DISSERTAÇÃO DE MESTRADO

A COMPUTER VISION SYSTEM FOR RECOGNIZING PLANT SPECIES IN THE WILD USING CONVOLUTIONAL NEURAL NETWORKS

René Octavio Queiroz Dias

Brasília, julho de 2017

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

MASTER'S THESIS

A COMPUTER VISION SYSTEM FOR RECOGNIZING PLANT SPECIES IN THE WILD USING CONVOLUTIONAL NEURAL NETWORKS

René Octavio Queiroz Dias

Master's Thesis submitted to the Department of Mechanical Engineering as a partial requisite to obtain a Master's degree in Mechatronic Systems

Examining Committee

FICHA CATALOGRÁFICA

DIAS, RENÉ OCTAVIO QUEIROZ
A COMPUTER VISION SYSTEM FOR RECOGNIZING PLANT SPECIES IN THE WILD USING CONVOLUTIONAL NEURAL NETWORKS [Distrito Federal] 2017.
xvi, 64 p., 210 x 297 mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2017).
Dissertação de Mestrado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Mecânica

1. Aprendizagem de Máquina	2. Inteligência Artificial
3. Visão Computacional	4. Classificação de Plantas
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

DIAS, R.O.Q. (2017). *A COMPUTER VISION SYSTEM FOR RECOGNIZING PLANT SPECIES IN THE WILD USING CONVOLUTIONAL NEURAL NETWORKS*. Dissertação de Mestrado, Publicação ENM.DM-126/2017, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 64 p.

CESSÃO DE DIREITOS

AUTOR: René Octavio Queiroz Dias
TÍTULO: A COMPUTER VISION SYSTEM FOR RECOGNIZING PLANT SPECIES IN THE WILD USING CONVOLUTIONAL NEURAL NETWORKS.
GRAU: Mestre em Sistemas Mecatrônicos ANO: 2017

René Octavio Queiroz Dias
Depto. de Engenharia Mecânica (ENM) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedication

To my parents, Rubení and René, and to my sister, Anna.

René Octavio Queiroz Dias

Acknowledgements

I would like to thank my family for their incredible support; they always have been encouraging me to keep going forward with this project, even when I was in the brink of giving up. A special thanks to my mother and sister, whom, in their spare time, went with me to the field, and helped with the pictures of the plants.

I am eternally grateful to my grand-father, Joviniano, whom months before passing way, advised me to keep studying instead of telling me how to pass through the roughness of life. This really impressed me, especially for a person with rural background, little instruction, and even incapable to tell what I was studying.

Thank you, professor Díbio, you offered me new perspectives in this project, and helped me follow ahead when I got stuck or I when I had many ideas and had to select the feasible ones.

I also want to express my gratitude to the professors and friends at Universidade Federal de Uberlândia, Bucknell University and Universidade de Brasília, whom laid the cornerstone which upon I built this project.

René Octavio Queiroz Dias

“Eppur si muove!",

attributed to **Galileo Galilei**.

ABSTRACT

Classifying plant species has been a recurrent topic in the Computer Vision community. Visually, plants present a high level of variability, mostly because of seasonal effects, age and background. Early classification systems had difficulties to deal with this variability and early databases relied on simple images, using dismembered parts of the plants, such as leaves and flowers, and a distinctive background (usually white).

With the advent of Deep Neural Networks, which proved to be very competitive as a general-purpose classifier, we aim to assess them with a more specific-purpose database, which can be further strained by trying to classify similar plant species in some very different poses.

We created a new database that focus on how the common user takes plant pictures. This database, named *Plantas*, is meant to be highly unconstrained. Initially, it contains 50 common different species and cultivars used in gardening worldwide, and more than 33,000 images. These images were taken on site and download from the Internet.

Then, we train this database with the latest state of the art techniques, such as Encoding Methods and Deep Neural Networks. We further explore neural networks by testing some recent activation functions and also fine-tuning.

RESUMO ESTENDIDO

Classificação de plantas tem sido um problema recorrente na comunidade de Visão Computacional. Visualmente, as plantas apresentam uma variabilidade muito grande, decorrente principalmente de efeitos sazonais, idade e fundos. Sistemas de classificação mais antigos tinham problemas para lidar com estas variações e seus bancos de dados usavam imagens mais simples com apenas partes desmembradas de plantas (como folhas e flores) e fundo branco.

Com o advento das Redes Neurais Profundas, que demostraram ser bastante competitivas como classificadores de propósito geral, o objetivo é testá-las com um banco de dados de propósito mais específico, que podem tencionar mais estes classificadores tentando classificar espécies de plantas similares em poses bastante diferentes.

Construiu-se um banco de dados que é focado em como o usuário comum tira retratos de plantas. Este novo banco de dados, chamado *Plantas*, foi feito para ter poucas restrições. Inicialmente, há 50 espécies diferentes que são usados comumente em jardinagem, e há mais de 33.000 imagens. Estas fotos foram tiradas *in loco* e da Internet.

Depois, treinou-se com técnicas recentes do estado da arte, como os Métodos de Codificação e Redes Neurais Profundas. Nos Métodos de Codificação, são usados três codificadores: Saco de Palavras Visuais (BoVW), Vetores Fisher (FV) e Vetores de Descritores Linearmente Agregados (VLAD).

Nos Métodos de Codificação, há duas fases: uma aprendizagem sem-supervisão e em seguida uma supervisionada. Em todos os métodos, o processo é parecido. Na fase sem-supervisão, obtêm-se os descritores SIFT, retira-se uma amostra destes descritores, faz uma aprendizagem da projeção da Análise de Componentes Principais e usa-se *k*-médias para agregar estas características em k grupos, que são o número de palavras. Aqui se separa o treinamento de BoVW e VLAD dos Vetores Fisher. Para os primeiros, cria-se uma árvore *k*-d para facilitar o posterior processo de pesquisa. Para os Vetores Fisher, usa-se os grupos como inicialização dos Modelos de Mistura de Distribuições Normais.

Na fase de aprendizagem supervisionada, passa-se uma imagem pelos processos de obtenção dos descritores SIFT, amostragem e PCA. Então, para cada característica de uma imagem, pesquisa-se o grupo a qual pertencente. Para BoVW, obtém-se um histograma que conta cada palavra da imagem que tem o equivalente no dicionário. Para VLAD, obtém-se o desvio à média destas palavras, e com Vetores Fisher, além do desvio à média, calcula-se o desvio à covariância. Estes, representam os descritores finais que são posteriormente treinados com uma Máquina de Vetores de Suporte Linear (Linear-SVM).

Nas redes neurais, são treinadas diferentes arquiteturas recentes como AlexNet, CaffeNet, GoogLeNet e ResNet. Elas contêm técnicas que exploram a estrutura espacial das imagens, como

as camadas de convoluções, e usam técnicas de regularização que evitam sobreajuste — que era algo especialmente comum em redes com muitos parâmetros — como *Dropout* e Normalização em Lotes. Também foi a primeira vez em que se usou uma função de ativação que não sofre problemas de saturação, a Unidade Linear Retificada (ReLU) que tomou o lugar de Sigmóides e Tangentes Hiperbólicas.

Usando estas arquiteturas, faz-se experimentos para saber como elas respondem ao novo banco de dados, e quais são as melhores especificações para obter-se a melhor acurácia e quais as razões que uma escolha é melhor que a outra.

Nestes experimentos, funções de ativações mais recentes como a Unidade Linear Retificada Parametrizada (PReLU) e a Unidade Linear Exponencial (ELU) foram testadas. Também, usa-se técnicas de ajuste fino em que se reutiliza parâmetros de uma rede treinada para um certo banco de dados em outro, também conhecido como transferência de conhecimento.

SUMMARY

FIGURES LIST

TABLES LIST

SYMBOLS LIST

Latin Symbols

$\mathbf{B}$	A mini-batch
$L(\bullet)$	Loss Function
$p(\bullet)$	Probability distribution
$\hat{p}(\bullet)$	Empirical distribution of the data
$P(\bullet)$	Predicted probabilities
$\mathbf{x}$	An image
$\mathbf{X}$	A set of images and their ground-truth label
y	Ground-Truth
$\mathbf{y}$	A ground-truth one-hot vector
$\hat{\mathbf{y}}$	The output of a layer
$\mathbf{z}$	A descriptor
$\mathbf{Z}$	A descriptor matrix

Greek Symbols

$\nabla(\bullet)$	Differential Operator
π	Prior Probability
μ	Mean
σ^2	Variance
$\mathbf{\Sigma}$	Covariance Matrix
$\boldsymbol{\theta}$	Parameters

Dimensionless Groups

e	Euler’s number

Subscripts

max	Maximum
min	Minimum

Superscripts

$-$	Mean value

Initials

AESA	Nearest Neighbor Approximating and Eliminating Search Algorithm
ANN	Artificial Neural Network
BoVW	Bag of Visual Words
BP	Backwards Propagation of errors
B/W	Black and White image
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DB	Database
ELU	Exponential Linear Unit
EM	Expectation-maximization algorithm
FNN	Feed-forward Neural Network
FV	Fisher Vector
IDSC	Inner Distance Shape Context
i.i.d.	Independent and Identically Distributed
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HEX	Hierarchical and Exclusion Graph
HoCS	Histogram of Curvature over Scale
HOG	Histogram of Oriented Gradients
HSV	Hue-Saturation-Value Color Space
MR8	Maximum Response 8 Filter Bank
MRF	Markov Random Field
PCA	Principal Component Analysis
PReLU	Parametric Rectified Linear Unit
RBF	Radial Basis Function
RGB	Red-Green-Blue Color Space
ReLU	Rectified Linear Unit
SIFT	Scale-Invariant Feature Transform
SDCA	Stochastic Dual Coordinate Ascent
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VGG	Visual Geometry Group (University of Oxford)
VLAD	Vector of Linearly Aggregated Descriptors
VQ	Vector Quantization

1 INTRODUCTION

The Computer Vision community have been researching classification of plant species in the last decades. The creation of a classification system needs two components: data and model. They are intrinsically bounded and building a complex database pushes the development of models.

Past models needed discriminative images with low ambiguity. Tree leaves and plant flowers dominated these image databases. The first for their morphological features (each tree have leaves with distinctive shapes), and the latter for their color, texture, and also morphological structures. Albeit the variation of flowers shape within a species challenged these models (SÖDERKVIST, 2001; WU et al., 2007; NILSBACK; ZISSERMAN, 2006; NILSBACK; ZISSERMAN, 2008; CERUTTI et al., 2011; LAGA et al., 2012; KUMAR et al., 2012).

These features were extracted using handcrafted descriptors, and later replaced by automatic features detectors and extractors, such as Scale-Invariant Feature Transform (SIFT) and Histogram of Oriented Gradients (HOG) (SÖDERKVIST, 2001; WU et al., 2007; NILSBACK; ZISSERMAN, 2006; NILSBACK; ZISSERMAN, 2008; CERUTTI et al., 2011; LAGA et al., 2012; KUMAR et al., 2012).

In recent years, with the advent of deep neural networks, we can train very complex and diverse image databases. Krizhevsky, Sutskever & Hinton (2012), for instance, trained their network on the 2012 ImageNet database (with 1,000 different classes such as dogs, boats, and trees) breaking the accuracy plateau of the past years.

While models have evolved, current plant database are not fully exploiting these models capabilities, with the exception of PlantCLEF (GOËAU; BONNET; JOLY, 2015). PlantCLEF have a large number of species. However, it focuses on trees, ferns, and herbs only; has a small number of samples for some species; and favors a parted plant classification (such as classifying images of leaves, branches, fruits, etc.).

Our primary goal is to devise a system that can classify plants on site in an unconstrained way, thus closing this gap of model and data.

First, we construct a new database, called *Plantas*, in which the plant pictures are taken on site, with no white background and no dismemberment of plant parts, the only limitation being that the plant must occupy the most part of the picture. In most cases, leaves cover the most part of the pictures; but flowers, stems or any part of the plant might appear. We increased the diversity of plants, including succulents, trees, vines, and ferns. And, we downloaded several images of the selected species and manually curated them; adding variability (DIAS; BORGES, 2016).

Then, we train models using two separate approaches. In the first approach, we use the prior state of the art in recognition (VEDALDI; FULKERSON, 2008) to serve as a baseline classifier. These models are based on the construction of a dictionary of visual words — using

Scale-Invariant Feature Transform (SIFT) and *k*-means — followed by training using a Linear Support Vector Machine (SVM) and one of the following methods: Bag of Visual Words (BoVW), Fisher Vector (FV) and Vector of Locally Aggregated Descriptors (VLAD) (DIAS; BORGES, 2016; PERRONNIN; DANCE, 2007; PERRONNIN; SÁNCHEZ; MENSINK, 2010; JÉGOU et al., 2010).

In the second approach, we aim to shed the light on the capability of deep neural networks to train more fine-grained database such as plant species, which some species are visually very similar. These networks usually has the following architecture: we push an image through the network (forward pass), which passes through different layers (such as convolutional and fully-connected) till it reaches the loss layer (which tells how off the classification has been compared to the ground truth), then the solver uses this information and updates the network's weights by doing a backward pass.

We train the following deep network architectures: AlexNet, CaffeNet, GoogLeNet, Inception, and ResNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; JIA et al., 2014; SZEGEDY et al., 2014; IOFFE; SZEGEDY, 2015; HE et al., 2016). Then, we further explore these architectures by changing the common activation function Rectified Linear Unit (ReLU) to the Parametric Linear Unit (PReLU) and Exponential Linear Unit (ELU) (GLOROT; BORDES; BENGIO, 2011; HE et al., 2015; CLEVERT; UNTERTHINER; HOCHREITER, 2015); and assess how fine-tuning — the use of weights trained in a different database — affects accuracy of these architectures trained on our database (DIAS; BORGES, 2016).

The first contribution is the *Plantas* database, aimed to push the frontier of plant classification. The second is the construction of a system that classify plants on site, which is derived by assessing different recognition techniques.

In chapter 2, we describe the related works, especially about plant database construction, prior classification systems and recent evolution of techniques in neural networks. In chapter 3, we introduce our *Plantas* database. In chapter 4, we present the Encoding Methods mathematical modeling. In chapter 5, we do the same for neural networks. In chapter 6, we describe our experiments and in chapter 7 we present our results.

2 RELATED WORKS

This chapter is divided into two parts: Plant Classification Systems and Neural Networks. In Plant Classification systems, we revise how systems of visual recognition of plant species have been built. Most of these works are divided into three pieces: the construction of the image database, segmentation, and systems of feature extraction.

In the second part, we revise Neural Network focused on the recent field of Deep Learning, which is centered on Convolutional Layers. We show how these deep networks evolved in time, and what are the new techniques involved in their success, such as Rectified Linear Unit, Dropout, and Batch Normalization.

2.1 PLANT CLASSIFICATION SYSTEMS

2.1.1 Databases

Söderkvist (2001) created a database consisting of 1,125 leaf images of 15 Swedish trees; averaging 75 samples per species. Scanners, adjustments, environment, and positioning varied. These leaves, cleaned and complete, were scanned in a laboratory.

Nilsback & Zisserman (2006, 2008) created a new database, but for flowers instead. First, with 17 species and 80 images each; later increased to 103 species with a total of 8189 images. They added difficulties by: choosing similar species; using flowers because they are non-rigid objects; increasing intra-class variation and inter-class similarity; and varying viewpoint, scale and illumination. The authors collected the images from the Internet and also on site; intentionally selecting images hard to discriminate alone by color or shape, for example. However, the background is usually greenish and flowers have very distinctive patterns and colors, which helped to isolate the flower alone in the picture during segmentation.

Wu et al. (2007) created a database called *Flavia* that consists mostly of trees and some plants with easily detachable leaves from China. It contains 1,800 clean and complete leaves of 33 species, averaging 54 leaves per species. A white background was used in all pictures, and it lacks further information on database construction.

Belhumeur et al. (2008) built a database of tree, shrubs and some vascular plant leaves collected by botanists in U.S. Northeast. The database is composed of three sets: The Flora of Plummers Islands with 5,014 samples of 157 species of vascular plants; The Woody Plants of Baltimore-Washington D.C. with 7,481 samples of 245 tree and shrubs; and The Trees of Central Park in New York with 4,320 samples of 144 trees. The collected leaves were flattened by pressing, and photographed with a ruler and a color chart, under correct luminosity in a laboratory. The ruler and color chart were then removed by an automated process before use.

Continuing the previous work, Kumar et al. (2012), expanded the database, called now LeafSnap. It contains 23,147 laboratory images and new 7,719 field images of 185 tree species of the Northeastern United States, averaging 185 samples per species. For the field images, they dismembered the leaves from the trees and used a white background, taking the pictures with common digital cameras.

PlantCLEF (GOËAU; BONNET; JOLY, 2015) was built of images from the social network Tela Botanica, composed by botanists that classify plant pictures sent by users. It contains 113,205 pictures of 1,000 species of herbs, trees, and ferns commonly found in France; and it is divided into 7 sets: LeafScan, Leaf, Flower, Fruit, Stem, Entire Plant, and Branch. All sets have images from natural environment, except LeafScan, which leaf images are taken with a white background. Flowers and leaves are the most populated sets. The diversity of users in different places, taking pictures in several periods of year, increases the variability compared to previous database.

2.1.2 Segmentation

In earlier classification systems, we needed to segment the leaf or the flower from the background. Most segmentation techniques relied on the image having a distinct background, which aids selecting the region of interest.

Back in 2001, when digital cameras were a rarity, scanner was a common device to digitize images. Söderkvist (2001) required users of the system to take a leaf sample and put it inside a scanner, which have white background. The image was segmented in black and white using an empirically selected threshold value; and the leaf was the biggest body, discovered by counting pixels. The holes were removed by selecting objects without boundaries with the background, images were cropped to fit the leaf, and the border was sampled in point.

Nilsback & Zisserman (2006, 2008) collected some images from the Internet, which are more cluttered and possess a more greenish background. They created two RGB distribution for the foreground and background by labeling a subset of pixels of an image — a part of the plant and a part of the greenery. Then, they binary segmented them by using contrast dependent prior-MRF cost function of Boykov and Jolly (BOYKOV; JOLLY, 2001).

Wu et al. (2007) transformed RGB images with white background to grayscale and created a RGB histogram of all pictures combined to select the threshold value that would separate the leaves from the background. The separated images, which became black and white, were smoothed by a noise filter and the margin of the leaves was obtained by applying a Laplacian filter.

Belhumeur et al. (2008) transformed the images from the RGB to the HSV color-space. Because the light in a forest have a greenish hue, the white background appears green, thus, hue was discarded. The pixels with their saturation and brightness values were separated into two clusters: background and foreground. They initialize with k-means clustering, then perform a real-time Expectation-Maximization (EM) on 5% of the pixels and classify the remaining pixels according to the two calculated Gaussian distributions.

Kumar et al. (2012) follow a similar pattern, however instead of initializing with *k*-means, they represent each pixel as a probability distribution modeled as a sum of two Gaussian distributions, with two different means but sharing the same covariance, which increases the speed of the EM algorithm and permits the use of 25% of the pixels. After segmentation, they remove any near-border remaining body, and eliminate the stem searching for an elongated body in the picture.

Cerutti et al. (2011) create a Parametric Active Polygon model according to a botanic classification of leaves (Lanceolate, Ovate, Oblong, etc.). These polygon shapes have points and parametric settings trained to match the shape of the leaves, and are intended to segment a leaf in a cluttered image, with other leaves or shadows. The training is initiated using the centralized pixels — the images must be centered and vertically oriented — and is divided in two parts: leaf color and shape. The color is modelled into shaded and lighted parts of leaves using two GMMs, and the objective function, using color distances, tries to maximize the number of pixels while penalizing pixels with distant colors. Shape training is initialized by ten hand-crafted initial shapes that are left to evolve using *k*-means clustering.

Laga et al. (2012) use an elastic metric for the shape of leaves. The outer boundary, represented as a Square Root Velocity function, is guaranteed to be scale invariant. The statistical analysis of these shapes representations, using a Riemannian structure, outputs the mean shape and the principal direction of variation within a species. Thus, they could attest the similarity between shapes without a correct positioning, a correspondence between shapes, and the optimal deformation that aligns shapes. However, while being more flexible than the previous works, they used the *Flavia* database (WU et al., 2007), which have uncluttered images.

2.1.3 Feature Descriptors and Classification

Earlier systems needed features to train and those were obtained by using handcrafted descriptors — which extracts geometric, color, and texture features from the images. Later, more automated extractors, such as SIFT, became more common.

Söderkvist (2001) extracted features from the shape of leaves. The descriptors required a B/W image or the sample points of the leaf boundary, and they were: Area, Circularity, Flusser-moments, Hu-moments, Eccentricity, Curvature Scale Space, and Incremental Circle Transform. These features were trained on an Artificial Neural Network (ANN) with three layers: input layer, 100 nodes hidden layer, and a 15 nodes layer corresponding to each tree species.

Nilsback & Zisserman (2006) extracted features from their 17 Flower Species database using a Combined Histogram of Visual Words. They built three vocabularies: for shape, color, and texture. The first through SIFT descriptors with regular grid; empirically optimizing the values of grid spacing, support regions, and clusters. The second by changing the color-space to HSV — which is better for illumination invariability — and clustering the images pixels using *k*-means. The latter by clustering and building a histogram of features from a MR8 filter bank (VARMA; ZISSERMAN, 2002). A linear combination of these vocabularies resulted in trainable system, and the authors obtained the weights by setting one of the weights to one, while varying the others, which they acknowledge as a suboptimal solution, however less computational expensive than actually training the weights.

Continuing their work, Nilsback & Zisserman (2008) changed the approach to shape and texture, which now have local and global features. For local shape, texture, and boundary, they use SIFT descriptors of the image foreground region for the first two, and foreground boundary for the latter; and capture the global spatial properties of the flower by extracting features using HOG on the entire flower. Finally, they train an SVM with multiple Mercer kernels multiplied by a weight, which learns in a one-vs-rest fashion.

Wu et al. (2007) derived 12 digital morphological features based on 5 geometric features of the leaf. One of the geometric feature must be entered by the user, the others are determined automatically from the leaf image. They calculate these 12 features that are further reduced to 5 using Principal Component Analysis (PCA). Then, the authors use an Artificial Neural Network using RBF as an activation function, which is composed by 3 three layers: the first with 5 nodes for each feature, 1800 nodes hidden layer, and 32 outputs. Although having a lower accuracy than the state of the art at the time, it was faster.

Belhumeur et al. (2008) use a shape matching system using the Inner Distance Shape Context (IDSC). The IDSC samples the boundary points and build a 2D Histogram for each point, which represents the distance and angle of one sample point to all others, along a path that passes only inside the leaf, calculated using all pairs shortest path algorithm, done off-line. The querying calculates the χ^2 distance from images sample points to all others in the others shape, and the matching is calculated using Nearest Neighbor AESA algorithm.

Kumar et al. (2012) replace the IDSC algorithm with the Histogram of Curvature over Scale (HoCS). HoCS calculates the area or arc-length inside a circle of some radius (or scale) for each point in the leaf boundary. The curvature image — in the shape of number of points (x), number of scale (y), and value of area or arc-length (z) — have each of their row (y-axis) taken as a histogram. A nearest neighbors identifies the species by using the HoCS of the input image compared to all others of the training set.

2.2 CONVOLUTIONAL NEURAL NETWORKS

2.2.1 Architectures

Most of modern image recognition neural networks relies on convolutional layers introduced by LeCun et al. (1989). Convolutional neural networks take advantage of some properties of natural signals such as: local connections, shared weights, pooling and use of many layers (LECUN; BENGIO; HINTON, 2015). This cause a reduction of parameters and connections; improving computational time and facilitating training (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Krizhevsky, Sutskever & Hinton (2012) created the first competitive convolutional neural network trained in a large-scale image database. In order to train such a large network, they used GPUs optimized for convolutions. The architecture of this network combined some novel and unusual features such as: rectified linear units, local response normalization, max-pooling, data augmentation, and Dropout.

Averaging on multiple models was the new technique used by Clarifai, the 2013 ImageNet Challenge image classification winner. They also used some insights provided by Zeiler & Fergus (2013), in which de-convolutional neural network permitted to explore internal representations. Thus, they modified the filter sizes and stride of the Krizhevsky, Sutskever & Hinton (2012) convolutional network.

GoogLeNet emerged as the winner of the 2014 ImageNet Challenge image classification. They worked for an increase in depth and at the same time addressing the problems that come with it: over-fitting and computational resources. The main idea of their new Inception architecture is to find out how an optimal local sparse structure in network can be approximated by dense components (SZEGEDY et al., 2014). They also adopted a more aggressive data augmentation and the removal of fully connected layers; favoring average pooling.

Simonyan & Zisserman (2014) major contribution was to increase depth using only small convolution filters and still having competitive results. Their convolutional layers are composed only by filters of size 3x3. Scale jittering was used for data augmentation and local response normalization was removed.

