f= open("list1.txt","r")
f1=f.readlines()
for i in f1:
    image = cv2.imread(i)
    alpha = 1.5 # Contrast control (1.0-3.0)
    beta = 0 # Brightness control (0-100)
    adjusted = cv2.convertScaleAbs(image,alpha,beta)
    alpha = 0.5 # Contrast control (1.0-3.0)
    beta = 0 # Brightness control (0-100)
    adjusted2 = cv2.convertScaleAbs(image,alpha,beta)
    alpha = 2.5 # Contrast control (1.0-3.0)
    beta = 0 # Brightness control (0-100)
    adjusted3 = cv2.convertScaleAbs(image,alpha,beta)
    cv2.imwrite(i+'out.png', adjusted)
    cv2.imwrite(i+'out1.png', adjusted2)
    cv2.imwrite(i+'out2.png', adjusted3)
f.close()
```

A.2.2 Data Augmentation Script

```
#! /usr/bin/python

import numpy as np
import cv2
import os
import glob
import Augmentor

'''
Path TRAIN
'''
pathTrainLeafs='/DataFinal/train'
pathValLeafs='/DataFinal/validation'

```

```
def generateDataAugmented(pathSrc):
    p=Augmentor.Pipeline(pathSrc)
    p.rotate(probability=1,
    max_left_rotation=5,
    max_right_rotation=5)
    p.random_distortion(probability=1,
    grid_width=4, grid_height=4, magnitude=8)
    p.flip_left_right(probability=0.5)
    p.flip_top_bottom(probability=0.5)
    p.zoom_random(probability=0.5, percentage_area=0.8)
    p.flip_top_bottom(probability=0.5)
    p.sample(35000)

generateDataAugmented(pathTrainLeafs)
print ("Fin Aug train")
generateDataAugmented(pathValLeafs)
print ("Fin Aug val")
```

A.2.3 LMDB Creation Script

```

#!/usr/bin/env sh
set -e
EXAMPLE='/DataFinal'
DATA='/DataFinal'
TOOLS='/apps/caffe/1.0.0/build/tools/convert_imageset'
TRAIN_DATA_ROOT='DataFinal/train/output'
VAL_DATA_ROOT='DataFinal/validation/output'

```

```
RESIZE_HEIGHT=256
RESIZE_WIDTH=256
RESIZE=true

echo "Creating train lmdb..."

GLOG_logtostderr=0 $TOOLS \
    --shuffle \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    $TRAIN_DATA_ROOT \
    $DATA/train.txt \
    $EXAMPLE/leaf_train_lmdb

echo "Creating val lmdb..."

GLOG_logtostderr=0 $TOOLS/ \
   --shuffle \
  --resize_height=$RESIZE_HEIGHT \
 --resize_width=$RESIZE_WIDTH \
    $VAL_DATA_ROOT \
   $DATA/val.txt \
   $EXAMPLE/leaf_val_lmdb

echo "Done."
```

A.2.4 Quinde Script

```
#BSUB -e bench_n_fj.%J.err.log
```

```
#BSUB -o bench_n_fj.%J.out.log
#BSUB -J benchmark_n.job
#BSUB -cwd /home/gcaluna/tesis/ResNet101/test1
#BSUB -q normal
#BSUB -n 4

module purge
module load cuda/8.0.61
module load boost/1.53.0
module load gcc/5.4.0
module load hdf5/1.10.0
module load caffe/1.0.0

cd /home/gcaluna/tesis/ResNet101/

/apps/caffe/1.0.0/build/tools/caffe train --solver
/home/gcaluna/tesis/ResNet101/solver.prototxt -gpu 1 2>&1 | tee
/home/gcaluna/tesis/ResNet101/test1/model_ResNet101_try4.log
```

A.2.5 Deployment Script

```
#! /usr/bin/python3
import sys
import os
import glob
import cv2
import caffe
import numpy as np
from caffe.proto import caffe_pb2
```

```
caffe.set_mode_cpu()
'''
Reading mean image, caffe model and its weights
'''
#Read mean image
mean_blob = caffe_pb2.BlobProto()
with open('/mean.binaryproto','rb') as f:
    mean_blob.ParseFromString(f.read())
    mean_array = np.asarray(mean_blob.data, dtype=np.float32).
    reshape(
    (mean_blob.channels, mean_blob.height, mean_blob.width))
#Read model architecture and trained model's weights
net = caffe.Net('/deploy.prototxt',
                '/CNN.caffemodel',
                caffe.TEST)
#Define image transformers
transformer = caffe.io.Transformer({'data':
net.blobs['data'].data.shape})
transformer.set_mean('data', mean_array)
transformer.set_transpose('data', (2,0,1))
'''
Making predicitions
'''
#Reading image paths
test_img_paths = [img_path for img_path in glob.glob("/*JPG")]
#Making predictions
test_ids = []
correct_label_ids = []
preds = []
```

```
for img_path in test_img_paths:
    print ('reading ' + img_path + '...')
    img = cv2.imread(img_path)
    net.blobs['data'].data[] = transformer.preprocess('data',img)
    out = net.forward()
    pred_probas = out['prob']
    print ([img_path.split('/')[-1][:-4]])
    test_ids = test_ids + [img_path.split('/')[-1][:-4]]
    preds = preds + [pred_probas.argmax()]
    path, extension=img_path.split('.')
    correct_label_ids=correct_label_ids + [path[-1][-1]]
    print (img_path)
    print ('Label: '+
    getLabelTextByNumber(int(path[-1][-1])))
    print ('Prediction: ' +
    getLabelTextByNumber(pred_probas.argmax()))
    print ('-------')
with open("/prediction.csv","w") as f:
    f.write("id,label, prediction\n")
    for i in range(len(test_ids)):
        f.write(str(test_ids[i])+","+
        str(correct_label_ids[i])+","+
        str(preds[i])+"\n")
f.close()
```

All the source code of the scripts used to perform the data pre-processing and to implement, train and deploy the models are available here.

A.3 Graphic Interface

The source code of the graphic interface is available in the following site.