Ioffe & Szegedy (2015) address the internal co-variate shift — which is input layer distribution change during training — by normalizing the inputs for each training mini-batch. This serve also as a regularizer, and Dropout became optional. Some of the changes in the network were: the increase of learning rate, removal of dropout, acceleration of the learning decay rate, and reduction of the photo-metric distortions.

Very deep networks suffer from a degradation problem: while the depth increases, accuracy gets stalled and then decreases. This is not an over-fitting problem, and adding more layers leads to a higher training error. He et al. (2016) introduced the *deep residual framework* that shortcut connections of some layers (e.g. layer 3 has the result of layer 2 and layer 1 as input, instead of having only layer 2 as input), helping back-propagating the signal through several layers; reducing training error. This was the first time that more than 100 layers have been successfully trained and obtained the best result in the 2015 ImageNet Challenge.

2.2.2 Activation Functions

In the early years, logistic sigmoid and hyperbolic tangent functions were largely used (LECUN; BENGIO; HINTON, 2015). Logistic sigmoid were used more for biological modeling of neurons and hyperbolic tangent for working better with multi-layer neural network. However, rectified linear units (ReLUs) are more biologically plausible and worked better for deep networks (GLOROT;

BORDES; BENGIO, 2011).

Some experiments indicated that traditional deep network was hard to train (SCHMIDHUBER, 2015). Typical deep networks suffer from vanishing or exploding gradients while using standard activation functions, because cumulative back-propagation errors would shrink or explode. This became known as the Fundamental Deep Learning Problem (HOCHREITER, 1991; SCHMIDHUBER, 2015).

The substitution of the traditional activation functions for rectified linear units (ReLUs) alleviated the problem because of their linearity (GLOROT; BORDES; BENGIO, 2011). The saturating properties of logistic sigmoid and hyperbolic tangent prevent them to train fast, so the non-saturating ReLUs train much faster and without them would be impossible to train deep networks (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Thus, pre-training deep supervised network became unnecessary (LECUN; BENGIO; HINTON, 2015). Also, as computational power increased, errors could be propagated to inner layers within reasonable time, which mitigated the Fundamental Problem effect (SCHMIDHUBER, 2015).

The Batch Normalization technique — which ensures that the distribution of non-linearity inputs remains more stable — makes possible the use of saturating non-linearities as the network will be more unlikely to get trapped in a saturated mode. However, the accuracy of ReLU-based network is still better than the sigmoid-based network (IOFFE; SZEGEDY, 2015).

By studying the properties of rectifiers, He et al. (2015) proposed a new generalization of ReLU, called Parametric Rectified Linear Unit (PReLU). It permits a slope for negative input, and the slope trains jointly with the model. This way, zero gradients was avoided and accuracy increased for challenging tasks. The increase in the number of parameters is small, which avoids the risk of over-fitting.

The Exponential Linear Unit (ELU) proposes an exponential response to a negative input with a hyper-parameter determining the point of saturation. This pushes the mean activation to zero and the gradient closer to the natural gradient, mitigating the problem of *bias shift*, which leads the unit to oscillate; hampering learning. Because Batch Normalization addressed a similar problem, networks with ELUs are better without it. ELUs lead to higher accuracy in earlier epochs — especially with network that have more than 5 layers — although training in slightly more time for the same number of epochs compared to ReLU (CLEVERT; UNTERTHINER; HOCHREITER, 2015).

2.2.3 Error Optimization

The training in a neural network occurs when we minimize the error on the training set. For example, given some images, we want the network to correctly classify most of those images, changing the weight values accordingly. In a supervised learning, we know beforehand an image and its class, and after the image has been classified by the network, we analyze if it was correctly classified. A loss (also known as cost or objective) function calculates the error between the predicted class and the given class.

Back-propagation (backward propagation of errors) derives the loss function with respect to the output of each neuron, starting from the last layers and going into the inner layers, as practical application of the chain rule (LECUN; BENGIO; HINTON, 2015).

Gradient descent calculates the minimum by selecting a random point in the function, analyzing the slope, and choosing the way down. The iteration stops when a local minimum is found and the weights are updated. Throughout the 1990s, it was largely believed that gradient descend would get stuck in poor local minimum (LECUN; BENGIO; HINTON, 2015). However, getting in poor local minimum is a problem only for small networks and unimportant for larger networks (CHOROMANSKA et al., 2014).

Stochastic gradient descent (SGD) trains most of deep networks, because it minimizes the loss of a network using mini-batches of images to approximate the gradient of the loss function, and this is faster than traditional gradient descent. Also, it is impossible to load massive database into memory, and we can train using only chunks of it. The downside is that hyper-parameters must be careful selected.

2.2.4 Classifiers

Softmax (a multinomial logistic regression) classifier is used by most recent networks (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; ZEILER; FERGUS, 2013; SZEGEDY et al., 2014; SIMONYAN; ZISSERMAN, 2014; IOFFE; SZEGEDY, 2015). It outputs a normalized class probabilities — class score sum to one — that leads to a more meaningful interpretation on how "likely" an image belongs to a certain class. Softmax classifier uses a cross-entropy as a loss function.

Linear Multi-class Support Vector Machine (SVM) classifier works on the basis that given a margin, the score of the classified class must be at least this margin ahead of other classes. It uses a hinge loss as a cost function. In practice, however, we use more the squared hinge loss (L^2-SVM) because it is differentiable. Tang (2013) claims that L^2-SVM can perform better than Softmax classifier because of the use of hinge loss instead of cross-entropy loss; however, this was not tested on large-scale image database, such as PASCAL or ImageNet. Zeiler & Fergus (2013) tested both SVM and a Softmax classifier for the Caltech datasets, using an ImageNet pre-trained network, and the results were very similar for both SVM and Softmax while using networks of the same depth.

As most of all current approaches uses multi-class classification or binary classification, Deng et al. (2014) propose a new method of classification to capture the complexity of semantic labels. Because of the multi-class classification nature of assuming mutually exclusive labels and of binary classification accepting overlapping labels, this new method (HEX graphs) clearly defines a hierarchical and exclusive labels, accepting mutual exclusion relationship, overlapping and subsumption. HEX had better results using a modified Krizhevsky, Sutskever & Hinton (2012) network than Softmax, however Softmax is still competitive.

2.2.5 Depth

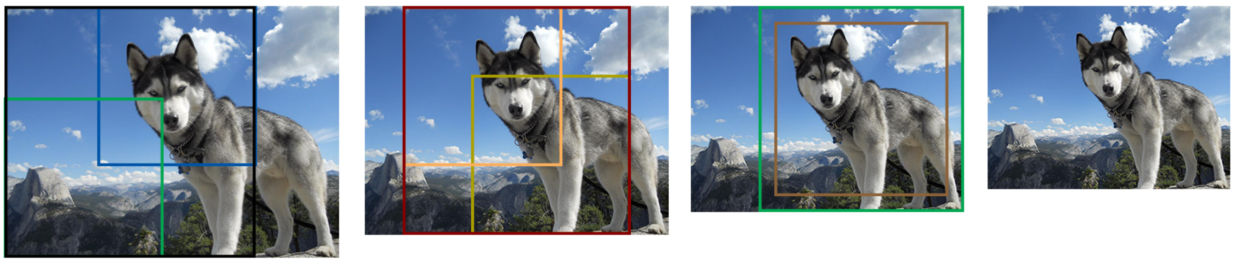
Increasing depth aids a network to discriminate better and captures a higher level of abstraction. Krizhevsky, Sutskever & Hinton (2012) created a network of depth 8 —5 convolutional and 3 fully connected layers— and removing any of the convolutional layer would lead to a increase in error rate. Zeiler & Fergus (2013) experimented removing and modifying some of these layers and discovered that removing the fully connected layers would not impact the model much and removing the two middle convolutional layers also made a small difference. However, removing both the two middle convolutional layers and the two fully connected layers increased the error by a significant margin. Changing the size of fully connected layers had a little impact, and increasing the size of convolutional layers improved the results, but the model over-fit.

Being tempted to increase depth and at the same time sparing some computational resources, GoogLeNet moved from a fully connected to a sparsely connected architecture and also implemented dimension reduction modules. These improvements made possible the expansion of the network in depth and breadth without incurring in a large increase in computational cost (SZEGEDY et al., 2014). On the other hand, the VGG network increased depth but using only small 3x3 convolutions as a way to avoid an increase of computational resources. They observed that error rate diminished with increasing depth, however a limit was found as their 16-deep and 19-deep networks scored the same (SIMONYAN; ZISSERMAN, 2014).

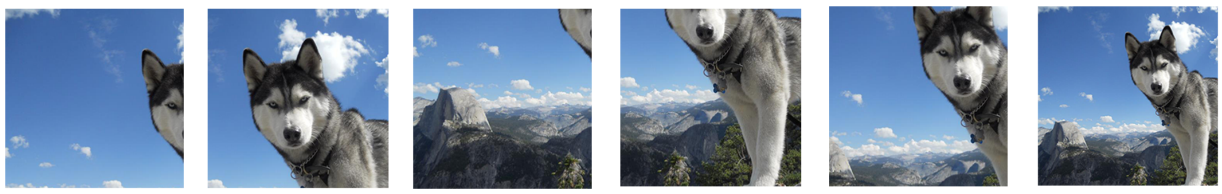
2.2.6 Data Augmentation

Krizhevsky, Sutskever & Hinton (2012) enlarged the database using label-preserving transformation. First, it generated translation and horizontal reflection of images. At training, it was extracted 224x224 patches of a 256x256 image, and their horizontal reflections as well. For evaluating an image, the network extract five 224x224 patches (corners and center) and their reflections for a total of ten patches. And the final prediction is the average of predictions of each patch made by a Softmax layer. The intensities of the RGB channels of training images was altered, making object identity invariant to changes in intensity and color of illumination.

Szegedy et al. (2014) are unclear in defining what was done in the GoogLeNet network, as they tested several ways to augment data for training. However, some directives — some of them inspired by the work of Howard (2013) — was given, such as: sampling different patch sizes with different aspect ratio, use of photo-metric distortion, and random interpolation methods for re-sizing. For testing, the image was re-sized to four scales, the shortest size being 256, 288, 320 and 352. For each of these images, three squares were taken (left, center, right). In these squares, a 224x224 cropping was done in each of the four corners and at the center. The square image was also re-sized to 224x224. A horizontal and vertical flip was done in these resulting cropped images. Fig. 2.1 shows the breakdown of this process.



(a) Scaling the shortest size of image down to 256, 288, 320 and 352 and then selecting the squares. Inside the square is shown the four corners and center 244x244 cropping.



(b) The four five images are the four 224x224 corner and central crops of the first square of the first image. The last is the square re-sized to 224x244. Vertical and horizontal flips of these images is also used.

Figure 2.1: Illustration of the data augmentation method used by GoogLeNet for testing purpose. This method gives a total of 144 images.

Two approaches were used for training the VGG network: single-scale training and multi-scale training (SIMONYAN; ZISSERMAN, 2014). In the first, the idea is the same of Krizhevsky, Sutskever & Hinton (2012). However, two networks were trained; the smallest size of the image was 256 in one, and 384 in the other. The patches continued to be of 224x224 pixels. In multi-scale training, each training image is re-scaled by sampling randomly the smallest size to be between a range of 256 and 512. The VGG network also divides their testing into single and multi-scale evaluation. In single-scale evaluation, the image has the same smaller side of the training image, or 384 if multi-scale training was used. In multi-scale evaluation, the testing image is re-sized three times, being the sizes of 256, 384 and 512 that gave the best results. Then, the network is densely run over these testing images, such as in the work of Sermanet et al. (2013). This approach is faster than cropping at run time and lead to similar errors.

The network based on the batch normalization method shuffle the training images in a way that prevent the same examples appearing the same mini-batch together. It also reduced the photometric distortions (IOFFE; SZEGEDY, 2015). Except for these changes, the network behaves the same way as described by Szegedy et al. (2014).

2.2.7 Regularization

Training deep network with a large-scale data can lead to over-fitting, and its avoidance was crucial to Krizhevsky, Sutskever & Hinton (2012). The Dropout technique helped generalizing the network. The idea is to turn off some neurons with a probability of 50%. So, these neurons do not contribute to forward pass and in back-propagation. A different network is now sampled for every input, and neurons learn more robust features. However, it can almost double the number of iterations to converge (SRIVASTAVA et al., 2014; HINTON et al., 2012).

Dropout have been used by the majority of the recent deep networks. Szegedy et al. (2014) even stated that it was essential to their network. However, with the new Batch Normalization method, Dropout became optional. Because the training network produces non-deterministic values for a training example, this aids in generalizing the network (IOFFE; SZEGEDY, 2015).

The addition of auxiliary classifiers can help the ability of a deep network to back-propagate their gradients more effectively. Adding classifiers in inner layers encourages discrimination in lower stages; providing extra regularization (SZEGEDY et al., 2014).

2.2.8 Model Averaging

Model averaging is the practice of averaging the output of several trained neural networks. Clarifai network stated that averaging multiple models improved performance. GoogLeNet averaged 7 models of their network, changing only sampling methods and the order of input images (SZEGEDY et al., 2014). VGG team obtained their best result averaging only their two best models and their error was slightly worse than GoogLeNet. The Batch Normalization network averaged six networks varying on following modifications: increased the initial weights, using Dropout and using non-convolutional, per-activation Batch Normalization with last hidden layers of the model (IOFFE; SZEGEDY, 2015).

2.2.9 Feature Generalization and Fine-tuning

A large-scale trained network can be used for the classification of other databases. Zeiler & Fergus (2013) propose the use of its ImageNet-trained network for the Caltech and PASCAL VOC database. All layers of the network are kept the same, except the Softmax that is replaced. The new layer is then quickly trained for each of these datasets with a small number of examples. The pre-trained network beats the state of the art for Caltech datasets at the time, and offered competitive results for PASCAL.

Oquab et al. (2014) focus more on the PASCAL VOC database and fine-tune a pre-trained ImageNet network. First, a network based on the Krizhevsky, Sutskever & Hinton (2012) is trained on the ImageNet dataset. Then the last fully-connected layer is removed, and two new fully-connected layers are trained for the new database. But using only the ILSVRC'12 dataset was not enough to beat the best PASCAL VOC'12 winners. So, some extra classes of ImageNet that overlapped with the ones of PASCAL was used, and it finally could offer a better result.

Simonyan & Zisserman (2014) generalize their ImageNet-trained network on the Caltech and PASCAL datasets without fine-tuning. It used the penultimate layer as image features combined with a linear SVM classifier. The results were very competitive, and surpassed several states of the art records.

3 THE PLANTAS DATABASE

The Plantas Database[1] (DIAS; BORGES, 2016) contains fifty different plant species or cultivars, located most in mid-western Brazil. The pictures were taken at public and private gardens and parks, between December 2015 and March 2016, so mostly in summer and in beginning of spring, Fig. 3.3.

We took the pictures considering how a common user would take them, in a mostly unconstrained way, the limitations being to avoid large display of background (sky and soil) and to keep the plant in the center of the picture. All pictures were taken during daylight, although the time varied; and some were taken at different hours and days for the same species. We used popular devices for taking these pictures: a smartphone and a digital camera, Table 3.1.

Table 3.1: Cameras Properties

Devices	Apple iPhone 6	Samsung ES25
Sensor Resolution	8 Megapixels	12.2 Megapixels
Optical Sensor Size	29 mm	10.9 mm
Lens Aperture	f/2.2	f/3.5-5.0

We collected images on the Internet, Fig. 3.5, increasing variability. First, we searched for plants using their Latin binomial nomenclature and also synonyms if they had one. We automatically discarded images with resolution below 256x256, and manually for blurred or noisy as in Fig. 3.1.

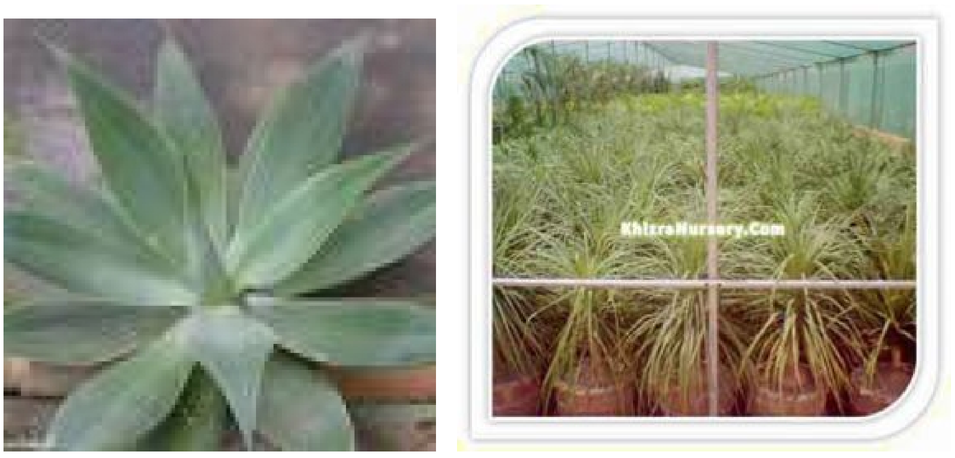


Figure 3.1: Examples of blurred and noisy images discarded by the author discretion.

We also manually verified if the plant correctly matched those searched, and if more than one plant appeared in the picture, we selected only if the target plant was in a privileged position — being centralized and bigger than the other. Then, we used the Geeqie software to look for repeated

[1]Plantas50 (Basic + Extra) by René Octavio Queiroz Dias is licensed under a Creative Commons Attribution 4.0 International License. Some of the images of Internet dataset may have copyright. Training and using recognition model for research or non-commercial use may constitute fair use of data. We do not host any images from the Internet dataset. The images of Plantas50 (Basic + Extra) and the Internet images link can be found at: https://github.com/reneoctavio/plantas.

or similar pictures.

The Geeqie software (ELLIS, 2004) searches for images with similar color content. The algorithm creates a 32x32 array for each color channel, then it divides the picture in 32 rows and columns (e.g. if a picture has 2048x1536, each element is 64x48). Afterwards, the average color of each element is saved in the array and compared to the other picture. The similarity is the percent of the correct matching of the elements of the arrays. For the author, only values greater than 0.85 are significant.

The algorithm displays the matches before deletion, thus we chose a value of 0.10 to increase the number of matches, and we visually inspected if the images were similar, and if they were the same we discarded the one with the lowest resolution.

Table 3.2: Properties of the Plantas Database

Set	No. Photos	Avg. Photos/Species	Image Resolution
Plantas50Basic	9,398	≈ 188	2048x1536, 24 bits color
Plantas50Extra	1,277	≈ 26	2048x1536, 24 bits color
Plantas50Internet	22,661	≈ 453	≥ 256x256, 24 bits color
Plantas50Expanded	33,336	≈ 667	≥ 256x256, 24 bits color

The database was divided in three sets: Plantas50Basic (Fig. 3.3), Plantas50Extra (Fig. 3.4) and Plantas50Internet (Fig. 3.5). The Plantas50Extended is the union of these three. The difference between the Basic and Extra, is that the latter contains more heterogeneous characteristic (such a bigger display of background) and is composed of discarded pictures from the Basic, but still contains enough information to identify the species. The Internet set contains all the pictures downloaded from the Internet. The Database has a total of 33,336 images with an average of 667 per class, Fig. 3.2.

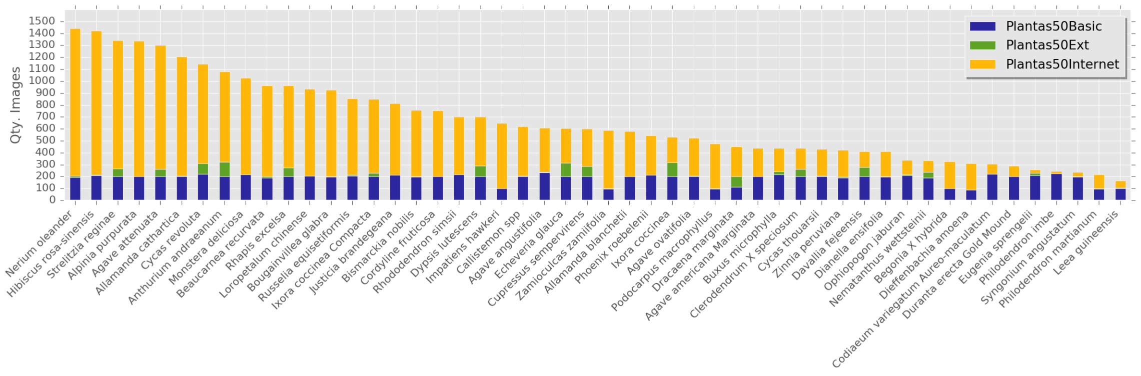


Figure 3.2: Quantity of images per classes (species or cultivars) (DIAS; BORGES, 2016)

For all the 50 species, we also collected their tree taxonomy information from the *Catalogue of Life* (ROSKOV et al., 2017) website, displayed in the Table 3.3.

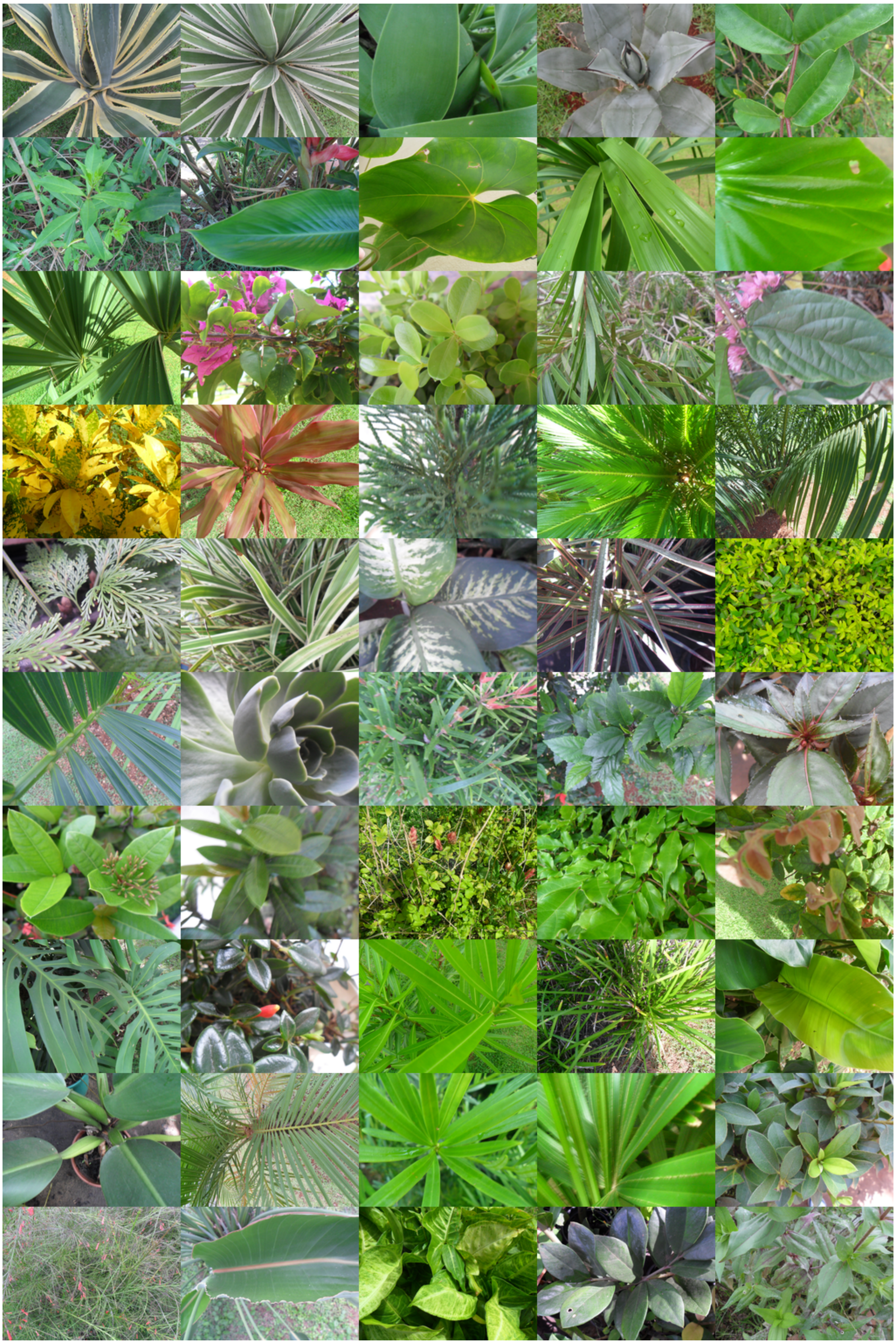


Figure 3.3: Image samples of the Plantas50Basic

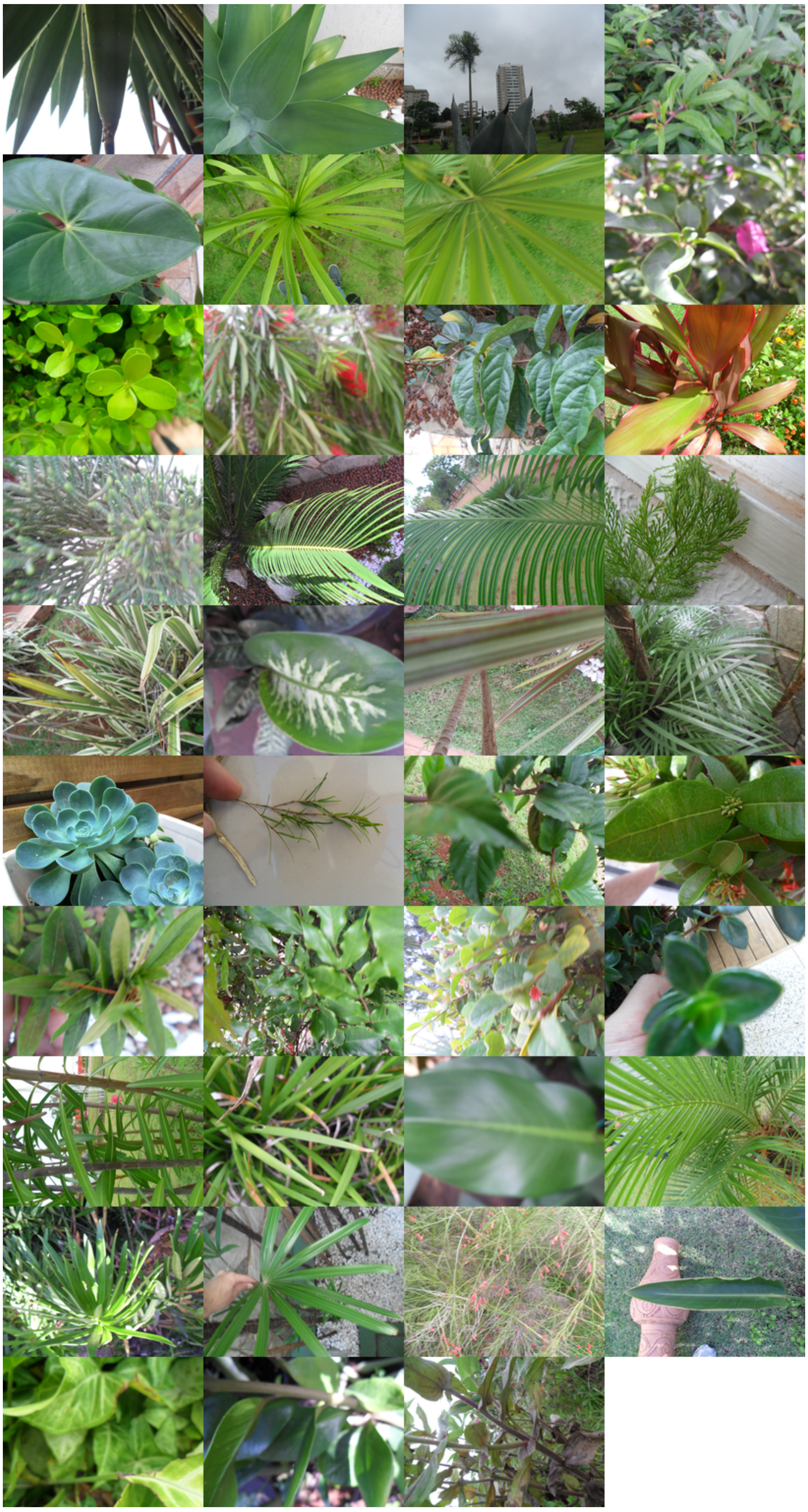


Figure 3.4: Image samples of the Plantas50Extra

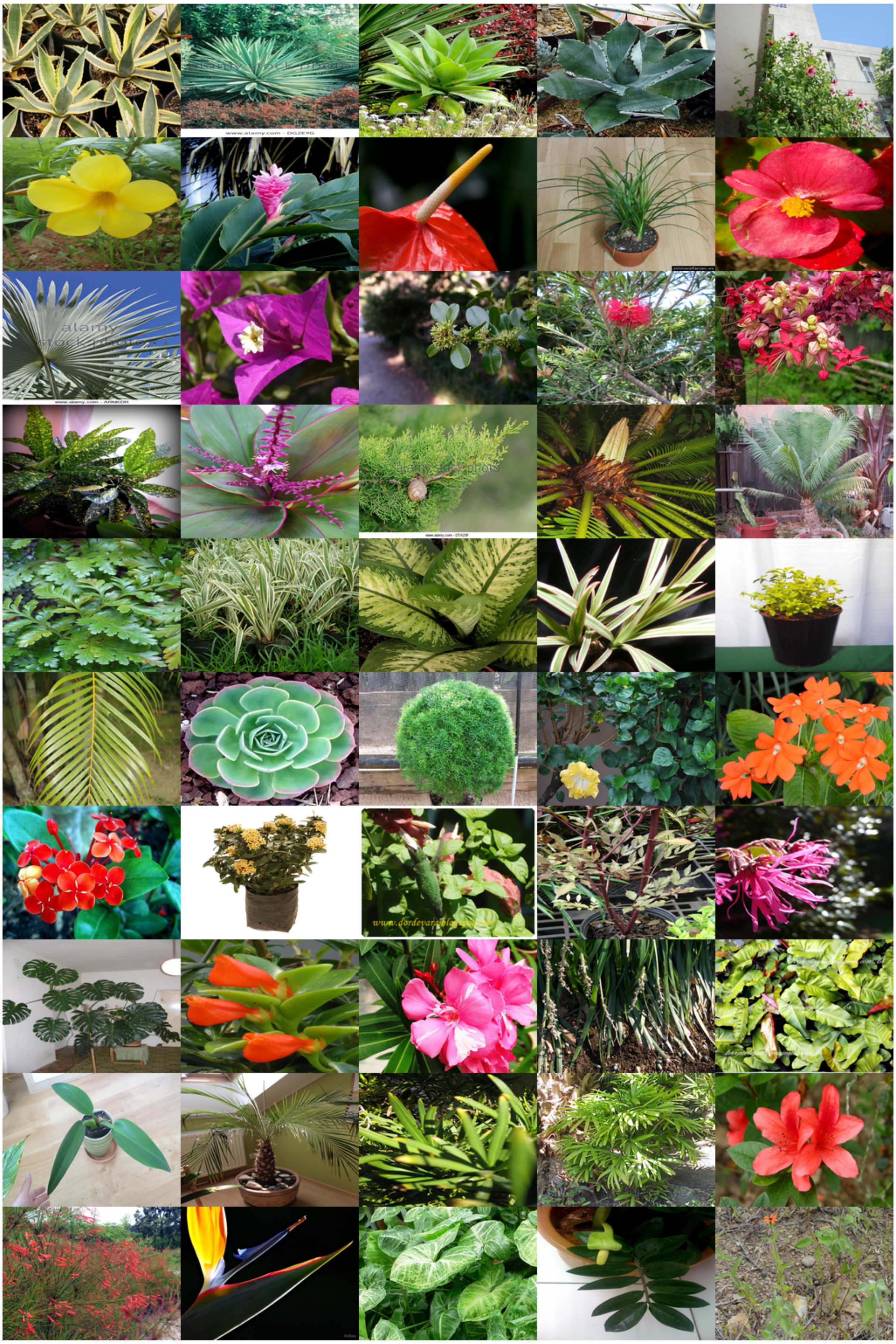


Figure 3.5: Image samples of the Plantas50Internet

Table 3.3: Species of Plantas Database and their taxonomic information

Species	Kingdom	Phylum	Class	Order	Family	Genus
Agave americana 'Marginata'	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Agave
Agave angustifolia	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Agave
Agave attenuata	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Agave
Agave ovatifolia	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Agave
Allamanda blanchetii	Plantae	Tracheophyta	Magnoliopsida	Gentianales	Apocynaceae	Allamanda
Allamanda cathartica	Plantae	Tracheophyta	Magnoliopsida	Gentianales	Apocynaceae	Allamanda
Alpinia purpurata	Plantae	Tracheophyta	Liliopsida	Zingiberales	Zingiberaceae	Alpinia
Anthurium andraeanum	Plantae	Tracheophyta	Liliopsida	Alismatales	Araceae	Anthurium
Beaucarnea recurvata	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Beaucarnea
Begonia × hybrida	Plantae	Tracheophyta	Magnoliopsida	Cucurbitales	Begoniaceae	Begonia
Bismarckia nobilis	Plantae	Tracheophyta	Liliopsida	Arecales	Arecaceae	Bismarckia
Bougainvillea glabra	Plantae	Tracheophyta	Magnoliopsida	Caryophyllales	Nyctaginaceae	Bougainvillea
Buxus microphylla	Plantae	Tracheophyta	Magnoliopsida	Buxales	Buxaceae	Buxus
Callistemon spp	Plantae	Tracheophyta	Magnoliopsida	Myrtales	Myrtaceae	Callistemon
Clerodendrum × speciosum	Plantae	Tracheophyta	Magnoliopsida	Lamiales	Lamiaceae	Clerodendrum
Codiaeum variegatum 'Aureo-maculatum'	Plantae	Tracheophyta	Magnoliopsida	Malpighiales	Euphorbiaceae	Codiaeum
Cordyline fruticosa	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Cordyline
Cupressus sempervirens	Plantae	Tracheophyta	Pinopsida	Pinales	Cupressaceae	Cupressus
Cycas revoluta	Plantae	Tracheophyta	Cycadopsida	Cycadales	Cycadaceae	Cycas
Cycas thouarsii	Plantae	Tracheophyta	Cycadopsida	Cycadales	Cycadaceae	Cycas
Davallia fejeensis	Plantae	Tracheophyta	Polypodiopsida	Polypodiales	Davalliaceae	Davallia
Dianella ensifolia	Plantae	Tracheophyta	Liliopsida	Asparagales	Xanthorrhoeaceae	Dianella
Dieffenbachia amoena	Plantae	Tracheophyta	Liliopsida	Alismatales	Araceae	Dieffenbachia
Dracaena marginata	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Dracaena
Duranta erecta 'Gold Mound'	Plantae	Tracheophyta	Magnoliopsida	Lamiales	Verbenaceae	Duranta
Dypsis lutescens	Plantae	Tracheophyta	Liliopsida	Arecales	Arecaceae	Dypsis
Echeveria glauca	Plantae	Tracheophyta	Magnoliopsida	Saxifragales	Crassulaceae	Echeveria
Eugenia sprengelii	Plantae	Tracheophyta	Magnoliopsida	Myrtales	Myrtaceae	Eugenia
Hibiscus rosa-sinensis	Plantae	Tracheophyta	Magnoliopsida	Malvales	Malvaceae	Hibiscus
Impatiens hawkeri	Plantae	Tracheophyta	Magnoliopsida	Ericales	Balsaminaceae	Impatiens
Ixora coccinea	Plantae	Tracheophyta	Magnoliopsida	Gentianales	Rubiaceae	Ixora
Ixora coccinea 'Compacta'	Plantae	Tracheophyta	Magnoliopsida	Gentianales	Rubiaceae	Ixora
Justicia brandegeana	Plantae	Tracheophyta	Magnoliopsida	Lamiales	Acanthaceae	Justicia
Leea guineensis	Plantae	Tracheophyta	Magnoliopsida	Vitales	Vitaceae	Leea
Loropetalum chinense	Plantae	Tracheophyta	Magnoliopsida	Saxifragales	Hamamelidaceae	Loropetalum
Monstera deliciosa	Plantae	Tracheophyta	Liliopsida	Alismatales	Araceae	Monstera
Nematanthus wettsteinii	Plantae	Tracheophyta	Magnoliopsida	Lamiales	Gesneriaceae	Nematanthus
Nerium oleander	Plantae	Tracheophyta	Magnoliopsida	Gentianales	Apocynaceae	Nerium
Ophiopogon jaburan	Plantae	Tracheophyta	Liliopsida	Asparagales	Asparagaceae	Ophiopogon
Philodendron imbe	Plantae	Tracheophyta	Liliopsida	Alismatales	Araceae	Philodendron
Philodendron martianum	Plantae	Tracheophyta	Liliopsida	Alismatales	Araceae	Philodendron
Phoenix roebelenii	Plantae	Tracheophyta	Liliopsida	Arecales	Arecaceae	Phoenix
Podocarpus macrophyllus	Plantae	Tracheophyta	Pinopsida	Pinales	Podocarpaceae	Podocarpus
Rhapis excelsa	Plantae	Tracheophyta	Liliopsida	Arecales	Arecaceae	Rhapis
Rhododendron simsii	Plantae	Tracheophyta	Magnoliopsida	Ericales	Ericaceae	Rhododendron
Russelia equisetiformis	Plantae	Tracheophyta	Magnoliopsida	Lamiales	Plantaginaceae	Russelia
Strelitzia reginae	Plantae	Tracheophyta	Liliopsida	Zingiberales	Strelitziaceae	Strelitzia
Syngonium angustatum	Plantae	Tracheophyta	Liliopsida	Alismatales	Araceae	Syngonium
Zamioculcas zamiifolia	Plantae	Tracheophyta	Liliopsida	Alismatales	Araceae	Zamioculcas
Zinnia peruviana	Plantae	Tracheophyta	Magnoliopsida	Asterales	Asteraceae	Zinnia

4 ENCODING METHODS AND CLASSIFICATION

Some encoding methods served as baseline to compare with convolutional neural networks trained on the *Plantas* database. These methods were: Bag of Visual Words (BoVW), Vector of Locally Aggregated Descriptors (VLAD), and Fisher Vectors (FV) (VEDALDI; FULKERSON, 2008; PERRONNIN; DANCE, 2007; PERRONNIN; SÁNCHEZ; MENSINK, 2010; JÉGOU et al., 2010).

We break these methods in two parts: an unsupervised and a supervised learning process. For the first part, we obtain a dictionary of visual words that later we will query with visual words extracted from images. For the last, we train a Linear SVM with the query response and the image label.

The dictionary is built with the following steps: we obtain several descriptors for each training image using a Dense SIFT (Fig. 4.1); we sample some of these descriptors for each image and stack the remaining descriptors together in a Descriptor Matrix; we, then, reduce the dimensionality by learning the Principal Component Analysis (PCA) projection with possible whitening regularization; geometric augmentation can be appended as feature (Fig. 4.2).

We can obtain the dictionary of visual words based on the Descriptor Matrix by following one of these approaches: Vector Quantization (VQ) or Gaussian Mixture Model (GMM). We use the first for Bag of Visual Words (BoVW) or Vector of Linearly Aggregated Descriptors (VLAD) and the latter for Fisher Vector (FV).

In Vector Quantization, with a previously selected number of words, we create a dictionary of visual words using *k*-means clustering, being the number of clusters the number of words, and to accelerate the query, we build a *k*-d-tree using these clusters centers. The Gaussian Mixture Model have a similar structure, we launch the clusters using *k*-means, and use these clusters' mean, variance, and priors as an initial guess of the GMMs. After some iterations, the GMMs output its own clusters (or modes), with their respective mean, variance and prior probability (Fig. 4.2).

In supervised learning, we obtain the visual words of each image by passing through the process of obtaining the Dense SIFT descriptors, sampling, PCA, and geometric augmentation. Then, we query the dictionary for the visual words, and the output depends on the method. The Bag of Visual Words outputs a histogram of counted visual words. VLAD gives, for each image, their visual words deviation from the mean for every visual word in the dictionary; and Fisher Vector provides the mean and covariance deviation. Then, for each image label, we train a Linear SVM with the image final descriptor (Fig. 4.3).

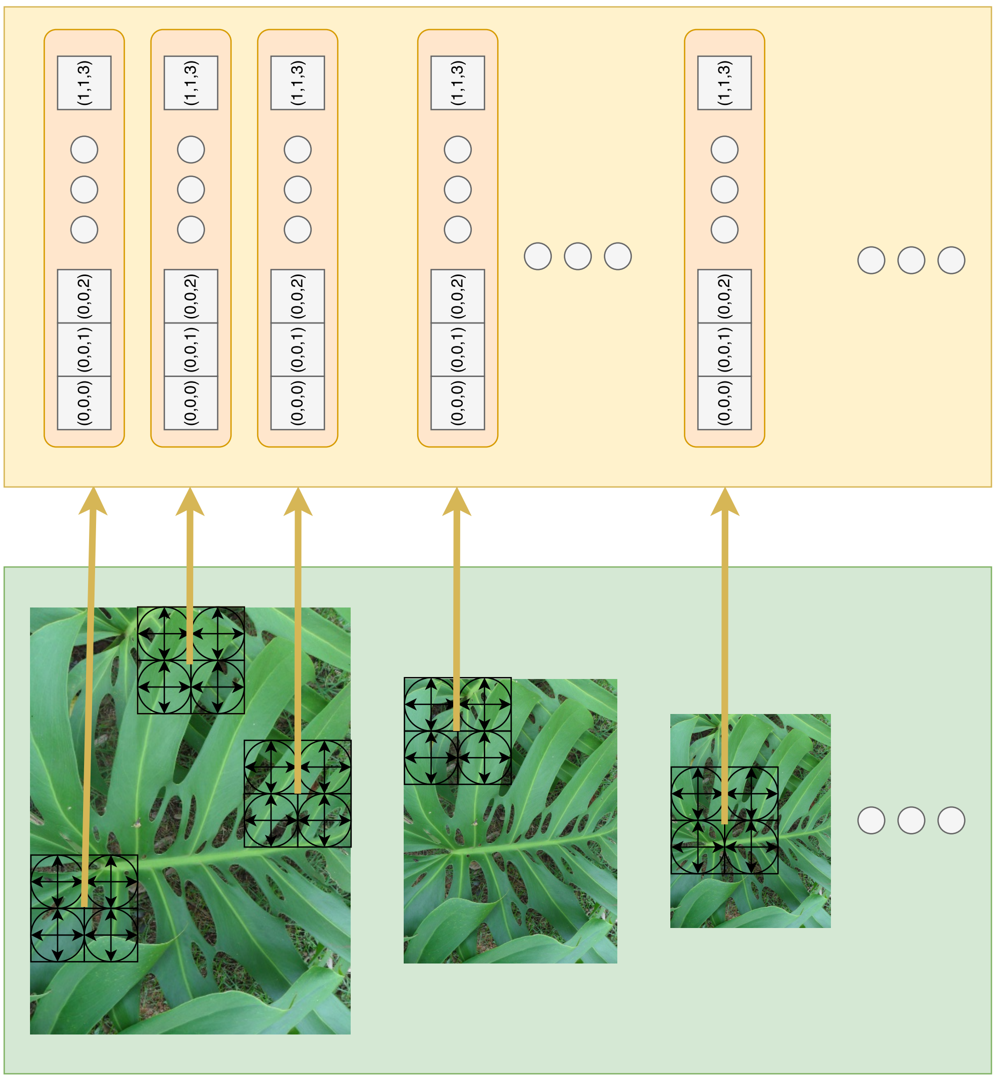


Figure 4.1: On the bottom, SIFT descriptors are taken for every image at seven different scales, then all of these descriptors are stacked in a Descriptor Matrix. In this example, the descriptor have a total of 16 bins (2x2x4).

4.1 DENSE SCALE-INVARIANT FEATURE TRANSFORM (DSIFT)

4.1.1 SIFT Descriptor

Lowe (2004) developed the SIFT algorithm in four stages: scale-space extrema detection, keypoint localization, orientation assignment, and keypoint descriptor. The first three stages detect important points, and this was a way to avoid calculating the descriptors for several points and scales. However, in Dense SIFT, we densely sample points at steps and scales determined by the user, then we apply the last stage to the selected points.

In Dense SIFT (DSIFT), all samples points have the same scale and orientation, and the parameters are the spatial bin size and sampling steps (the distance of one keypoint to another, in pixels). The size of a spatial bin, in pixels, is $m\sigma$, where σ is the scale and m the magnification factor. Thus, DSIFT alone is scale variant and we must use several bin sizes to have different

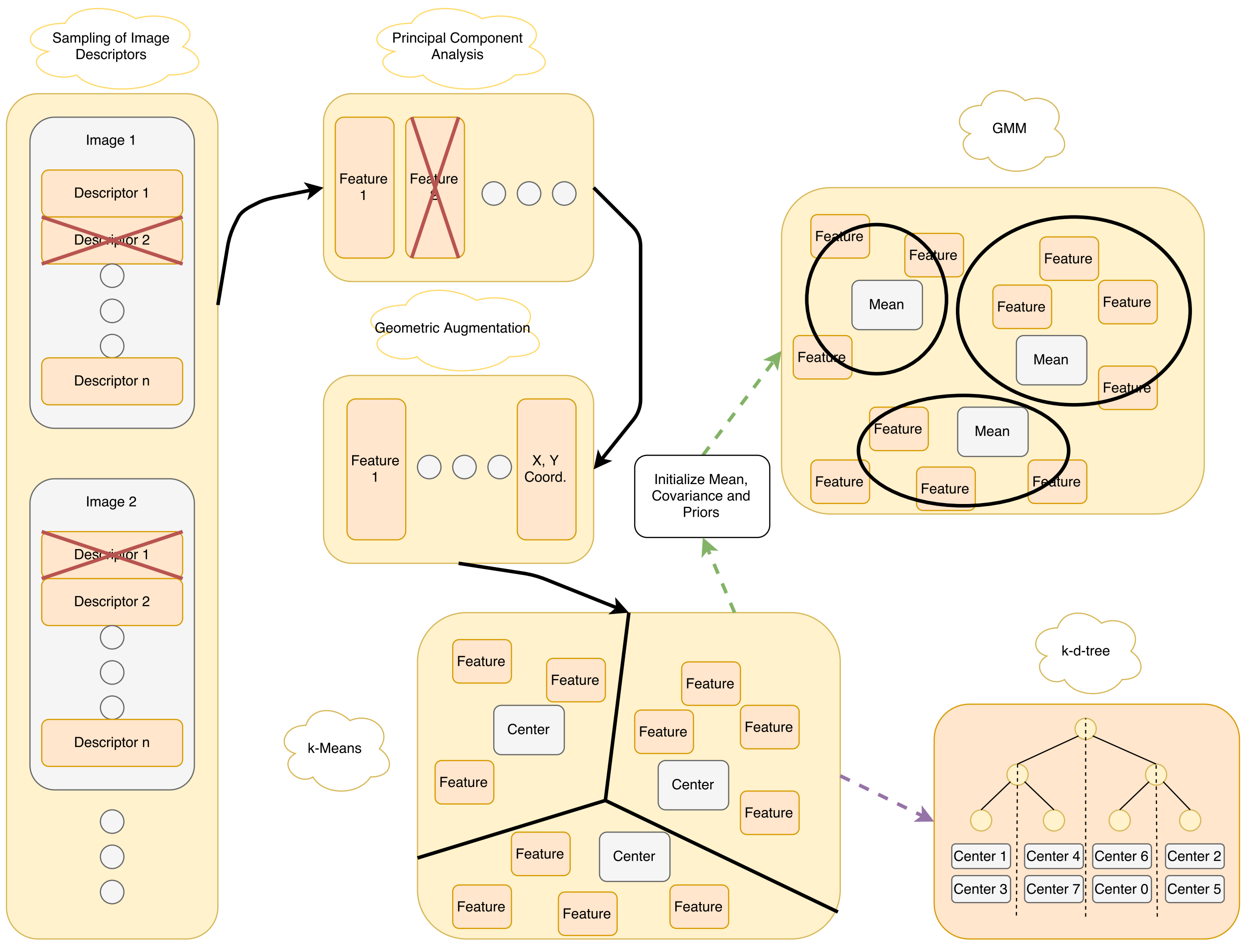


Figure 4.2: This is the part of building a visual dictionary. After we have the stacked descriptors, we sample some of them, do a PCA, cluster these features using k-means, being k the number of words. If we train a Fisher Vector, we can use this clustering for an initial guess on each word mean, covariance and prior probability, then feed a GMM (GMM is required for Fisher Vector because it needs first and second order statistics). For Bag of Visual Words or VLAD, we build a k-d-tree with these words, this tree is used for accelerating the query.

scales and become more invariant (VEDALDI; FULKERSON, 2008).

The histogram has three dimensions: two spatial and one orientation. We use 4 bins for each spatial dimension (x, y), and 8 orientation bins, which were determined empirically. The descriptor of each keypoint have a total of 128 bins (LOWE, 2004).

To construct a keypoint descriptor, we calculate the gradient magnitude and orientation at each image sample point around a keypoint, which are further weighted by the Gaussian window centered on the keypoint — so, large changes in the boundaries of the histogram impact little. Then, we accumulate these samples into orientation histogram for each spatial bin. To avoid boundary effects, a sample is projected into neighboring bins proportionally to the distance (in units of histogram bin spacing) of the sample from the bin center, which is calculated by trilinear interpolation (LOWE, 2004; VEDALDI; FULKERSON, 2008).

We used the *VLFeat* framework (VEDALDI; FULKERSON, 2008), which notation we follow.

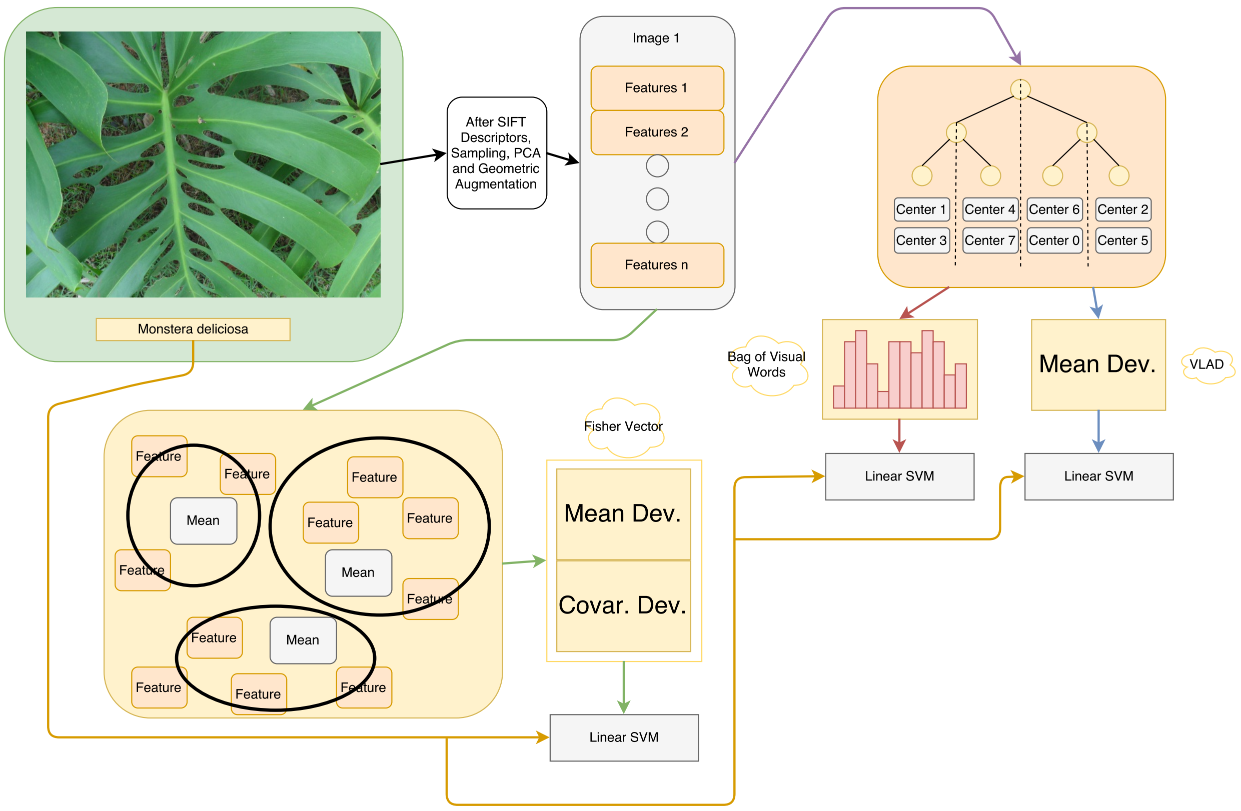


Figure 4.3: This is the part that we encode an image and train or classify it. For an image, we take its descriptor after passing through SIFT features, sampling, PCA and Geometric Augmentation. Then, we query the *k*-d-tree for Bag of Visual Words or VLAD; or a GMM for Fisher Vector. In the Bag of Visual Words, we build a histogram of the words found; in VLAD, we calculate the mean deviation of each feature; and in Fisher Vector, we calculate the mean and covariance deviations. These histogram or deviations are used to train or classify an image using a Linear SVM.

For the scale-space, we use the Gaussian function as kernel, so let g_σ be a variable-scale Gaussian:

$$g_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{1}{2}\frac{x^2 + y^2}{\sigma^2}\right). \tag{4.1}$$

Let $I_{\sigma_n}(x, y)$ be the input image and because of its finite resolution is assumed to be pre-smoothed at $\sigma_n = 0.5$ (this value is the minimum needed to prevent significant aliasing (LOWE, 2004)). Thus, the Gaussian scale space — which is the collection of smoothed images — is:

$$I_\sigma = g_{\sqrt{\sigma^2 - \sigma_n^2}} * I_{\sigma_n}, \sigma \geq \sigma_n, \tag{4.2}$$

where $*$ is the convolution signal.

The SIFT descriptor is a 3D spatial histogram of the images gradients at the local region, let $J(x, y)$ be the gradient field at a scale σ, thus:

$$J(x, y) = \nabla I_\sigma(x, y) = \left[\frac{\partial I_\sigma}{\partial x} \frac{\partial I_\sigma}{\partial y}\right]. \tag{4.3}$$

The canonical frame — which the x,y axes are centered on the local region of the descriptor not on the axes of the image — of the descriptor have each spatial bin with side one and the descriptor and images axes are the same, as in Fig. 4.4. Let N_θ, N_x and N_y be the number of orientation, x-axis, and y-axis bins.

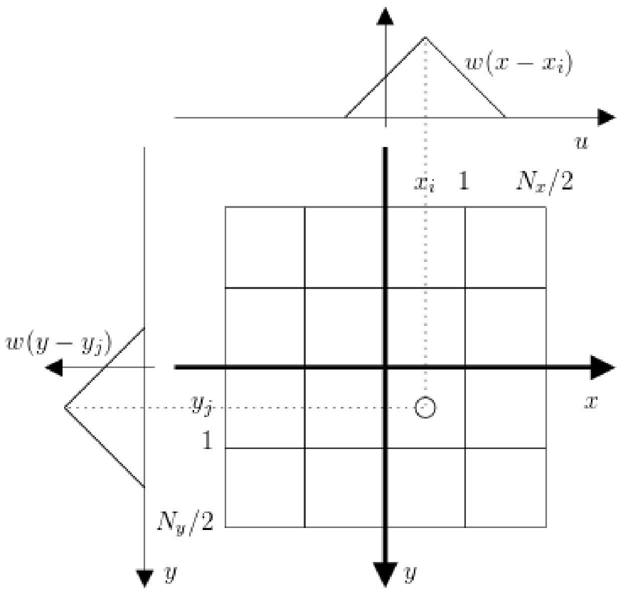


Figure 4.4: Canonical SIFT descriptor and spatial binning functions (VEDALDI; FULKERSON, 2008).

Let i, j, and t be the indexes of the x-axis, y-axis and orientation bins, respectively, and their centers are:

$$x_i = i - \frac{N_x - 1}{2}, 0 \leq i \leq N_x - 1, \tag{4.4}$$

$$y_j = j - \frac{N_y - 1}{2}, 0 \leq j \leq N_y - 1, \tag{4.5}$$

$$\theta_t = \frac{2\pi}{N_\theta} t, 0 \leq t \leq N_\theta - 1. \tag{4.6}$$

The contribution of each sample to its neighboring bins, can be done using trilinear interpolation:

$$w(z) = \max(0, 1 - |z|), \tag{4.7}$$

$$w_{ang}(z) = \sum_{k=-\infty}^{+\infty} w\left(\frac{N_\theta}{2\pi} z + N_\theta k\right). \tag{4.8}$$

The density map of these weighted contribution is:

$$f(\theta, x, y) = |J(x, y)|\delta(\theta - \angle J(x, y)). \tag{4.9}$$

Weighting by the Gaussian window of σ_{win}, the histogram is then:

$$h(t, i, j) = \int g_{\sigma_{win}}(x, y) w_{ang}(\angle J(x, y) - \theta_t) w(x - x_i) w(y - y_j) f(\theta, x, y) |J(x, y)| \mathrm{d}x\mathrm{d}y. \tag{4.10}$$

This histogram is l^2 normalized, clamped at 0.2, and l^2 normalized once more. The first unit normalization is meant to reduce the effects of illumination change; however, non-linear changes occur because of camera saturation or illumination changes that affect 3D surfaces with differing orientation by different amounts, which leads to clamping and a re-normalization. The value of 0.2 was determined empirically by the author (LOWE, 2004).

In order to rotate and translate from the canonical frame to the image frame (Fig. 4.5), let $\hat{\mathbf{x}} = [c\hat{x}\ \hat{y}]^{\mathrm{T}}$ be a coordinate in the canonical frame and $\mathbf{x} = [cx\ y]^{\mathrm{T}}$ be a coordinate in the image frame, thus:

$$\mathbf{x} = A\hat{\mathbf{x}} + T, \tag{4.11}$$

$$\hat{I}_{\hat{\sigma}}(\hat{\mathbf{x}}) = I_{A_{\hat{\sigma}}}(\mathbf{x}). \tag{4.12}$$

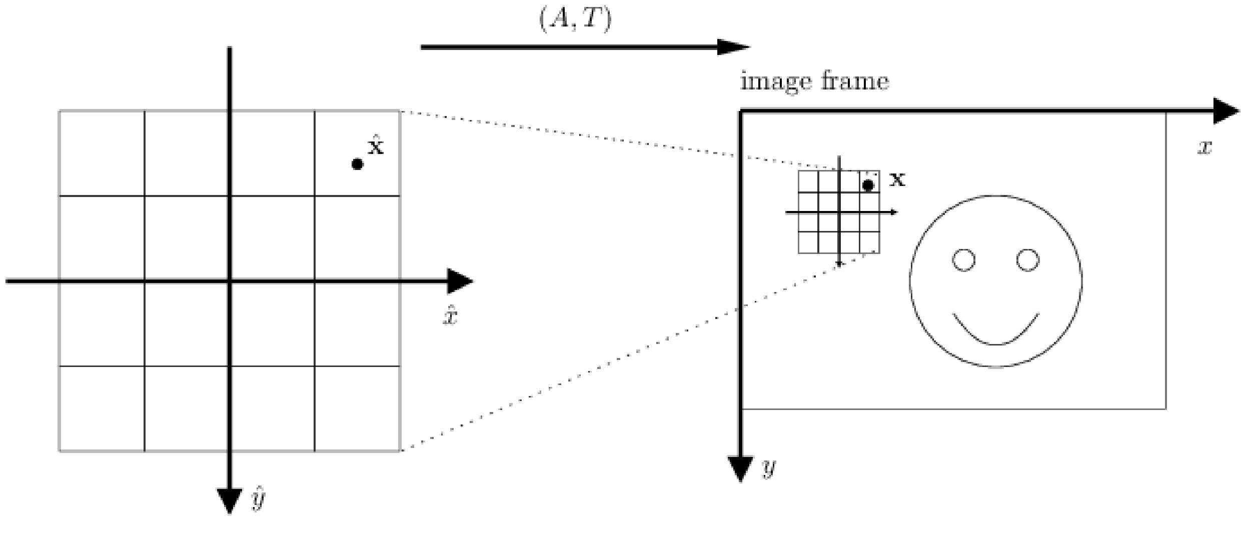


Figure 4.5: Affine transformation from canonical frame to the image frame (VEDALDI; FULKERSON, 2008).

The descriptor can be calculated in the image or canonical frame as:

$$\begin{aligned} h(t, i, j) &= \int g_{\hat{\sigma}_{win}}(\hat{\mathbf{x}}) w_{ang}(\angle \hat{J}(\hat{\mathbf{x}}) - \theta_t) w_{ij}(\hat{\mathbf{x}}) |\hat{J}(\hat{\mathbf{x}})| d\hat{\mathbf{x}} \\ &= \int g_{A\hat{\sigma}_{win}}(\mathbf{x} - T) w_{ang}(\angle J(\mathbf{x})A - \theta_t) w_{ij}(A^{-1}(\mathbf{x} - T)) |J(\mathbf{x})A| d\mathbf{x}, \end{aligned} \tag{4.13}$$

being the product of two spatial binning function:

$$w_{ij}(\hat{\mathbf{x}}) = w(\hat{x} - \hat{x}_i)w(\hat{y} - \hat{y}_j). \tag{4.14}$$

4.1.2 Dense Descriptors

Because multiple keypoints only change their position and have no orientation, we can simplify, thus:

$$\mathbf{x} = m\sigma\hat{\mathbf{x}} + T, \tag{4.15}$$

where m is the magnification factor, and $m\sigma$ is the descriptor bin size in pixels.

The kernels are defined as:

$$k_i(x) = \frac{1}{\sqrt{2\pi}\sigma_{\text{win}}} \exp\left(-\frac{1}{2}\frac{(x - x_i)^2}{\sigma_{\text{win}}^2}\right) w\left(\frac{x}{m\sigma}\right), \tag{4.16}$$

$$k_j(y) = \frac{1}{\sqrt{2\pi}\sigma_{\text{win}}} \exp\left(-\frac{1}{2}\frac{(y - y_j)^2}{\sigma_{\text{win}}^2}\right) w\left(\frac{y}{m\sigma}\right). \tag{4.17}$$

Now, the histogram becomes:

$$h(t,i,j) = (k_i k_j * \bar{J}_t)\left(T + m\sigma\begin{bmatrix} x_i \\ y_j \end{bmatrix}\right), \tag{4.18}$$

$$\bar{J}_t(\mathbf{x}) = w_{\text{ang}}(\angle J(\mathbf{x}) - \theta_t)\,|J(\mathbf{x})|. \tag{4.19}$$

If a Flat Window is used instead of Gaussian Windowing, the samples themselves are projected to the bins without being penalized by the Gaussian. However, the whole bin is then reweighed by the average Gaussian window of that bin (VEDALDI; FULKERSON, 2008). If a Flat Window is used, this can be further simplified, being σ_{win} the side of the window, then:

$$k(z) = \frac{1}{\sigma_{\text{win}}} w\left(\frac{z}{m\sigma}\right), \tag{4.20}$$

$$h(t,i,j) = (k(x)k(y) * \bar{J}_t)\left(T + m\sigma\begin{bmatrix} x_i \\ y_j \end{bmatrix}\right). \tag{4.21}$$

4.2 SAMPLING OF IMAGE DESCRIPTORS

Initially, we calculate the number of descriptors per image that will be selected:

$$\text{Number of Selected Descriptors} = \frac{\text{Number of Words} * \text{Number of Samples per Word}}{\text{Number of Images}}$$

For BoVW, it was selected a number of samples per word of 200 and for the others, 1000. Then, we select, randomly, a subset of the descriptors for each image according to the number of selected descriptors calculated beforehand.

Bag of Visual Words is less discriminative than its counterparts (zero-order estimator), and this means that it needs more visual words to match the higher dimensionality representation of Fisher Vector and VLAD. Because the dictionary of visual words have a greater number of entries, the number of samples per word needs to be lower due to the extensive use of computational resources (PERRONNIN; DANCE, 2007).

4.3 LEARNING OF PRINCIPAL COMPONENT ANALYSIS (PCA) PROJECTION

If a number of PCA dimension is used or whitening is selected, then the PCA projection is calculated. First, we calculate the sample mean $\bar{\mathbf{z}}_m$ for every descriptor. Be the descriptor matrix $\mathbf{Z}_{n,m}$, n the number of descriptors and m the number of features of all images, the new data matrix, zero centered is:

$$\mathbf{Z}'_{n,m} = \mathbf{Z}_{n,m} - \mathbf{1}_{n,1} * \bar{\mathbf{z}}^T_{m,1}. \tag{4.22}$$

Then, we calculate the covariance matrix $\mathbf{Q}_{n,n}$:

$$\mathbf{Q}_{m,m} = \frac{1}{n-1} \mathbf{Z}'^T_{n,m} \cdot \mathbf{Z}'_{n,m}. \tag{4.23}$$

Now, we calculate the eigenvalues and eigenvectors of the covariance matrix. Let $\mathbf{V}_{m,m}$ be the matrix of eigenvectors that diagonalize the covariance matrix $\mathbf{Q}_{m,m}$, being $\mathbf{D}_{m,m}$ the diagonal matrix:

$$\mathbf{V}^{-1}_{m,m} \mathbf{Q}_{m,m} \mathbf{V}_{m,m} = \mathbf{D}_{m,m}. \tag{4.24}$$

In this step, we sort the eigenvalue matrix $\mathbf{D}_{m,m}$ in a decreasing fashion. Then also organize the eigenvector matrix $\mathbf{V}_{m,m}$ according to this sorting. Then, if whitening is used, we sum each eigenvalue by the whitening regularization number multiplied by the maximum eigenvalue of the matrix, afterwards, we calculate a standard deviation vector $\mathbf{s}$ and update the matrix $\mathbf{V}_{m,m}$ such as:

$$\mathbf{s} = [s_1, \ldots, s_m], s_j = \sqrt{D_{jj}}, \tag{4.25}$$

$$\mathbf{V}_{m,m} = \mathbf{V}_{m,m} \oslash \mathbf{1}_{m,1} \cdot \mathbf{s}_{1,m}, \tag{4.26}$$

where $\oslash$ is the element-wise division.

We limit now the size of the columns of the matrix to the number of chosen PCA dimensions, d, so, $\mathbf{V}_{m,m}$ becomes $\mathbf{V}_{m,d}$. And, the transformed descriptors matrix is:

$$\tilde{\mathbf{Z}}_{n,d} = \mathbf{Z}'_{n,m} \cdot \mathbf{V}_{m,d}. \tag{4.27}$$

4.4 GEOMETRIC AUGMENTATION

If we use the Geometric Augmentation, we simply concatenate the coordinates (x, y) of the center of each frame to the descriptors matrix, and ensure that the frames coordinates and the descriptors are normalized.

4.5 LEARNING OF VISUAL VOCABULARY

4.5.1 Vector Quantization Method for BoVW or VLAD

In this approach, we learn a visual vocabulary with *k*-means clustering using Elkan's algorithm (ELKAN, 2003) and L^2 distance, being the k the number of visual words. Then, we index the clustered words as a forest of 2 trees using *k*-d-tree, a norm of L^2, and splitting the data around the median. The *k*-d-tree is used to speed-up the search of a word in the dictionary.

The *k*-d tree is a binary tree that partition space. Basically, for the first dimension we split the set in two parts by the median of this dimension. And for each half set, we split it again using the median for the second dimension until we reach k dimension in their leaves. The dimensions to be split is usually selected by the dimension with the greatest variance first.

For the experiment, it is used a form of randomized *k*-d tree (VEDALDI; FULKERSON, 2008). Each tree is made independently, and we select the dimension randomly among the five with largest variance. The query uses a best bin first algorithm, which is an approximate way to find nearest neighbors in high dimensional spaces.

4.5.2 Gaussian Mixture Model for Fisher Vector

The Gaussian Mixture Model is a collection of K Gaussian distribution and each one of them represents a cluster (or mode). For this case, the clusters are the number of visual words, the collection being a dictionary (VEDALDI; FULKERSON, 2008).

According to Vedaldi & Fulkerson (2008), to sample from a Gaussian Mixture Model, we sample the component index, $k \in \{1, \dots, K\}$, from the prior probability π_k and also the vector $\mathbf{z} \in \mathbb{R}^d$ from the k-th distribution $p(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, $\boldsymbol{\mu}_k$ the mean and $\boldsymbol{\Sigma}_k$ the covariance of the distribution. The model have the parameters $\boldsymbol{\theta} = (\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k; k)$. The density probability can be found by marginalizing the component index k, such as:

$$p(\mathbf{z}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{4.28}$$

$$p(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^d \det \boldsymbol{\Sigma}_k}} \exp\left(-\frac{1}{2}(\mathbf{z} - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} (\mathbf{z} - \boldsymbol{\mu}_k)\right). \tag{4.29}$$

Being $\mathbf{Z} = (\mathbf{z}_1, \dots, \mathbf{z}_n)$ the descriptor matrix, maximizing the log-likelihood for the learning of the model can be done as:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim\hat{p}} \log p(\mathbf{z}; \boldsymbol{\theta}), \tag{4.30}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k p(\mathbf{z}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{4.31}$$

where $\hat{p}$ is the sample distribution of the data. This is solved by using the Expectation Maximization algorithm (DEMPSTER; LAIRD; RUBIN, 1977; VEDALDI; FULKERSON, 2008), and for the experiments, the clusters are initialized using k-means clustering.

4.6 ENCODING IMAGES AND TRAINING THE CLASSIFIER

After the learning of visual words, we proceed to encode images using the Bag of Visual Words, Fisher Vector and Vector of Linearly Aggregated Descriptors methods and train them using Linear SVM.

4.6.1 Bag of Visual Words

For an image, the Bag of Visual Words searches the k-d tree for the image visual words, and for every word in the dictionary, it counts how many of them was found in the image. Thus, an image has a histogram of visual words it contains for each visual word entry (Fig. 4.6). This histogram is used as a descriptor for later training.

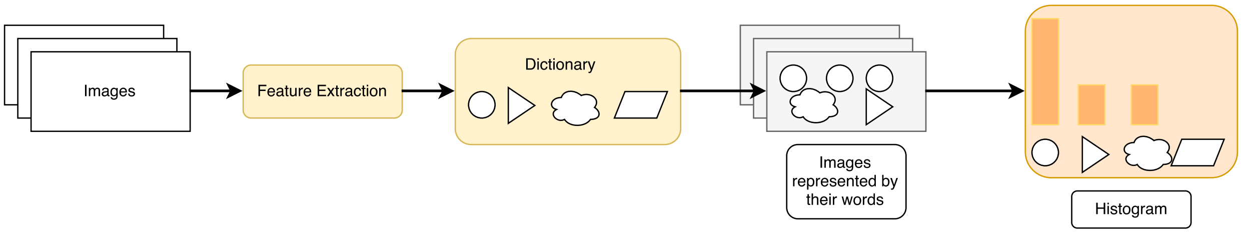


Figure 4.6: Detail of the Bag of Visual Words. First we construct an dictionary of visual words (which is done by clustering the features into k words) then for every image we obtain their visual word representation by querying the dictionary and counting how many visual words are found and build the histogram.

4.6.2 Fisher Vector

The Fisher Kernel can be modified to be used in image recognition. The image is characterized as a gradient vector derived from a Gaussian Mixture Model of visual words. This gradient vector can be used later for training in a discriminative classifier, such as SVM (PERRONNIN; DANCE, 2007; PERRONNIN; SÁNCHEZ; MENSINK, 2010; VEDALDI; FULKERSON, 2008).

Let $\mathbf{z}_i \in \mathbb{R}^d$ be a descriptor of an image taken from the descriptor matrix, $\mathbf{Z_{n,d}}$, and $\boldsymbol{\theta} = (\pi_k, \boldsymbol{\mu}_k, \Sigma_k; k = \{1, \ldots, K\})$ be the parameters of the GMM distributions. The posterior probability, the GMM associates each $\mathbf{z}_i$ to a mode k, is (VEDALDI; FULKERSON, 2008):

$$q_{ij} = \frac{\exp\left(-\frac{1}{2}(\mathbf{z}_i - \boldsymbol{\mu}_k)^{\mathrm{T}}\boldsymbol{\Sigma}_k^{-1}(\mathbf{z}_i - \boldsymbol{\mu}_k)\right)}{\sum_{t=1}^{K}\exp\left(-\frac{1}{2}(\mathbf{z}_i - \boldsymbol{\mu}_t)^{\mathrm{T}}\boldsymbol{\Sigma}_k^{-1}(\mathbf{z}_i - \boldsymbol{\mu}_t)\right)}, \tag{4.32}$$

where $j = \{1, 2, \ldots, d\}$ are the dimensions.

The mean and covariance deviation vectors are:

$$u_{ij} = \frac{1}{n\sqrt{\pi_k}}\sum_{1=1}^{n} q_{ij}\frac{z_{ji} - \mu_{jk}}{\sigma_{jk}}, \tag{4.33}$$

$$v_{jk} = \frac{1}{n\sqrt{\pi_k}}\sum_{1=1}^{n} q_{ij}\left[\left(\frac{z_{ji} - \mu_{jk}}{\sigma_{jk}}\right)^2 - 1\right]. \tag{4.34}$$

The Fisher Vector stacks these vectors for each of the modes:

$$\Phi(\mathbf{I}) = \begin{bmatrix} \vdots \\ \mathbf{u}_k \\ \vdots \\ \mathbf{v}_k \\ \vdots \end{bmatrix} \tag{4.35}$$

4.6.3 Vector of Linearly Aggregated Descriptors

This is similar to Fisher vectors. However, it has some differences such as it does not keep second-order features information and it can use k-means (VEDALDI; FULKERSON, 2008).

Let $\mathbf{z}_i \in \mathbb{R}^d$ be a descriptor of an image taken from the descriptor matrix, $\mathbf{Z_{n,d}}$, using a visual dictionary constructed by k-means clustering or GMM. Let q_{ik} the strength association of $\mathbf{x}_i$ to cluster mean $\boldsymbol{\mu}_k$, it must obey the following restrictions:

$$q_{ij} \geq 0, \tag{4.36}$$

$$\sum_{k=1}^{K} q_{ij} = 1. \tag{4.37}$$

The residuals which encodes features from $\mathbf{x}$ are:

$$\mathbf{v}_k = \sum_{i=1}^{n} q_{ij}(\mathbf{z}_i - \boldsymbol{\mu}_k). \tag{4.38}$$

These residuals are then staked:

$$\Phi(\mathbf{I}) = \begin{bmatrix} \vdots \\ \mathbf{v}_k \\ \vdots \end{bmatrix}, \tag{4.39}$$

and normalized.

In our experiments, we used square-rooting, which we apply the following function, f, to every scalar:

$$f(x) = \text{sign}(x)\sqrt{|x|}. \tag{4.40}$$

4.6.4 Training

We used the Linear Support Vector Machine (SVM) for the supervised learning of the model. The input of the Linear SVM is the histogram or deviations calculated and the label of the training images.

The Linear SVM was trained using the Stochastic Dual Coordinate Ascent algorithm (SHALEV-SHWARTZ; ZHANG, 2013). We used L^2 Parameter Regularization and Hinge-Loss.

5 NEURAL NETWORKS

Earlier methods required image descriptors before training a supervised classifier. However, deep neural networks can learn from raw data (LECUN; BENGIO; HINTON, 2015).

Training a neural network from raw data could be computationally expensive, especially in the absence of convolutional layers, which cut memory footprint by using shared weights. Although, by principle, a network with less connections are less expressive, convolutional layers exploit connections of local structures, which are enough to learn the most expressive features in most cases. For example, some pixels located at the beginning of an image is unlikely to have any correlation with pixels much further ahead.

Another improvement is that by expanding the depth of a network, it can learn more complex abstractions than those learned by shallow layers, which are usually common filters, such as edge and color detectors.

Current deep neural networks are divided into layers, which some are: the input layer, the fully-connected layer, the convolutional layer, the pooling layer and the loss layer.

Most architectures follow closely the same structure (Fig. 5.1): we forward input data (raw images) that passes through some convolutional layers and pooling layers till they reach some fully-connected layer that will output to a loss layer.

The loss function calculates how far the predicted output is from the ground-truth, and we need to minimize this function with respect to the parameters to train the network. This is an optimization problem that we will solve using Stochastic Gradient Descent in most cases.

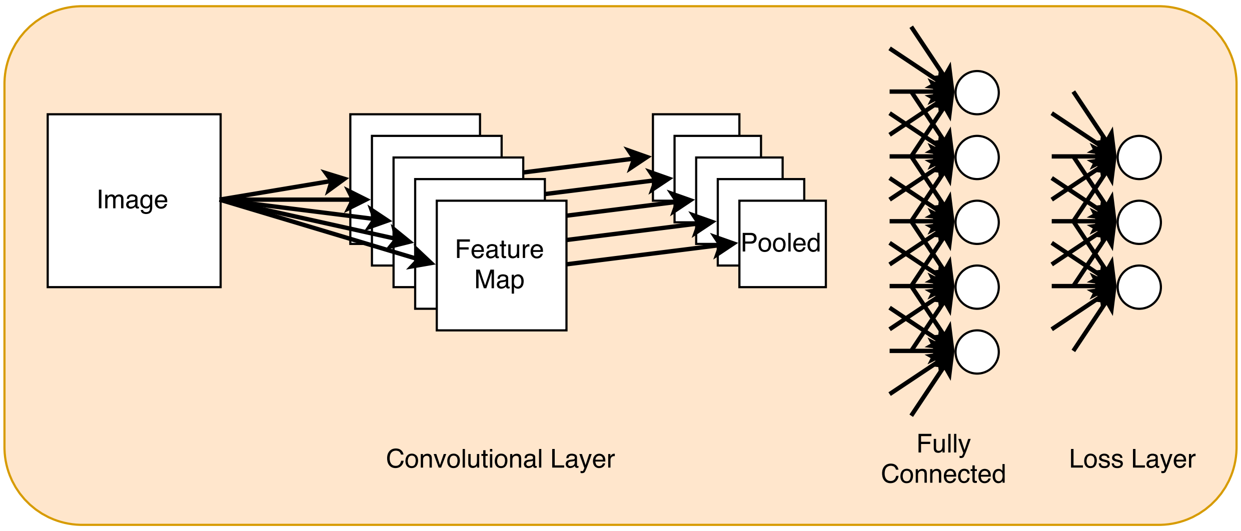


Figure 5.1: Example of a Convolutional Neural Network. We input an image, then apply some filters to this input and we have the feature maps, which pass through activation function, then by a pooling layer followed by a fully-connected layer, finally the loss is calculated in the last layer (the size of the output is the number of classes if Softmax is used).

Gradient Descent uses gradients to push the parameters in the direction that reduces the loss function to a minimum. If the function is convex — which is not the case for most deep neural networks models —, it can converge to a global minimum, if not, only a local minimum can be guaranteed. However, for deep neural networks, this is usually not a problem (CHOROMANSKA et al., 2014). We face a trade-off in choosing a learning rate: if it is high, the loss will decrease fast, however, it might not converge to a local minimum; and if it is too low, the training can take a long time.

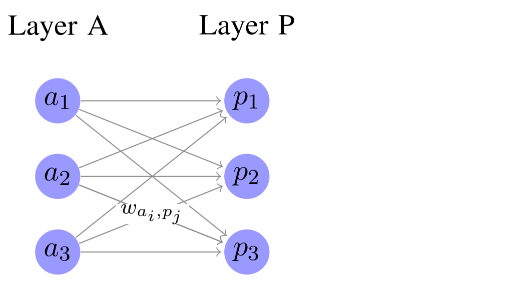
We calculate these gradients using back-propagation. Because, in neural network, a layer is a function of the activation of the anterior layer, which is a function of the input of that layer and so on, we used the chain rule of calculus to pass the partial derivatives from the very bottom to the very top of the network in order to update the parameters.

In next sections, we discuss the theoretical background of the deep neural network.

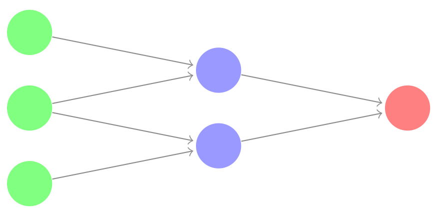
5.1 FEED-FORWARD NEURAL NETWORK (FNN)

For a feed-forward neural network, let a_i be the nodes of the anterior layer (A) and p_j the nodes of posterior layer (P). Also, let w_{a_i,p_j} be the weights that connect the interlayer nodes, and b_j the bias of the node. So, a p_j node will have a value of the activation function that receives the weighted sum of the outputs of the previous layer, as in Eq. 5.1. The anterior and posterior layers can be any two subsequent layers, including input and output layers. An example of a feed-forward network is shown in Fig. 5.2.

$$p_j = f\left(\sum_{a_i \in A} w_{a_i,p_j} a_i + b_j\right) \tag{5.1}$$



(a) A fully connected feed-forward neural network.



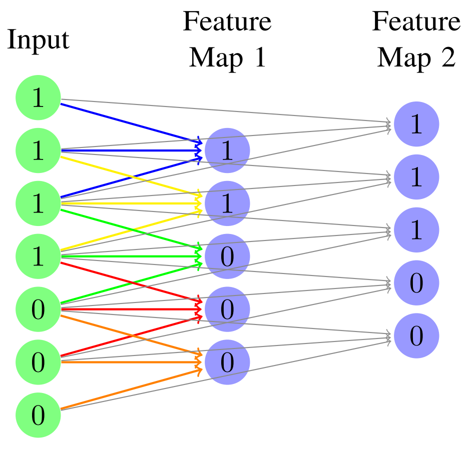
(b) A local connected feed-forward neural network.

Figure 5.2: Two feed-forward neural networks.

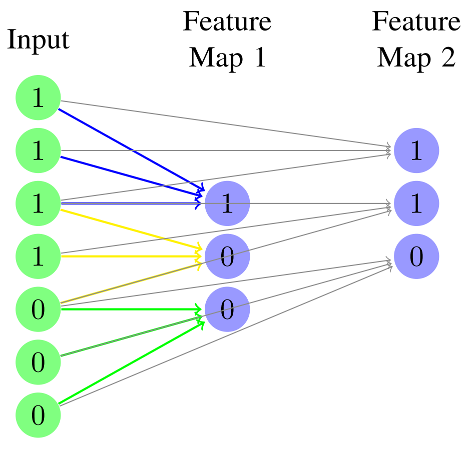
5.2 CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Networks (CNNs) are feed-forward neural networks, which are used mostly for two-dimensional input, such as images. These networks are composed by convolutional layers and have basically three components: input, trainable filters (or weights), and feature maps. The input is convoluted by some trainable filters that produce a feature map. These feature maps are tied to a spatial configuration of the image — information of images are usually packed together and not scattered — thus a local connected network is used.

Receptive field, stride, number of feature maps are parameters required in order to build a convolutional layer. Receptive field is the number of incoming connections that a neuron has. If a neuron has three receptive fields, it means that there are three neurons in the previous layer connected to it. Stride determines how the neurons will receive overlapping information as can be seen at Fig. 5.3. Higher strides lead to a higher dimensional reduction. The numbers of feature maps will be the numbers of trainable filters chosen for this layer.



(a) Stride = 1.



(b) Stride = 2.

Figure 5.3: These are convolutional layers with an input with size 7 and with two features maps. The first feature map is the result of a convolution by a filter [0,0,1], and the second by a filter [0,1,0]. The receptive field is of size 3, and in the first feature map, the colors represent the receptive field of each neuron. We, then, apply the activation function to the feature map before pushing the results forward. When the network is learning, it changes the values of the filters in order to minimize the loss.

5.3 POOLING

Pooling is a common technique used after the feature maps are obtained, and is applied to their activated output. It has two parameters: stride and filter size. The filter size corresponds to the size of the region that it will pass through an image, and the stride is how far a selected region is from another. For the values inside a region, they will be averaged if it is an average pooling, or the highest number will be chosen, if it is a max-pooling operation. Fig. 5.4 is an example of a max-pooling operation with stride 2 and filter 2x2 for a random 4x4 matrix.

$$\left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 8 & 7 & 6 \\ 2 & 3 & 4 & 5 \end{array}\right] \quad \left[\begin{array}{c|c} 6 & 8 \\ \hline 9 & 7 \end{array}\right]$$

Figure 5.4: Example of Max Pooling

5.4 ACTIVATION FUNCTIONS

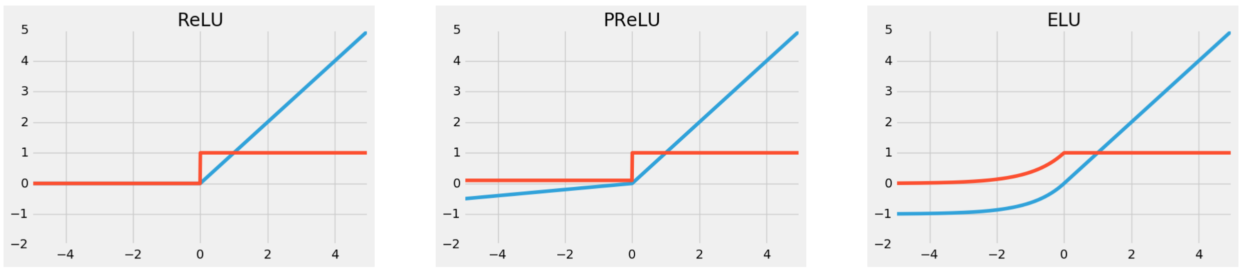
We tested the following activation functions: the Rectified Linear Unit (ReLU) (GLOROT; BORDES; BENGIO, 2011; KRIZHEVSKY; SUTSKEVER; HINTON, 2012), the Parametric Rectified Linear Unit (PReLU) (HE et al., 2015), and the Exponential Linear Unit (ELU) (CLEVERT; UNTERTHINER; HOCHREITER, 2015).

Let $\hat{\mathbf{y}}$ be the output of a layer (e.g. if it is a convolution layer, $\hat{\mathbf{y}}$ is the feature map), the ReLU, PReLU and ELU functions are respectively:

$$f(\hat{y}) = \max(0, \hat{y}), \tag{5.2}$$

$$f(\hat{y}) = \begin{cases} \hat{y} & \text{if } \hat{y} > 0 \\ a\hat{y} & \text{if } \hat{y} \leq 0 \end{cases}, \tag{5.3}$$

$$f(\hat{y}) = \begin{cases} \hat{y} & \text{if } \hat{y} \geq 0 \\ \alpha(e^{\hat{y}} - 1) & \text{if } \hat{y} < 0 \end{cases}. \tag{5.4}$$



(a) Rectified Linear Unit

(b) Parametric Rectified Linear Unit ($a = 0.1$)

(c) Exponential Linear Unit ($\alpha = 1$)

Figure 5.5: Activation functions in blue, and their derivatives in red.

5.5 OPTIMIZATION

In optimization, the solver is obliged to change the weights (or parameters) of the network to minimize a loss (or objective) function. A naïve way is to randomly change the weights and save the configuration that minimize the loss. However, this would take a very long time to obtain a good solution — because the number of weights of a deep network is massive —, and a more plausible way is to minimize the loss function by taking the derivatives w.r.t. the weights. These gradients will inform the solver the way it need to change the weights to minimize the loss.

Stochastic Gradient Descent needs the derivatives to minimize the loss and update the weights which are calculated by a backward pass of the network called backpropagation. Backpropagation applies the chain rule of calculus by stacking derivatives of functions inside functions (almost all layers have a function, and we have functions applied to functions).

The choice of loss layer depends on how a system will classify its input. In this case, we have a system with several independent classes (e.g. species) and we want the ground truth class to have the highest score possible and the others the lowest possible. We can take a probabilistic view and say that the last layer provides a probability score for each class and we consider that the last layer provides unnormalized probabilities. We use the Softmax function as way to normalize these scores, so that all these probabilities sums to one and that each probability is between 0 and 1. This can be seen as the parameter estimation by using Maximum Likelihood (ML), which is the same as minimizing the negative log-likelihood (NLL).

5.5.1 Maximum Likelihood

For a collection of training images $\mathbf{X} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, let $\mathbf{x}$ be an image and y its ground-truth, $p(\mathbf{x};\boldsymbol{\theta})$ be a parametric family of probability distributions, the maximum likelihood estimator is (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(\mathbf{x}_i;\boldsymbol{\theta}), \tag{5.5}$$

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(\mathbf{x}_i;\boldsymbol{\theta}). \tag{5.6}$$

We can rescale the function dividing by n and obtain:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}} \log p(\mathbf{x};\boldsymbol{\theta}), \tag{5.7}$$

where $\hat{p}$ is the empirical distribution of the training images.

Generalizing for the case of estimating the conditional $P(y|\mathbf{x};\boldsymbol{\theta})$ and assuming the examples to be i.i.d. (GOODFELLOW; BENGIO; COURVILLE, 2016), the estimator can be rewritten as:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i;\boldsymbol{\theta}). \tag{5.8}$$

5.5.2 Stochastic Gradient Descent

Gradient Descent is an optimization algorithm, which we try to minimize a function by taking its derivative. The derivative calculates the slope of a function at a particular point, and we move in the direction of the slope to get closer to the minimal of a loss function (by modifying the parameters) in this case.

If the function is not convex, this algorithm cannot guarantee that it moves towards the global minimum, only to a local minimum. The scale of the descent must be carefully selected to avoid being trapped in a poor local minimum. Thus, it cannot be large so it passes over the minimum, and also not small, because it would take a long time to converge.

Most deep learning problems have very large datasets and it is impossible to load and push it forward at once. Thus, Stochastic Gradient Descent divides the dataset in several batches drawn uniformly. The gradient is an expectation, and it can be approximated by a mini-batch (GOODFELLOW; BENGIO; COURVILLE, 2016).

Let $\mathbf{X} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ be a batch of images, $\mathbf{x}$ be an image with its ground-truth y, and $\hat{p}$ the empirical distribution of data; the loss function is:

$$L(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i;\boldsymbol{\theta}), \tag{5.9}$$

The gradient is computed as:

$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = -\frac{1}{n}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i;\boldsymbol{\theta}). \tag{5.10}$$

5.5.2.1 Backpropagation

To calculate these gradients, we need to take the derivatives from the last layer to the top, and these layers are composed by function of a function of a function, etc. Thus, we use the chain rule of calculus to calculate these derivatives, and this chain rule is backpropagation.

This is a recursive application of the chain rule of calculus, where derivatives are calculated by taking several known derivatives (GOODFELLOW; BENGIO; COURVILLE, 2016). Formalizing, let $\mathbf{a} \in \mathbb{R}^m$ and $\mathbf{b} \in \mathbb{R}^n$; $g : \mathbb{R}^m \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. If $\mathbf{b} = g(\mathbf{a})$ and $c = f(\mathbf{b})$, then:

$$\frac{\partial c}{\partial a_i} = \sum_j \frac{\partial c}{\partial b_j}\frac{\partial b_j}{\partial a_i}, \tag{5.11}$$

$$\nabla_{\mathbf{a}} c = \left(\frac{\partial \mathbf{b}}{\partial \mathbf{a}}\right)^{\mathrm{T}} \nabla_{\mathbf{b}} c. \tag{5.12}$$

5.5.2.2 Momentum

The momentum algorithm (Alg. 1) can accelerate learning by accumulating an exponentially decaying moving average of past gradients and keeping moving in that direction, which is called the velocity $\mathbf{v}$ (GOODFELLOW; BENGIO; COURVILLE, 2016). Then, for α as learning rate, and μ as momentum, which weighs the previous velocity, the update of parameters is given by:

$$\mathbf{v} \leftarrow \mu\mathbf{v} - \alpha\mathbf{g}, \tag{5.13}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}. \tag{5.14}$$

Algorithm 1 Stochastic Gradient Descent (SGD) with momentum (GOODFELLOW; BENGIO; COURVILLE, 2016)

```
Require: Learning rate $\alpha$, momentum parameter $\mu$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\mathbf{v}$.
1: while stopping criterion not met do
2:     Sample a mini-batch of $n$ examples from the training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$
3:     Compute gradient estimate: $\mathbf{g} \leftarrow -\frac{1}{n}\nabla_{\boldsymbol{\theta}} \sum_i \log P(y_i|\mathbf{x}_i; \boldsymbol{\theta})$
4:     Compute velocity update: $\mathbf{v} \leftarrow \mu\mathbf{v} - \alpha\mathbf{g}$
5:     Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$
```

5.5.3 Softmax

For Softmax (Fig. 5.6), let $\hat{\mathbf{y}}_i$ be the predicted unnormalized log probabilities of an image, where $\hat{y}_{i,k} = \log \tilde{P}(y_i = j|\mathbf{x}_i; \boldsymbol{\theta})$, the predicted probabilities is given by:

$$P(y_i = j|\mathbf{x}_i; \boldsymbol{\theta}) = \frac{\exp(\hat{y}_{i,j})}{\sum_k \exp(\hat{y}_{i,k})}. \tag{5.15}$$

This implies that for every image, the probabilities for each label must be between 0 and 1, and the sum of all probabilities have to be 1. The loss is calculated as:

$$L(\boldsymbol{\theta}; \mathbf{X}) = -\frac{1}{n}\sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i; \boldsymbol{\theta}). \tag{5.16}$$

For an image, its loss is zero when the predicted label is the same of ground truth label. The loss with the Softmax, then, becomes:

$$L(\boldsymbol{\theta};\mathbf{X}) = -\frac{1}{n}\sum_{i=1}^{n}\left[\hat{y}_{i,j} - \log\sum_{k}\exp\hat{y}_{i,k}\right], \tag{5.17}$$

where j is the ground-truth label.

To facilitate the calculation of the gradients, we call a one-hot vector, $\mathbf{y}_i$, the ground-truth vector of an image, which the position of the correct class has a value of one, and the others zero. For example, if the correct class of an image is 5, the one-hot vector at position 5 will have a value of 1, and the other positions will be 0. We can rewrite the Softmax function and loss as:

$$P(y_{i,j} = 1|\mathbf{x}_i;\boldsymbol{\theta}) = \frac{\exp(\hat{y}_{i,j})}{\sum_k \exp(\hat{y}_{i,k})}, \tag{5.18}$$

$$L(\boldsymbol{\theta};\mathbf{X}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j} y_{i,j}\log P(y_{i,j}|\mathbf{x}_i;\boldsymbol{\theta}). \tag{5.19}$$

Now, we calculate the gradient of a single image (at position i) and a single label (at position j) w.r.t. the input of Softmax (using back-propagation), which is:

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = \sum_k \frac{\partial L_i}{\partial P(y_{i,k}|\mathbf{x}_i;\boldsymbol{\theta})}\frac{\partial P(y_{i,k}|\mathbf{x}_i;\boldsymbol{\theta})}{\partial \hat{y}_{i,j}}, \tag{5.20}$$

$$= \frac{1}{n}\left[P(y_{i,j}|\mathbf{x}_i;\boldsymbol{\theta}) - y_{i,j}\right]. \tag{5.21}$$

We can rewrite the gradient of an image as a subtraction of the normalized predicted output from the ground-truth vector, such as:

$$\left(\nabla_{\hat{\mathbf{y}}} L\right)_i = \frac{1}{n}\left[P(\mathbf{y}_i|\mathbf{x}_i;\boldsymbol{\theta}) - \mathbf{y}_i\right]. \tag{5.22}$$

5.6 REGULARIZATION

Over-fitting is common in learning deep representations of a neural network, because of massive number of parameters. To mitigate this risk, we can use some regularization techniques that penalizes training in order to generalize better for testing sets.

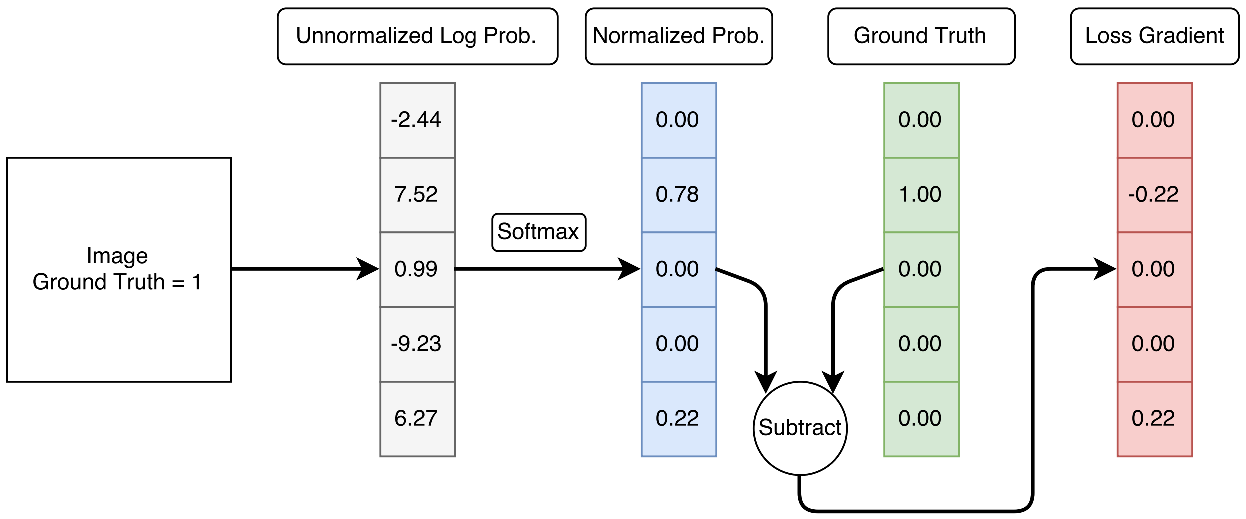


Figure 5.6: Example of an implemented Softmax Loss Layer.

5.6.1 L^2 Parameter Regularization (Weight Decay)

Weight Decay increases the variance of input values, which are not considered so valuable anymore; thus, resulting in gradients with smaller magnitude that change little the value of the parameters. This is useful with small datasets because the network will be penalized if it tries to exactly fit the input. In the end, it will generalize better to unseen data (GOODFELLOW; BENGIO; COURVILLE, 2016). We add the last term to the objective function, such as:

$$\tilde{L}(\boldsymbol{\theta};\mathbf{X}) = L(\boldsymbol{\theta};\mathbf{X}) + \alpha\frac{1}{2}||\boldsymbol{\theta}||_2^2, \tag{5.23}$$

where α is the weight decay parameter. Low α increases the value of data and increase the chance of over-fitting, while a very high α can even impede the network to learn.

5.6.2 Dropout

The contributions of Dropout (Fig. 5.7) are twofold: it prevents over-fitting by adding noise to the hidden units, and combine several sampled neural networks — because the neurons are shut down with given chance, this is equivalent to train several different networks (HINTON et al., 2012; SRIVASTAVA et al., 2014; GOODFELLOW; BENGIO; COURVILLE, 2016).

Let $\hat{\mathbf{y}}$ be the output of a layer and p be the probability of a node to output one, the dropout can be described as:

$$\mathbf{r} \sim \text{Bernoulli}(\mathbf{p}), \tag{5.24}$$

$$\tilde{\mathbf{y}} = \mathbf{r} \circ \hat{\mathbf{y}}, \tag{5.25}$$

where $\circ$ is the element-wise multiplier.

The neurons are shutdown during training with probability p. However, during test phase, the neurons are never shut down, instead, the weights are multiplied by p (SRIVASTAVA et al., 2014).

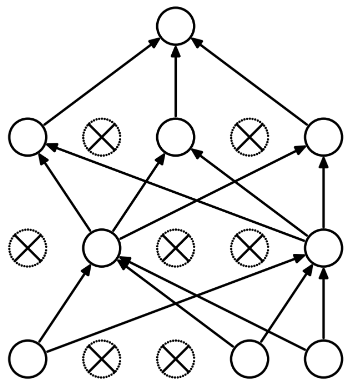


Figure 5.7: Neural Network with Dropout during training (SRIVASTAVA et al., 2014).

5.6.3 Local Response Normalization

According to Krizhevsky, Sutskever & Hinton (2012), Local Response Normalization aids in generalization, despite ReLUs not requiring normalization (because of their non-saturating property). Let $a^i_{x,y}$ be the activity of a neuron by applying the kernel i at position (x, y), the response normalized is:

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}, \tag{5.26}$$

where n are the adjacent kernel maps at the same spatial position, and N is the number of kernels. The constants k, α, and β are hyper-parameters that can be determined by the validation set (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

5.6.4 Batch Normalization

During back-propagation, the gradients update the parameters considering that other layers are not updating theirs at the same time. However, in practice, they do, and this can lead to unpredicted results, especially for deep models. One way to resolve is by building a n-th order optimization algorithm, but this is very expensive. Batch normalization re-parametrize these deep models, mitigating this coordination problem, which is done by keeping responses as a unit Gaussian (zero mean and unit variance) (IOFFE; SZEGEDY, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016).

For a mini-batch of activations of a layer $\mathbf{B}_{n,m}$, being n the number of examples and m the number of activations, we normalize on each activation:

$$\mathbf{B}' = \frac{\mathbf{B}_{n,m} - \boldsymbol{\mu}_m}{\boldsymbol{\sigma}_m}, \tag{5.27}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation of each activation.

During training, they are calculated as:

$$\boldsymbol{\mu} = \frac{1}{n}\sum_i \mathbf{B}_{i,:}, \tag{5.28}$$

$$\boldsymbol{\sigma} = \sqrt{\frac{1}{n}\sum_i (\mathbf{B} - \boldsymbol{\mu})_i^2}. \tag{5.29}$$

This information is kept during training time by a moving average. Because normalizing to a unit Gaussian diminishes the expressive power of a network, a per-channel bias, $\boldsymbol{\beta}$, and scaling factor, γ, can be learned (GOODFELLOW; BENGIO; COURVILLE, 2016; IOFFE; SZEGEDY, 2015; JIA et al., 2014). The final output is:

$$\mathbf{B}' = \gamma\mathbf{B}' + \boldsymbol{\beta}. \tag{5.30}$$

5.7 CAFFE FRAMEWORK

We chose *Caffe* Framework (JIA et al., 2014) to test how some current deep learning models would respond to our new database. The framework was developed as a compositional model for deep neural networks. They call *Net* a complete model with *Layers* that an architecture might have. Because of the flux of information of forward and back-propagation, the framework designed a data holder, called *Blob*, which also has the ability of synchronization between the CPU and GPU memory.

The *Blob* was designed with computer vision applications in mind. So, they have a 4D format, with the shape of $(N \times K \times H \times W)$, being N the batch size, K the number of channels, H the height, and W the width. However, the user can modify any kind of data before inputting it to the network. The *Blob* allocate two arrays, the *data* and *diff*, the first for forwarded data, and the latter holds the computed gradient (JIA et al., 2014).

The *Layer* is where the operations occurs, for example, it can convolve filters and apply non-linearities. It takes its input through *bottom* connections and output through *top* connections. *Setup*, *forward*, and *backward* are the phases that must be computed. In *setup*, the layers and their connection are initialized once. *Forward* computes the output based on the input and send to top. *Backward* computes the gradient w.r.t. the input given the gradient w.r.t. the top and send it to the bottom, if the layer has parameters, also computes the gradient w.r.t. its parameters and stores it (JIA et al., 2014).

The *Net* is a directed acyclic graph (DAG) of its layers (Fig. 5.8), and its responsible for initializing the blobs and layers and ensuring the correctness the forward and backward passes, the gradients used in back-propagation is calculated by automatic differentiation (JIA et al., 2014).

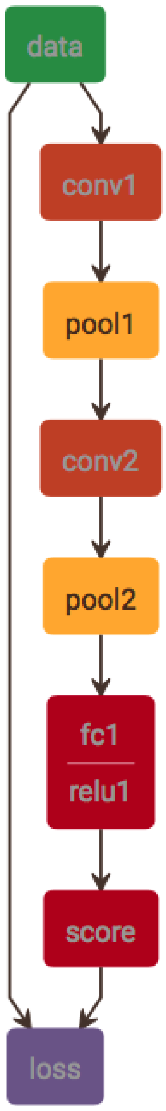


Figure 5.8: Variant of the LeNet (LECUN et al., 1989) as a Directed Acyclic Graph

The *Solver* is responsible for calling forward and backward passes in order to train the network, using a specific method, such as Stochastic Gradient Descent. It needs some parameters such as the learning policy, number of iterations, when to save snapshots of the network, and the interval to test the network. The *Solver* is also responsible for updating the parameters of the network seeking to minimize training loss (JIA et al., 2014).

6 EXPERIMENTS

The experiments are divided into two separate stages: in the first we train some models using Linear SVM and the encoded images to serve as baseline models; in the latter we use deep neural networks.

In the first stage, we encode these images using three different methods: Bag of Visual Words, Fisher Vector, and VLAD. We create models varying the type of encoding method and with or without Geometric Augmentation.

In the second stage, we choose some common deep learning architectures. Then, for each architecture we create models that varies the type of activation function and in some we also test fine-tuning, in which we load pre-trained weights.

These two stages are not connected, thus they have different database set-up. In this chapter, we present how we prepare the image database for each stage, the parameters selected for each model, and the models variants.

6.1 DATABASE PREPARATION

For each model, we trained using two different datasets: Plantas50Basic and Plantas50Extended. They were divided into training, validation, and testing sets in a 70%, 15%, and 15% fashion. We shuffled the images before assigning them to their sets. A cross-validation scheme was discarded because of the expensive computational cost involved for the size of this database, which is large enough to avoid it (GOODFELLOW; BENGIO; COURVILLE, 2016). We estimate that using a k-fold cross-validation, for $k = 10$, would take at least 45 days to finish training and testing.

For the encoding methods, we reduced the images' height to 256 pixels while keeping their aspect ratio. And, for neural networks, we resized them to 256x256 pixels. These files are compressed using the JPEG format with different rates of compression. However, these images are transformed to raw BGR 8-bits before being input to a network (which gives 192 kBytes per image if 256x256).

6.2 ENCODING METHODS AND CLASSIFICATION

We trained three encoding methods to serve as baselines: Bag of Visual Words, Fisher Vector, and VLAD (VEDALDI; FULKERSON, 2008; PERRONNIN; SÁNCHEZ; MENSINK, 2010; JÉGOU et al., 2010). We used Geometric Augmentation in all three, and also tested an additional Fisher Vector model without it (DIAS; BORGES, 2016). In total, we trained four models: BoVW-

Aug, Fisher Vector, Fisher Vector-Aug, and VLAD-Aug.

For all these experiments, we calculated DSIFT descriptors for seven different scales (S), $S = \{2^{-\frac{1}{2}i}, i \in [0, 1, \ldots, 6]\}$; a step of 4 pixels, which means the center distance of one frame to the next; and bins 8 pixels wide. Each descriptor has 128 features (4 x-axis, 4 y-axis, and 8 orientation dimensions).

The Linear SVM was solved using SDCA (SHALEV-SHWARTZ; ZHANG, 2013), with $\epsilon = 0.001$, a bias multiplier of 1, a maximum number of iterations of $100 *$ Number of Training Images, and $\lambda = \frac{1}{\text{Number of Training Images}}$. We used half of the number of words prescribed by the VLFeat framework (VEDALDI; FULKERSON, 2008), because we have less classes than Caltech101, to which database this system was devised.

Table 6.1: Encodings Parameters

Model	Num. of Words	Geometric Ext.	PCA Dim.	Whitening	Whit. Reg.
BoVW-Aug	2048	Yes	100	Yes	0.01
Fisher Vector	128	No	80	No	-
Fisher Vector-Aug	128	Yes	80	No	-
VLAD-Aug	128	Yes	100	Yes	0.01

6.3 DEEP LEARNING MODELS

6.3.1 Architectures

For training a neural network with our *Plantas* database, we considered that all plants are independent of each other, so the network outputs scores for each of the 50 species or cultivars (DIAS; BORGES, 2016).

We chose to test several different architectures instead of going with the most recent and successful, because, the deeper the architecture, the higher the chance of over-fitting a middle-sized database ($\approx 33,000$ images), and simpler network might work better with a smaller dataset. We tested the following architectures: AlexNet, CaffeNet, GoogLeNet, Inception, and ResNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; JIA et al., 2014; SZEGEDY et al., 2014; IOFFE; SZEGEDY, 2015; HE et al., 2016).

AlexNet (Table 6.2) was the first deep neural network to win the ImageNet Challenge in 2012, spurring new developments in the field. The major innovation was the use of *Dropout* and massive parallelization computation of Convolutional Layers (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). The CaffeNet is the modified version of AlexNet developed by the *Caffe* framework, which basically just swap the place of the Local Response Normalization with Max Pooling operations (JIA et al., 2014).

GoogLeNet introduces the Inception module (Fig. 6.1a), which permitted the creation of more deep network without increasing memory footprint (Table 6.3) (SZEGEDY et al., 2014). The

Inception architecture is GoogLeNet with Batch Normalization and some other minor modifications. Inception is also the first deep network to be trained without *Dropout* (IOFFE; SZEGEDY, 2015).

He et al. (2016) (Table 6.4) address propagation degradation problem by short-cutting connections of one layer to some more ahead of it, instead of just stacking layers, such as in Fig. 6.1b.

Table 6.2: Architecture of Krizhevsky, Sutskever & Hinton (2012) (AlexNet)

Type	Kernel/Stride	Output Size
Convolution	11x11/4	96x55x55
LRN		96x55x55
Max Pooling	3x3/2	96x27x27
Convolution	5x5/1	256x27x27
LRN		256x27x27
Max Pooling	3x3/2	256x13x13
Convolution	3x3/1	384x13x13
Convolution	3x3/1	384x13x13
Convolution	3x3/1	256x13x13
Max Pooling	3x3/2	256x6x6
Linear		4096
Dropout (50%)		4096
Fully-Connect.		4096
Dropout (50%)		4096
Fully-Connect.		1000
Softmax		1000

Table 6.3: Architecture of Szegedy et al. (2014) (GoogLeNet)

Type	Kernel/Stride	Output Size
Convolution	7x7/2	112x112x64
Max Pooling	3x3/2	56x56x64
Convolution	3x3/1	56x56x192
Max Pooling	3x3/2	28x28x192
Inception (3a)		28x28x256
Inception (3b)		28x28x480
Max Pooling	3x3/2	14x14x480
Inception (4a)		14x14x512
Inception (4b)		14x14x512
Inception (4c)		14x14x512
Inception (4d)		14x14x528
Inception (4e)		14x14x832
Max Pooling	3x3/2	7x7x832
Inception (5a)		7x7x832
Inception (5b)		7x7x1024
Average Pooling	7x7/1	1024
Dropout (40%)		1024
Fully-Connect.		1000
Softmax		1000

Table 6.4: Architecture of He et al. (2016) (ResNet)

Layer Name	Output Size	18-layer	34-layer	50-layer	101-layer	152-layer
Conv-1	112x112	7x7, 64, stride 2				
Conv-2_x	56x56	3x3 max pooling, stride 2				
		$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times 3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times 3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times 3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times 3$
Conv-3_x	28x28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times 4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times 4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times 4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times 8$
Conv-4_x	14x14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times 6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times 6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times 23$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times 36$
Conv-5_x	7x7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times 3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times 3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times 3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times 3$
	1x1	average pooling, 1000, fully-connected, softmax				

6.3.2 Models

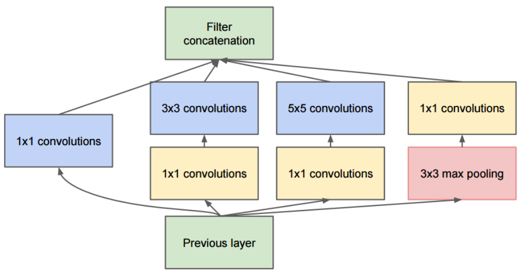
We called *Default* for models we trained without modification, following their implementation in the *Caffe* framework, or using the author given training files in the case of ResNet. Then, we tested the effect of some recent Activation Functions, namely: Rectified Linear Unit (ReLU), Parametric Rectified Linear Unit (PReLU), and Exponential Linear Unit (ELU) (GLOROT; BORDES; BENGIO, 2011; HE et al., 2015; CLEVERT; UNTERTHINER; HOCHREITER, 2015). We appended their acronym in the model name (DIAS; BORGES, 2016).

Then, we tested fine-tuned techniques, which we divided into three: Zero Method, SVM Method, and Complete Training Method. In the first, we zero all layers' learning rate, except the last, which is trained on a new dataset. In the second, we take the descriptor of the penultimate layer and train using a Linear SVM. And, in the latter, we train the whole network, as if we were training from scratch (Fig. 6.2) (DIAS; BORGES, 2016).

We used the pre-trained models of CaffeNet and GoogLeNet of the Berkeley Vision and Learning Center (JIA et al., 2014) and the pre-trained model of ResNet of He et al. (2016) on the 2012 ImageNet Challenge database.

We trained 23 deep learning models, of 5 architectures:

- AlexNet: AlexNet-Default;
- CaffeNet: CaffeNet-Default, CaffeNet-Finetuned, CaffeNet-Finetuned-SVM, CaffeNet-Finetuned-Zero, CaffeNet-PReLU;
- GoogLeNet: GoogLeNet-Default, GoogLeNet-ELU, GoogLeNet-Finetuned, GoogLeNet-Finetuned-PReLU, GoogLeNet-Finetuned-SVM, GoogLeNet-Finetuned-SVM, GoogLeNet-PReLU;
- Inception: Inception-Default, Inception-ELU, Inception-PReLU;
- ResNet: ResNet-50-Default, ResNet-50-Finetuned, ResNet-50-Finetuned-SVM, ResNet-50-PReLU, ResNet-101-Finetuned-SVM, ResNet-152-Finetuned-SVM.



(a) Inception module (IOFFE; SZEGEDY, 2015).

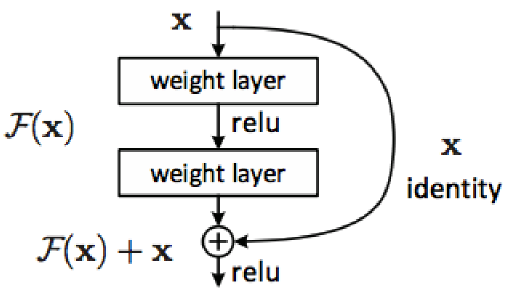

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
(b) Residual module (HE et al., 2016).

Figure 6.1: The Inception Module and Residual Module

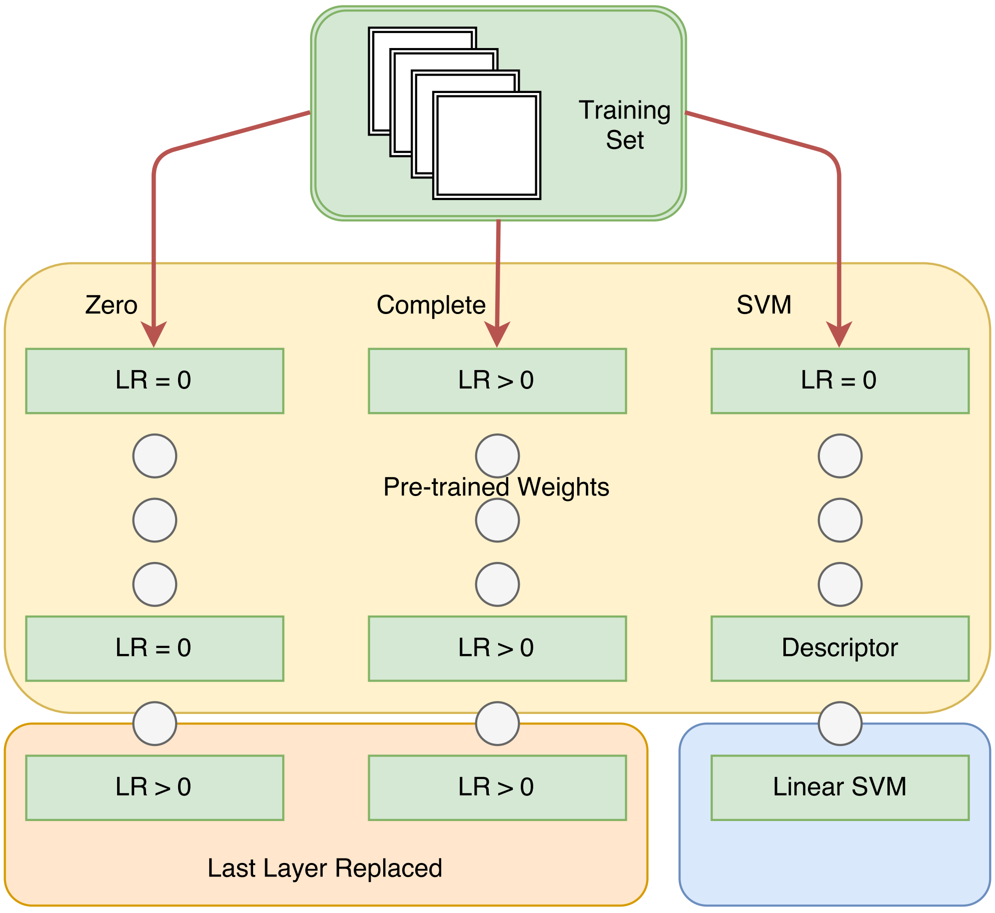


Figure 6.2: Fine-tuned models: Zero, Complete, SVM. In Zero, we set the learning rate to zero to all loaded pre-trained weights and train the last layer; in Complete, we train all layers; in SVM we take the training images descriptors of the penultimate layer and train them using a Linear SVM.

6.3.3 Data Setup

A training epoch happens when all the images of the training database have been presented to the network just once. In the training phase, for every epoch, we crop the images and randomly mirror them. So, for every epoch the image is cropped once and might have been mirrored. The position of cropping is random for every epoch and every image. We crop the 256x256 pixels images down to 224x224 pixels for every architecture — except for AlexNet, cropped to 227x227 pixels.

During validation, only a central crop without mirroring is taken. In testing, we resize the images, instead of cropping. While in training and validation we crop to augment data, in testing we only resize to keep the most information of the image.

The chances of only getting background information while cropping are very slim. A plant occupies the most part of the image, and the image have its area reduced to a maximum of 24% and this is most border area.

A per-pixel mean subtraction operation is done before forwarding these images.

6.3.4 Parameters

For AlexNet and CaffeNet, the Local Response Normalization had the following parameters: Local Size $= 5$, $\alpha = 0.0001$, and $\beta = 0.75$. The biases had an initial value of 0.1, and the weights were initialized by using a Gaussian with $\sigma = 0.01$. GoogLeNet and Inception used the Glorot & Bengio (2010) method for weight initialization, and biases initialized to 0.2. He et al. (2015) initialization was used for ResNets and networks with ELU.

For the activation functions, the a parameter of PReLU was set to 0. The α parameter of ELU was initially set to 1, then we also tested for the values of 0.5 and 0.1, because of convergence problems.

The batch sizes were 100 for AlexNet, 256 for CaffeNet, 128 for CaffeNet-PReLU, 32 for GoogLeNet, 12 for Inception, and 6 for ResNet-50. Because of the GPU fixed memory and deeper models occupying more memory, we had to reduce the batch size accordingly.

We trained all networks for 30 epochs, using Stochastic Gradient Descent (SGD) with a step policy that every 10 epochs the learning rate was divided by ten. The initial learning rate was of 0.01, the weight decay of 0.0005, and momentum to 0.9.

The Fined-tuned SVM models were trained separate from the neural network. First, for every fine-tuned neural network, we saved the descriptors of the penultimate layer for every training image. Then, we trained a linear SVM on these descriptors. The penalty parameter was set to $C = 1.0$, we used the squared hinge loss function, L^2 normalization, one vs. rest classification and 1,000 iterations, which is the default of the *scikit-learn* framework (PEDREGOSA et al., 2011; FAN et al., 2008).

6.4 EVALUATION METRICS

All metrics used weigh on the number of species images, thus, a class with more images have a bigger impact on the overall results. There are different ways of weighting, but we chose this one because recall is equivalent to accuracy, and accuracy is a common metric in the machine learning field. The importance is to keep consistency when comparing different architectures, by choosing always the same metric.

Consider that $\mathcal{Y}$ is the set of predicted pairs, $\hat{\mathcal{Y}}$ the set of true pairs, $\mathcal{L}$ the set of labels, $\mathcal{Y}_l$ the subset of $\mathcal{Y}$ with label l, and $\hat{\mathcal{Y}}_l$ the subset of $\hat{\mathcal{Y}}$ with label l, we define P and R as:

$$P(\mathcal{Y}_l, \hat{\mathcal{Y}}_l) = \frac{|\mathcal{Y}_l \cap \hat{\mathcal{Y}}_l|}{|\mathcal{Y}|}, \tag{6.1}$$

$$R(\mathcal{Y}_l, \hat{\mathcal{Y}}_l) = \frac{|\mathcal{Y}_l \cup \hat{\mathcal{Y}}_l|}{|\hat{\mathcal{Y}}_l|}. \tag{6.2}$$

If $\mathcal{Y}_l$ or $\hat{\mathcal{Y}}_l$ is empty, consider $P = 0$ or $R = 0$, respectively. Let $p(r)$ be the precision as a function of recall, the following equations calculate the precision, recall and mean average precision:

$$\text{Precision} = \frac{1}{\sum_{t\in\mathcal{L}} |\hat{\mathcal{Y}}_t|} \sum_{l\in\mathcal{L}} |\hat{\mathcal{Y}}_l| P(\mathcal{Y}_l, \hat{\mathcal{Y}}_l), \tag{6.3}$$

$$\text{Recall} = \frac{1}{\sum_{t\in\mathcal{L}} |\hat{\mathcal{Y}}_t|} \sum_{l\in\mathcal{L}} |\hat{\mathcal{Y}}_l| R(\mathcal{Y}_l, \hat{\mathcal{Y}}_l), \tag{6.4}$$

$$\text{mAP} = \frac{1}{\sum_{t\in\mathcal{L}} |\hat{\mathcal{Y}}_t|} \sum_{l\in\mathcal{L}} |\hat{\mathcal{Y}}_l| \int_0^1 p_l(r_l)\mathrm{d}r_l. \tag{6.5}$$

6.5 COMPUTER SPECIFICATIONS AND TRAINING TIME

The computer used is an Intel® Core™ i5-4440 CPU @ 3.10GHz, 8GB RAM DDR3 @ 1.6GHz, NVIDIA GTX 980 Ti 6GB, SanDisk SSD 128 GB. The operating system is Ubuntu 12.04 LTS and Caffe was compiled using CUDA 7.5, CuDNN, and Intel MKL. The elapsed time for training of some models can be seen at Table 6.5.

Table 6.5: Recorded elapsed time for training only of some classifiers (h:mm:ss)

Classifier	Plantas50Basic	Plantas50Extended
AlexNet-Default	0:12:32	0:42:47
CaffeNet-Default	0:10:49	0:44:50
CaffeNet-ELU	0:10:19	0:39:52
CaffeNet-Finetuned	0:10:40	0:39:57
CaffeNet-Finetuned-Zero	0:14:01	0:41:29
CaffeNet-PReLU	0:12:01	0:41:35
GoogLeNet-Default	0:18:55	1:06:41
GoogLeNet-ELU	0:18:52	1:06:33
GoogLeNet-Finetuned	0:18:52	1:06:29
GoogLeNet-Finetuned-ELU	0:18:46	1:06:24
GoogLeNet-Finetuned-PReLU	0:20:47	1:13:30
GoogLeNet-Finetuned-Zero	0:12:15	0:42:58
GoogLeNet-PReLU	0:20:44	1:13:25
Inception-Default	0:50:34	2:58:10
Inception-ELU	0:50:37	2:58:12
Inception-PReLU	0:52:56	3:07:33
ResNet-50-Default	1:41:56	5:59:59
ResNet-50-Finetuned	1:42:00	6:03:21
ResNet-50-PReLU	1:47:36	6:18:11

7 RESULTS AND DISCUSSION

For the specific purpose of classifying plants, the encoding methods are still competitive (Table 7.1). For the Plantas50Basic set, Fisher Vector had an accuracy of 94.4%, better than any deep neural network without fine-tuning, except for Inception-PReLU with 94.9% accuracy. However, for a set with higher variability, such as the Plantas50Extended, the most recent architectures showed better results. Inception and ResNet, in their default configuration, had 88.5% and 81.5% accuracy, which is an improvement over Fisher Vector with 77.5% accuracy. Although, Fisher Vector performed better than earlier architectures such as AlexNet (71.9%) and GoogLeNet (68.6%).

Geometric Augmentation, comparing only using Fisher Vector, decreases accuracy for Plantas50Basic, and almost no effect for Plantas50Extended. A hypothesis, is that for the Basic dataset, the photos taken tries to center on the plant alone, and the plant occupies most of the space of the picture, thus the descriptors taken are similar, and their position might add a position constraint that is unnecessary. In the Extended set, the images are more cluttered, and the position might matter more in order to pack the plant descriptors together, and segregate those that are unrepresentative of the plant (Table 7.1).

Fisher Vector as being calculated on first and second order (mean and variance deviations), showed the importance of higher order of discrimination than VLAD that is calculated on mean deviation only, and Bag of Visual Words that are calculated by counting words and building a histogram, being a zero-order estimator.

One of the major problems that affected deep neural networks was over-fitting. Because these networks have a massive number of parameters, they can almost 'memorize' the whole dataset and fail to generalize to unseen data. And, the smaller the database, the higher chance of over-fitting, and that was one of the concerns with our middle-sized database ($\approx$ 33,000 images). However, the recent advances in regularizing these networks was enough to train our *Plantas* database from scratch with little over-fitting. Over-fitting occurs when the loss decreases but the accuracy stalls. In Figs. 7.1 and 7.2, the validation set accuracy kept increasing, while training loss decreased.

Deeper architectures, such as ResNet, should have learned more discriminative features than Inception for example. However, for both Basic and Extended sets, Inception had better accuracy than ResNet in the Default configuration. This could be explained by an impaired batch statistics because of smaller batch size (6 vs. 12), as this might be the reason of the noisier loss curve for ResNet (Fig. 7.2b). Another reason is that more data are needed to fully exploit the expressivity of deep models, especially because deeper layers (highly discriminative) take a much longer time to learn than shallower layers (DIAS; BORGES, 2016).

Fine-tuned models have much better accuracy. For computer vision applications, the convolutional neural networks learn some features that can be universally used for any application.

Table 7.1: Classifiers Statistics[a] on the Test Set (DIAS; BORGES, 2016)

Classifier	Plantas50Basic					Plantas50Extended				
	mAP	Top 1		Top 5		mAP	Top 1		Top 5	
		Precision	Recall	Precision	Recall		Precision	Recall	Precision	Recall
AlexNet-Default	77.9%	72.1%	72.4%	94.9%	94.8%	77.8%	72.2%	71.9%	92.9%	92.7%
CaffeNet-Default	56.4%	50.1%	52.5%	86.5%	85.6%	71.1%	67.0%	65.7%	90.3%	89.9%
CaffeNet-ELU[b]	69.6%	64.0%	63.5%	92.5%	92.1%	70.7%	66.6%	65.7%	89.6%	89.2%
CaffeNet-Finetuned	99.2%	97.4%	97.3%	99.5%	99.5%	93.7%	88.8%	88.5%	98.0%	97.9%
CaffeNet-Finetuned-Zero	97.3%	93.0%	92.9%	99.6%	99.6%	86.1%	80.0%	79.8%	96.3%	96.2%
CaffeNet-PReLU	83.9%	79.0%	78.3%	95.8%	95.7%	83.8%	78.6%	78.1%	94.3%	94.2%
CaffeNet-Finetuned-SVM	95.9%	94.2%	94.1%	99.5%	99.5%	80.6%	77.9%	77.9%	94.5%	94.5%
GoogLeNet-Default	74.9%	70.8%	71.2%	94.1%	93.9%	72.7%	68.8%	68.6%	91.6%	91.4%
GoogLeNet-ELU[b]	79.6%	76.0%	76.6%	95.6%	95.4%	74.2%	70.8%	70.4%	91.8%	91.5%
GoogLeNet-Finetuned	99.8%	98.4%	98.3%	100%	100%	96.9%	93.8%	93.6%	98.8%	98.8%
GoogLeNet-Finetuned-PReLU	99.9%	99.3%	99.2%	100%	100%	97.6%	94.6%	94.5%	99.2%	99.2%
GoogLeNet-Finetuned-Zero	97.6%	94.5%	94.2%	99.9%	99.9%	86.6%	82.0%	81.8%	95.9%	95.8%
GoogLeNet-PReLU	93.5%	89.4%	89.1%	98.7%	98.6%	87.7%	82.2%	81.7%	95.9%	95.7%
GoogLeNet-Finetuned-SVM	95.1%	94.6%	94.4%	99.5%	99.4%	75.4%	76.3%	76.1%	92.3%	92.2%
Inception-Default	97.6%	93.3%	93.0%	99.5%	99.5%	93.3%	88.7%	88.5%	97.5%	97.5%
Inception-ELU[b]	96.9%	92.6%	92.4%	99.3%	99.3%	90.0%	84.7%	84.2%	96.8%	96.7%
Inception-PReLU	98.1%	94.9%	94.9%	99.5%	99.5%	93.7%	89.0%	88.7%	97.7%	97.7%
ResNet-50-Default	85.0%	87.6%	81.3%	97.6%	93.2%	86.0%	82.7%	81.5%	95.3%	94.7%
ResNet-50-Finetuned	99.7%	98.1%	98.0%	99.9%	99.9%	95.9%	92.2%	92.1%	98.4%	98.4%
ResNet-50-PReLU	85.8%	87.6%	83.3%	96.8%	94.1%	89.6%	84.0%	83.2%	96.3%	96.2%
ResNet-50-Finetuned-SVM	99.4%	98.6%	98.6%	99.9%	99.9%	90.5%	88.3%	88.2%	97.5%	97.5%
ResNet-101-Finetuned-SVM	99.3%	98.7%	98.6%	100%	100%	90.3%	88.4%	88.3%	97.6%	97.6%
ResNet-152-Finetuned-SVM	99.4%	99.2%	99.1%	100%	100%	90.3%	89.2%	89.1%	97.4%	97.4%
BoVW-Aug	75.5%	77.6%	77.5%	95.1%	95.0%	51.6%	53.2%	53.7%	79.7%	79.6%
Fisher Vector	95.7%	94.6%	94.4%	99.7%	99.6%	80.5%	77.5%	77.5%	93.7%	93.6%
Fisher Vector-Aug	94.1%	93.2%	92.9%	99.5%	99.4%	80.1%	77.8%	77.7%	92.5%	92.4%
VLAD-Aug	87.9%	87.2%	87.0%	98.2%	98.1%	61.9%	61.2%	61.0%	84.4%	84.3%

[a] These statistics weigh on test examples for each class. All images of the test set are used in the calculation of these metrics.
[b] For CaffeNet-ELU (Plantas50Basic), $\alpha = 0.5$, for others $\alpha = 0.1$.

It is already known, for example, that the shallower layers learn edge and colors filters, and deeper layers learn more specific representations. Because the network has already learned a several features, when we train it in a new database, there is no need to learn similar features again, and the network proceeds to learn the deeper layers that are more discriminative. This knowledge transfer also impacts the regularization of a network and aids in convergence. Thus, small and middle-sized database can be trained without fears of severe over-fitting.

We adopted three different policies for fine-tuning: Complete, Zero, and SVM. For all architectures tested, Linear SVM performed worst. This mean that despite the pre-loaded parameters having several features, in order to achieve better accuracy, some database specific features also must be learned; and this explain why Complete is superior to Zero as well, it has more parameters to learn something new, or to adapt their current features to be more database specific (DIAS; BORGES, 2016).

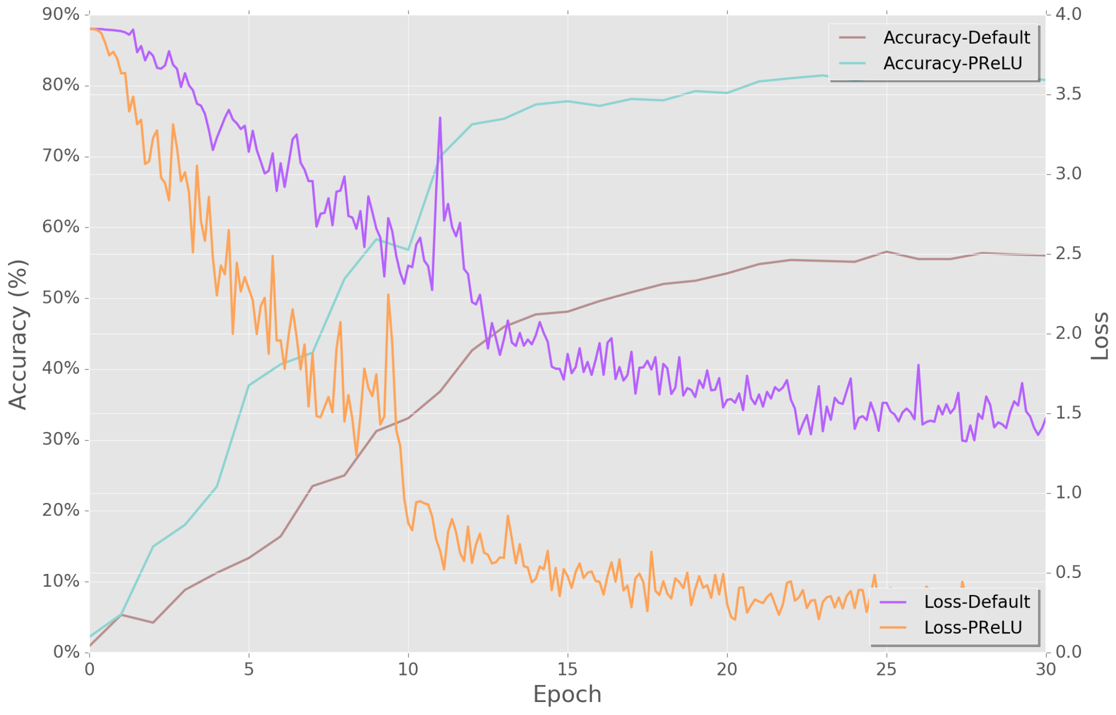
Despite Inception being the best for our *Plantas* database in plain configuration, depth is important for fine-tuning. Deeper networks pre-trained on the 2012 ImageNet Challenge had better results. For instance, take the fine-tuned SVM architectures trained on Plantas50Extended, CaffeNet, GoogLeNet, ResNet-50, ResNet-101, and ResNet-151 has the following accuracies: 77.9%, 76.1%, 88.2%, 88.3%, 89.1%. Deeper networks can learn more distinct features and thus can generalize better to other databases, and we believe that very deep networks trained with massive data can learn enough to be able to transfer this knowledge to any kind of visual classification problem, obtaining high accuracies with little information on the new database (DIAS; BORGES, 2016). However, the existent large-scale database showed signs of over-fitting on very deep networks, such as the ResNet with 1202 layers (HE et al., 2016).

The Parametric Rectified Liner Unit (PReLU) bettered the accuracies in every architecture tested. It was most beneficial to shallower networks, for example, as in Table 7.1, CaffeNet and GoogLeNet improved their accuracies by a large margin compared to the ResNet-50. Empirically, according to He et al. (2015), PReLU parameter a have a larger distance from zero in the first convolutional layer, and because it is composed mostly by Gabor-like filters (edge, texture detectors), the negative and positive output are respected. The trend is that for deeper layers, the a parameters starts going towards zero, thus privileging more non-linear systems. This way, the model is more informative in shallower layers and more discriminative in deeper ones (HE et al., 2015). We might conclude that shallower networks benefit the most because it releases the constraints of their shallow layers to be less informative than it should be; and because deeper networks have more parameters, they are less penalized by having more ‘opportunities’ to distribute throughout the network this informative behavior.

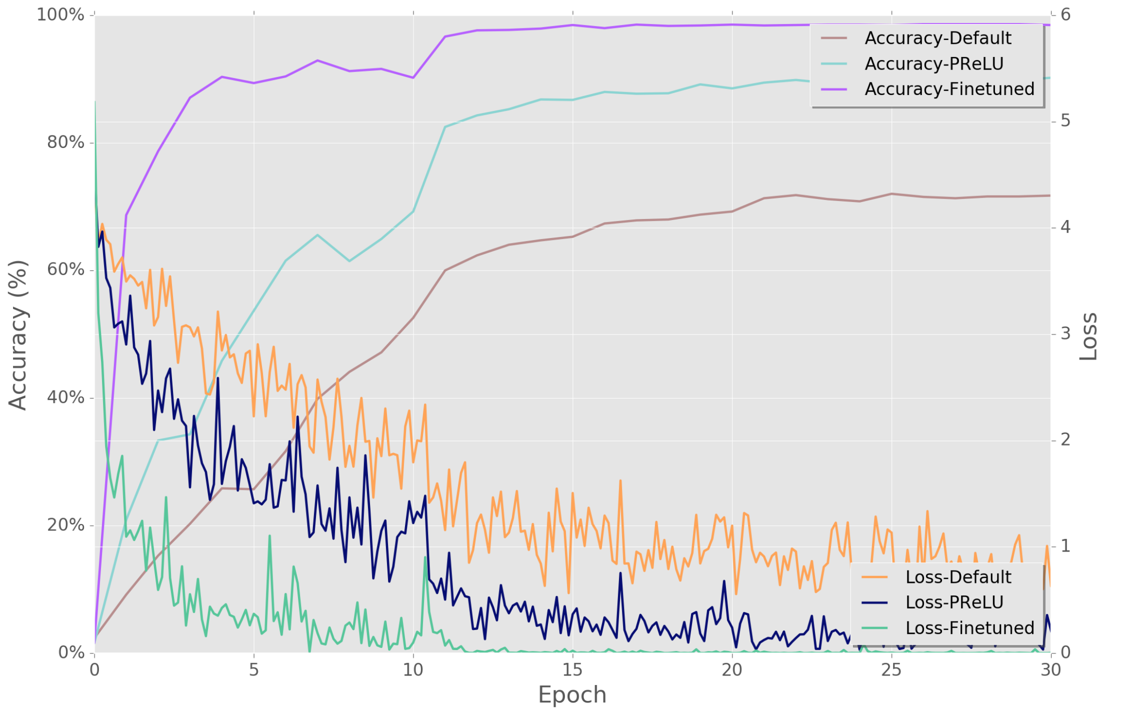
Exponential Rectified Unit (ELU) was tested using CaffeNet, GoogLeNet and Inception. Because ELU have little effect if trained in a network with Batch Normalization (CLEVERT; UNTERTHINER; HOCHREITER, 2015), Inception converged. However, the others diverged using $\alpha = 1$ and weights initialized by *Gaussian*, Glorot & Bengio (2010) and He et al. (2015). CaffeNet converged using $\alpha = 0.5$ and *Gaussian* for Plantas50Basic set, and $\alpha = 0.1$ and *Gaussian* for Plantas50Extended. GoogLeNet converged using $\alpha = 0.1$ and *Xavier*. We could not explore the full possibilities of ELU; however, we can say that higher α gets closer to the accuracy of PReLU and distance itself from ReLU, but we are incapable of testing the capability of ELUs having a higher accuracy in earlier epochs using our current configuration (DIAS; BORGES, 2016).

The images of *Plantas* database was taken during one season (Summer in Brazil) and only by a few people, thus we increased variability by adding images from the Internet of the selected species. However, this is error-prone because we are uncertain that some species were correctly selected, and also some species are much more common than others, and the database became unbalanced. Increased variability did challenge more the classifiers, for example, in the confusion matrix of Fisher Vector (Figs. 7.3 and 7.4) for the Extended set, there is much more confusion than the Basic. However, for our best model, GoogLeNet-Finetuned-PReLU (Figs. 7.5 and 7.6), there are low confusion even for the Extended set, thus, the error of selecting wrong images impaired little. And, although the database is now skewed, accuracy is not inflated by a good classified species with more images than others.

We can also affirm that is little background effect in aiding classification for two reasons: our images display very different background inside the same species, and there is similar background for different species.

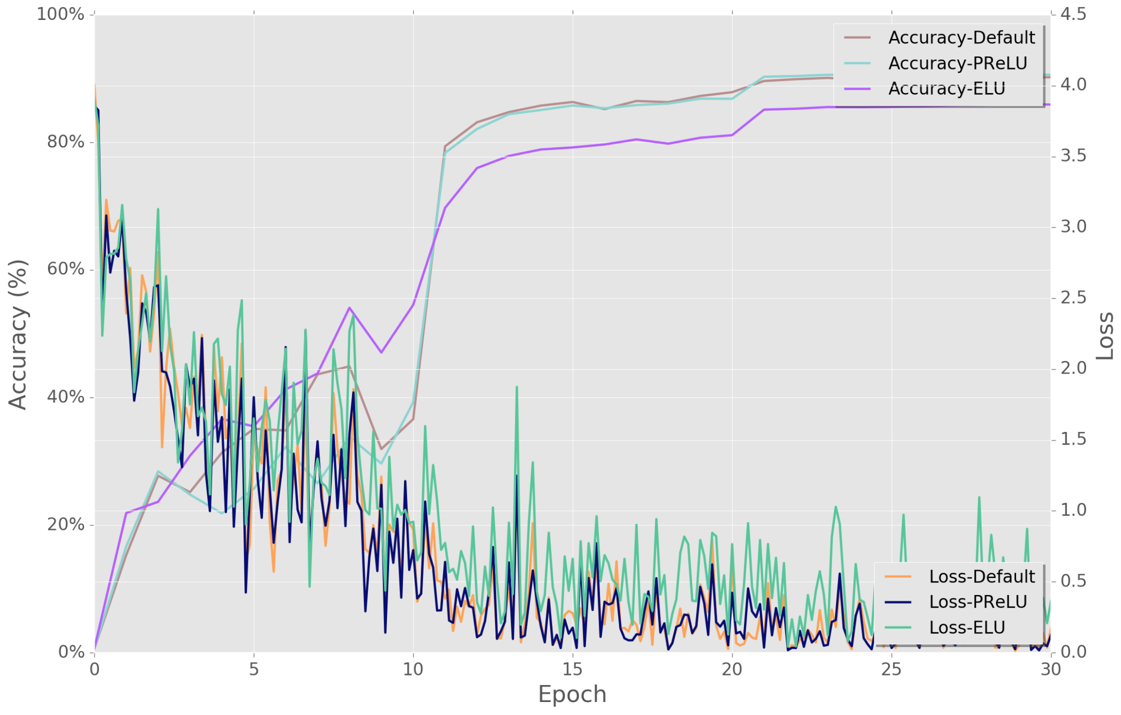


(a) CaffeNet (Plantas50Basic)

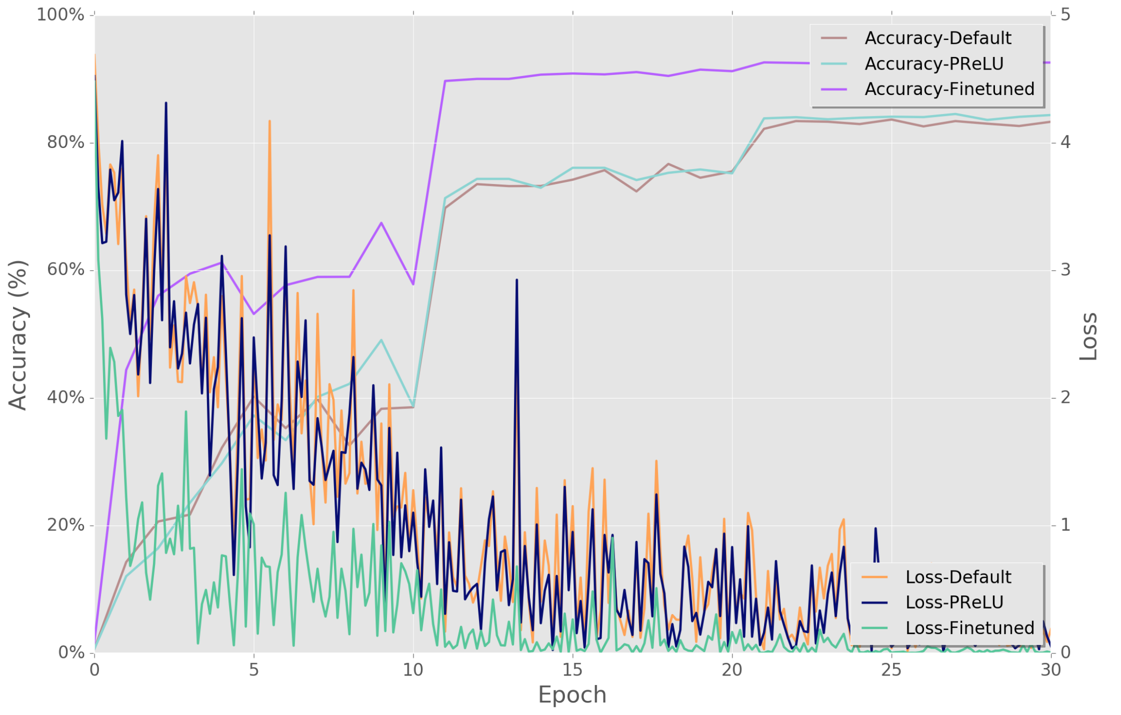


(b) GoogLeNet (Plantas50Basic)

Figure 7.1: Training loss and validation accuracy for 30 epochs and a step learning policy of reducing learning rate by 10 for every 10 epochs. Training of CaffeNet and GoogLeNet (batch size of 256 and 32, respectively) using Plantas50Basic.



(a) Inception (Plantas50Extended)



(b) ResNet-50 (Plantas50Extended)

Figure 7.2: Training loss and validation accuracy for 30 epochs and a step learning policy of reducing learning rate by 10 for every 10 epochs. Training of Inception and ResNet-50 (batch size of 12 and 6, respectively) using Plantas50Extended.

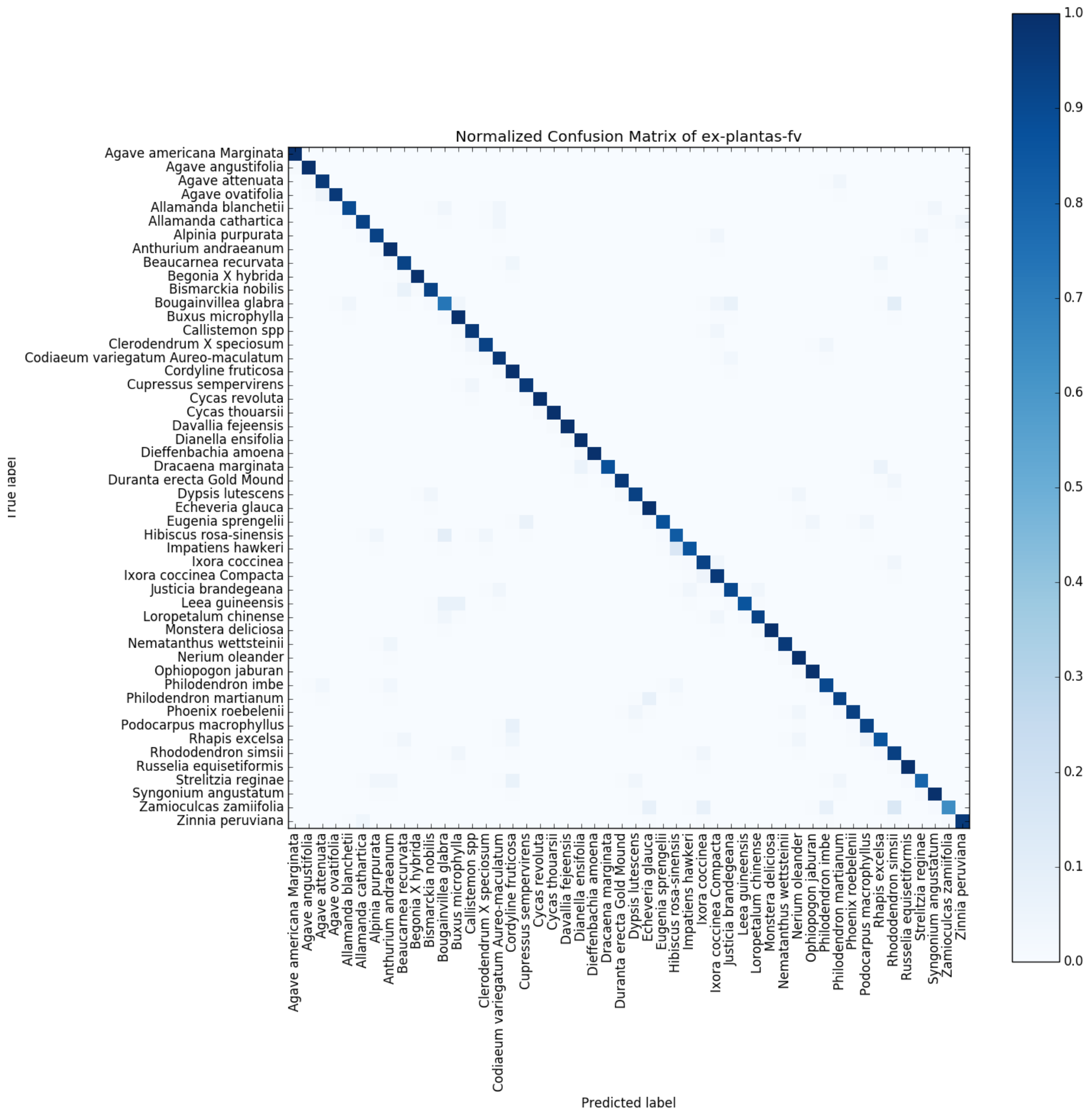


Figure 7.3: Confusion Matrix of Fisher Vector (Plantas50Basic)

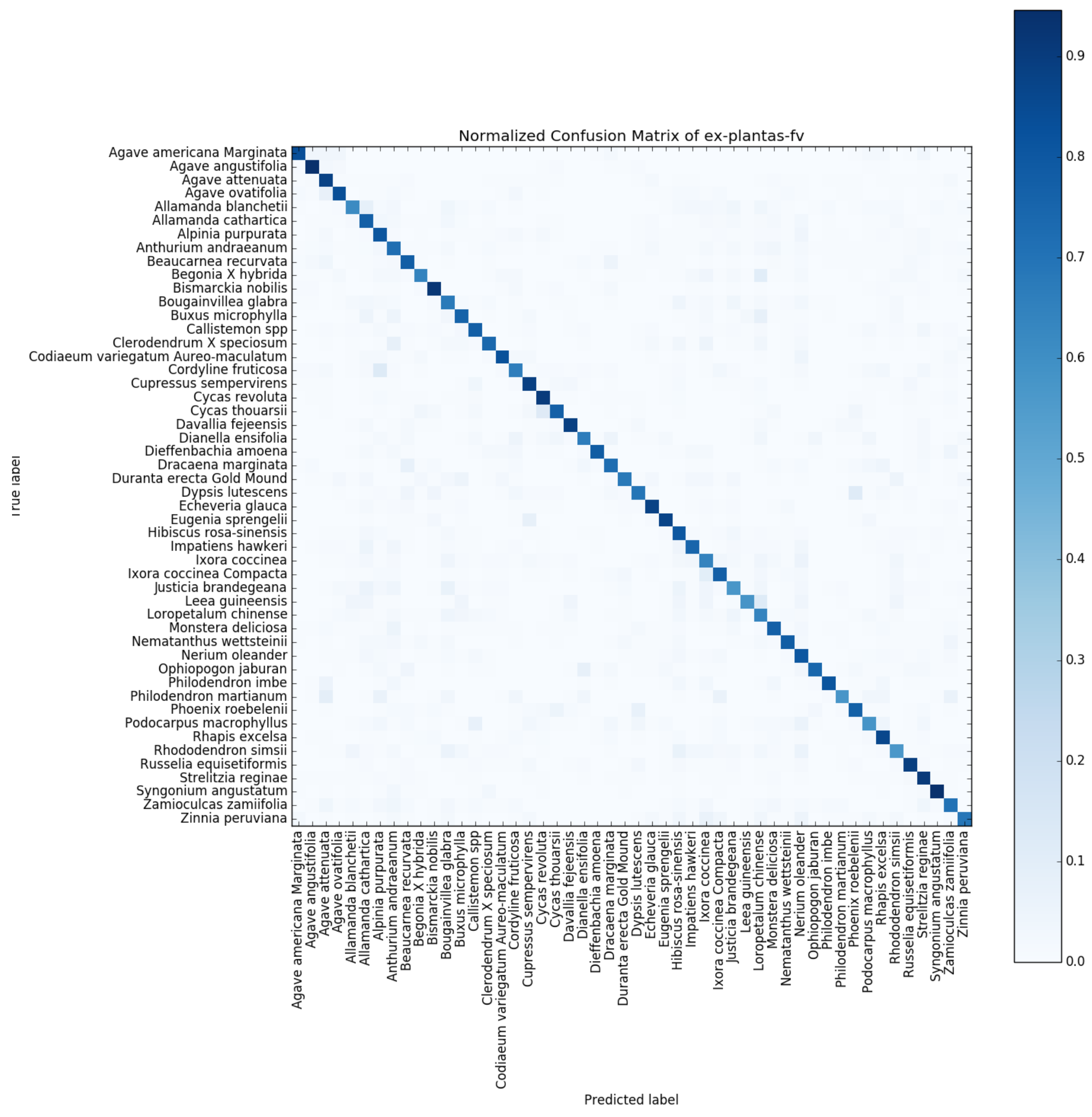


Figure 7.4: Confusion Matrix of Fisher Vector (Plantas50Extended)

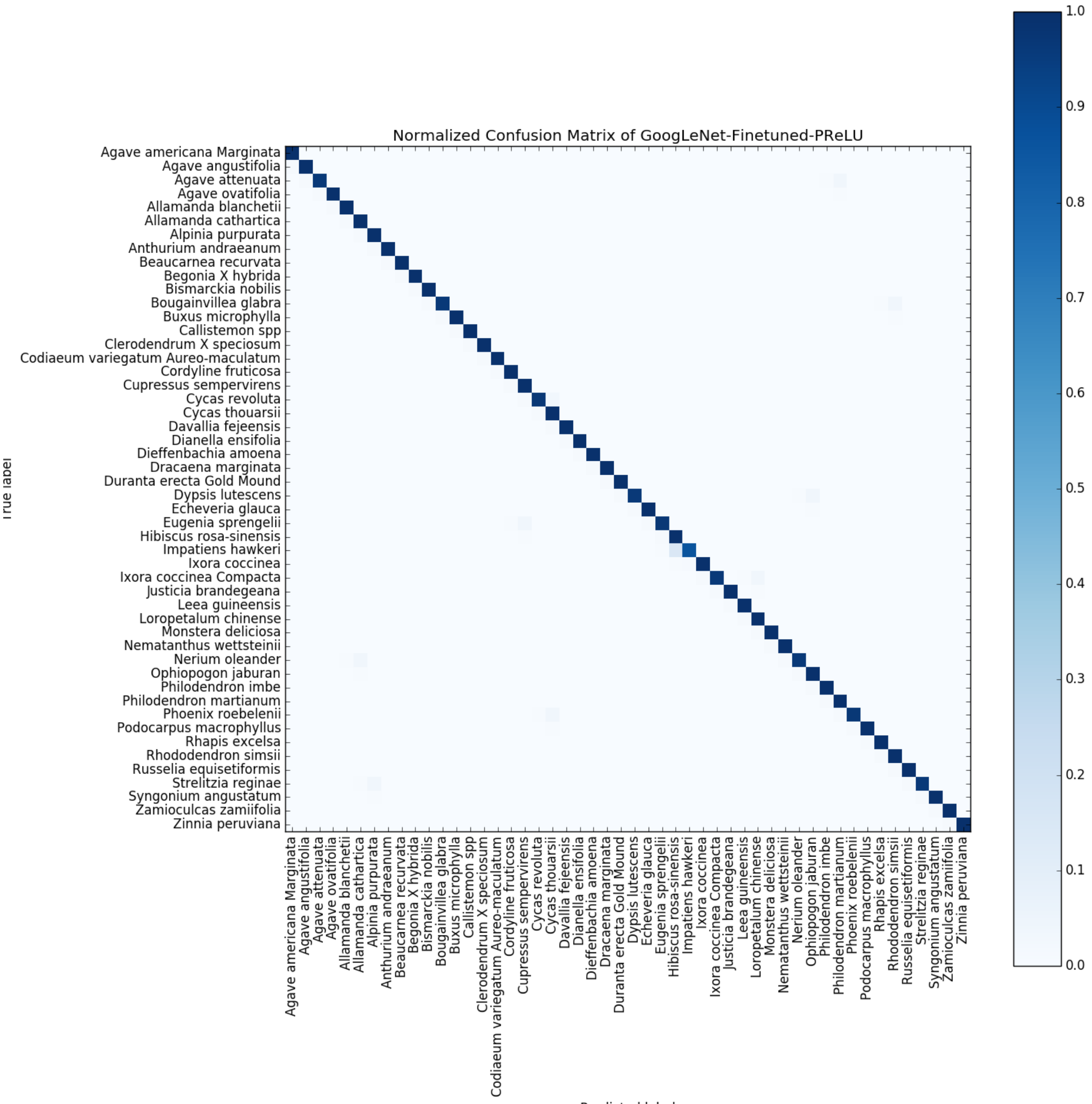


Figure 7.5: Confusion Matrix of GoogLeNet-Finetuned-PReLU (Plantas50Basic)

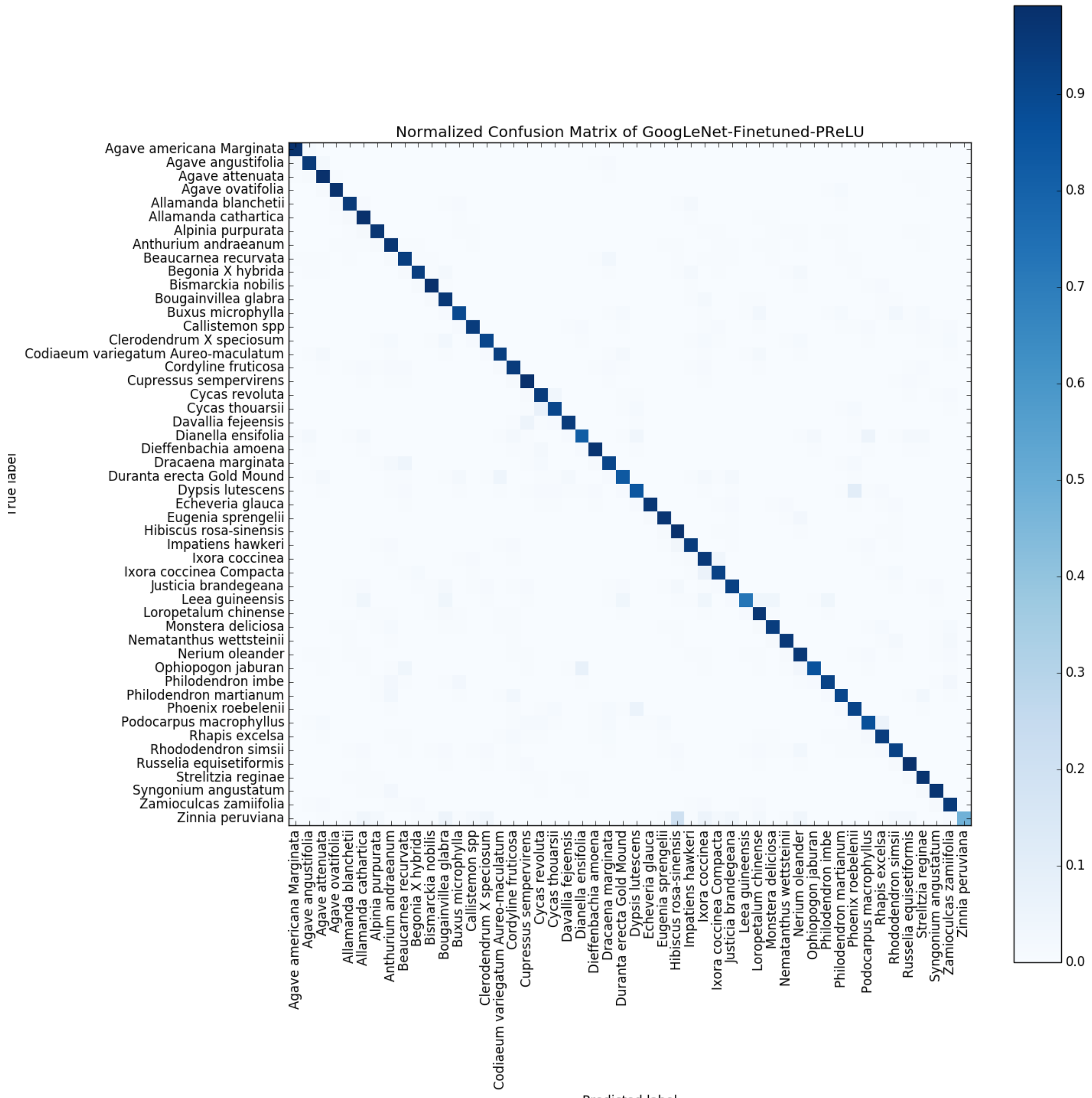


Figure 7.6: Confusion Matrix of GoogLeNet-Finetuned-PReLU (Plantas50Extended)

8 CONCLUSIONS

The *Plantas* database was built with the aim of offering several images of plant species taken in an uncontrolled environment and favoring the point of view of the common user. It contains 50 commonly cultivated species or cultivars, and about 33 thousand images.

The database is divided into three sets: Basic, Extra, and Internet. The first is composed by high-quality images centered on the plants and without much noise or clutter. The Extra contains some discarded images of Basic, but still have recognizable plants. Then, we added the Internet set, which contains images of these plants taken from the Internet and manually curated. This way, we increased variability, however, the database is now more unbalanced in regard to images per species.

Despite the increased unbalance, the classifiers showed little confusion between species, which indicates a high level of discrimination and small impact in accuracy measurement by species with large quantity of images.

The Fisher Vector method showed that descriptors containing high-order features have a great effect in classification, and it has obtained a higher accuracy than Bag of Visual Words and VLAD, which have less complex descriptors.

For non-fine-tuned models, Encoding Methods showed to be very competitive against neural networks, especially Fisher Vector, which have the best result in plain classifiers configuration (94.4% acc.), second only to the Inception with PReLU (94.9% acc.) in modified networks. However, for a more complex dataset (Plantas50Extended), recent deep neural networks surpass by large margins the Fisher Vector method (77.5% acc.). And the best result is Inception-PReLU with 88.7% accuracy.

Fine-tuning is the technique that impacts the most in accuracy increase. For instance, for the Plantas50Extended set, GoogLeNet-Default had an accuracy of 68.6% that jumped to 98.8%; and this happens for every network. The best overall classifier is GoogLeNet-Finetuned-PReLU, with 99.2% accuracy in Plantas50Basic and 94.5% accuracy in Plantas50Extended.

Parametric Rectified Linear Unit (PReLU) benefited all networks, especially the shallower ones. This can be attributed to more flexibility that networks have to learn more informative than discriminative features in its shallow convolutional layers, which absence penalizes shallow network more. We could not assess the Exponential Linear Unit (ELU) because of diverging problem that occurs when $\alpha = 1$.

Comparing to the whole known world of plant species, our *Plantas* database contains only 50 species and thus can be view as a low complexity database and current neural networks can tell them apart with some ease, and that increasing greatly the number of species, these networks might have a harder time differentiating between very similar species. However, we have two examples of very similar plants that the networks made little confusion: *Allamanda blanchetti* and *Allamanda cathartica*; *Ixora coccinea* and *Ixora coccinea 'Compacta'*. Thus, increasing the number of similar species seems to be feasible for current networks.

Our first contribution was to create an image database of plant species more complex than previous databases and with a focus on the common user. We also attest that: this database can be trained using the latest technologies of object recognition, such as deep neural networks and Fisher Vector, and offer reliable classifiers for future deployment; Fisher Vector method is competitive for classifying our database; fine-tuning models with pre-trained weights on general-purpose database improves our specific-purpose models; Parametric Rectified Linear Unit improves the tested models trained with our database; our database suffered little overfitting.

Topics for further exploration, beyond expanding the number of species, are: to develop more lightweight networks for mobile deployment while maintaining accuracy, and explore structural learning using the taxonomy tree and the known hierarchical learning that occurs in deep neural networks.

REFERENCES

BELHUMEUR, P.; CHEN, D.; FEINER, S.; JACOBS, D.; KRESS, W.; LING, H.; LOPEZ, I.; RAMAMOORTHI, R.; SHEOREY, S.; WHITE, S.; ZHANG, L. Searching the world's herbaria: A system for visual identification of plant species. In: . [s.n.], 2008. p. 116–129. Disponível em: <http://graphics.cs.berkeley.edu/papers/Ramamoorthi-STW-2008-10/>.

BOYKOV, Y. Y.; JOLLY, M. P. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. [S.l.: s.n.], 2001. v. 1, p. 105–112 vol.1.

CERUTTI, G.; TOUGNE, L.; VACAVANT, A.; COQUIN, D. A parametric active polygon for leaf segmentation and shape estimation. In: *7th International Symposium on Visual Computing*. Las Vegas, United States: [s.n.], 2011. p. 1. Disponível em: <https://hal.archives-ouvertes.fr/hal-00622269>.

CHOROMANSKA, A.; HENAFF, M.; MATHIEU, M.; AROUS, G. B.; LECUN, Y. The loss surface of multilayer networks. *CoRR*, abs/1412.0233, 2014. Disponível em: <http://arxiv.org/abs/1412.0233>.

CLEVERT, D.; UNTERTHINER, T.; HOCHREITER, S. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. Disponível em: <http://arxiv.org/abs/1511.07289>.

DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, v. 39, n. 1, p. 1–38, 1977.

DENG, J.; DING, N.; JIA, Y.; FROME, A.; MURPHY, K.; BENGIO, S.; LI, Y.; NEVEN, H.; ADAM, H. Large-scale object classification using label relation graphs. In: *Computer Vision–ECCV 2014*. [S.l.]: Springer International Publishing, 2014. p. 48–64.

DIAS, R. O. Q.; BORGES, D. L. Recognizing plant species in the wild: deep learning results and a new database. In: *2016 IEEE International Symposium on Multimedia (ISM)*. Los Alamitos, CA: IEEE Computer Society, 2016. p. 197–202. ISBN 978-1-5090-4571-6/16. Disponível em: <https://doi.org/10.1109/ISM.2016.0047>.

ELKAN, C. Using the triangle inequality to accelerate k-means. In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. AAAI Press, 2003. (ICML'03), p. 147–153. ISBN 1-57735-189-4. Disponível em: <http://dl.acm.org/citation.cfm?id=3041838.3041857>.

ELLIS, J. *Geeqie Image Viewer - Similarity Function*. 2004. <https://github.com/BestImageViewer/geeqie/blob/master/src/similar.c>.

FAN, R.; CHANG, K.; HSIEH, C.; WANG, X.; LIN, C. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, v. 9, p. 1871–1874, 2008. Disponível em: <http://jmlr.org/papers/volume9/fan08a/fan08a.pdf>.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256. Disponível em: <http://proceedings.mlr.press/v9/glorot10a.html>.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: GORDON, G. J.; DUNSON, D. B. (Ed.). *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011. v. 15, p. 315–323. Disponível em: <http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf>.

GOËAU, H.; BONNET, P.; JOLY, A. LifeCLEF Plant Identification Task 2015. In: CEUR-WS (Ed.). *CLEF 2015*. toulouse, France: [s.n.], 2015. (CLEF2015 working notes, v. 1391). Disponível em: <https://hal.inria.fr/hal-01182795>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. Washington, DC, USA: IEEE Computer Society, 2015. (ICCV ’15), p. 1026–1034. ISBN 978-1-4673-8391-2. Disponível em: <http://dx.doi.org/10.1109/ICCV.2015.123>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 770–778.

HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. Disponível em: <http://arxiv.org/abs/1207.0580>.

HOCHREITER, S. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 1991.

HOWARD, A. G. Some improvements on deep convolutional neural network based image classification. *CoRR*, abs/1312.5402, 2013. Disponível em: <http://arxiv.org/abs/1312.5402>.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. Disponível em: <http://arxiv.org/abs/1502.03167>.

JÉGOU, H.; DOUZE, M.; SCHMID, C.; PÉREZ, P. Aggregating local descriptors into a compact image representation. In: *CVPR 2010 - 23rd IEEE Conference on Computer Vision & Pattern Recognition*. San Francisco, United States: IEEE Computer Society, 2010. p. 3304–3311. Disponível em: <https://hal.inria.fr/inria-00548637>.

JIA, Y.; SHELHAMER, E.; DONAHUE, J.; KARAYEV, S.; LONG, J.; GIRSHICK, R.; GUADARRAMA, S.; DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGES, C.; BOTTOU, L.; WEINBERGER, K. (Ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012. p. 1097–1105. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

KUMAR, N.; BELHUMEUR, P. N.; BISWAS, A.; JACOBS, D. W.; KRESS, W. J.; LOPEZ, I.; SOARES, J. V. B. Leafsnap: A computer vision system for automatic plant species identification. In: *The 12th European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2012.

LAGA, H.; KURTEK, S.; SRIVASTAVA, A.; GOLZARIAN, M.; MIKLAVCIC, S. J. A riemannian elastic metric for shape-based plant leaf classification. In: *Digital Image Computing Techniques and Applications (DICTA), 2012 International Conference on*. [S.l.: s.n.], 2012. p. 1–7.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 521, n. 7553, p. 436–444, 05 2015. Disponível em: <http://dx.doi.org/10.1038/nature14539>.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 1, n. 4, p. 541–551, dez. 1989. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1989.1.4.541>.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, v. 60, n. 2, p. 91–110, 2004. ISSN 1573-1405. Disponível em: <http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94>.

NILSBACK, M. E.; ZISSERMAN, A. A visual vocabulary for flower classification. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. [S.l.: s.n.], 2006. v. 2, p. 1447–1454. ISSN 1063-6919.

NILSBACK, M.-E.; ZISSERMAN, A. Automated flower classification over a large number of classes. In: *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*. [S.l.: s.n.], 2008.

OQUAB, M.; BOTTOU, L.; LAPTEV, I.; SIVIC, J. Learning and transferring mid-level image representations using convolutional neural networks. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2014. (CVPR '14), p. 1717–1724. ISBN 978-1-4799-5118-5. Disponível em: <http://dx.doi.org/10.1109/CVPR.2014.222>.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PERRONNIN, F.; DANCE, C. Fisher kernels on visual vocabularies for image categorization. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2007. p. 1–8. ISSN 1063-6919.

PERRONNIN, F.; SÁNCHEZ, J.; MENSINK, T. Improving the fisher kernel for large-scale image classification. In: ____. *Computer Vision – ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 143–156. ISBN 978-3-642-15561-1. Disponível em: <http://dx.doi.org/10.1007/978-3-642-15561-1_11>.

ROSKOV, Y.; ABUCAY, L.; ORRELL, T.; NICOLSON, D.; BAILLY, N.; KIRK, P.; BOURGOIN, T.; DEWALT, R.; DECOCK, W.; WEVER, A. D.; NIEUKERKEN, E. v.; ZARUCCHI, J.; PENEV, L. (Ed.). *Species 2000 & ITIS Catalogue of Life, 30th April 2017*. 2017. Species 2000: Naturalis, Leiden, the Netherlands. ISSN 2405-8858. Disponível em: <www.catalogueoflife.org/col>.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, v. 61, p. 85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

SERMANET, P.; EIGEN, D.; ZHANG, X.; MATHIEU, M.; FERGUS, R.; LECUN, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. Disponível em: <http://arxiv.org/abs/1312.6229>.

SHALEV-SHWARTZ, S.; ZHANG, T. Stochastic dual coordinate ascent methods for regularized loss. *J. Mach. Learn. Res.*, JMLR.org, v. 14, n. 1, p. 567–599, fev. 2013. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=2502581.2502598>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. Disponível em: <http://arxiv.org/abs/1409.1556>.

SÖDERKVIST, O. *Computer Vision Classification of Leaves from Swedish Trees*. Dissertação (Mestrado) — Linköping University, 581 83 Linköping, Sweden, 2001.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, v. 15, p. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. Disponível em: <http://arxiv.org/abs/1409.4842>.

TANG, Y. Deep learning using support vector machines. *CoRR*, abs/1306.0239, 2013. Disponível em: <http://arxiv.org/abs/1306.0239>.

VARMA, M.; ZISSERMAN, A. Classifying images of materials: Achieving viewpoint and illumination independence. In: ____. *Computer Vision — ECCV 2002: 7th European Conference on Computer Vision Copenhagen, Denmark, May 28–31, 2002 Proceedings, Part III*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 255–271. ISBN 978-3-540-47977-2. Disponível em: <http://dx.doi.org/10.1007/3-540-47977-5_17>.

VEDALDI, A.; FULKERSON, B. *VLFeat: An Open and Portable Library of Computer Vision Algorithms*. 2008. <http://www.vlfeat.org/>.

WU, S. G.; BAO, F. S.; XU, E. Y.; WANG, Y. X.; CHANG, Y. F.; XIANG, Q. L. A leaf recognition algorithm for plant classification using probabilistic neural network. In: *Signal Processing and Information Technology, 2007 IEEE International Symposium on*. [S.l.: s.n.], 2007. p. 11–16.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. Disponível em: <http://arxiv.org/abs/1311.2901>.